Proceedings of the Fourth IN5550 Workshop on Neural Natural Language Processing (WNNLP 2022)

Andrey Kutuzov, Roxana Pop, David Samuel,
Erik Velldal, and Lilja Øvrelid (Editors)

June 09, 2022

University of Oslo, Norway

Published by

Language Technology Group
Department of Informatics
University of Oslo

Preface

We are delighted to present the proceedings of the Fourth IN5550 Teaching Workshop on Neural Natural Language Processing (WNNLP 2022). Spurred by great advancements in neural approaches to NLP, this is the fourth in a series of successful annual workshops, each showcasing some of the best efforts madeby the Oslo MSc students in Language Technology completing the IN5550 class—all tackling modern NLP research tasks. We are thankful to the Department of Informatics for hosting the workshop, but we are also thankful to Norwegian internet service providers.

The workshop received sixteen submissions (by 24 authors), of which all have been accepted for publication as part of the WNNLP 2022 proceedings (this volume).

This programme would not have been possible without the assistance of all our reviewers, whose careful and constructive feedback has been an important element in finalising the individual contributions. To encourage the spirit of good peer review, we have made the decision to include an Outstanding Reviewer award in this workshop, in addition to the traditional Best Paper award. Further, to emphasise the ties this workshop has had to the IN5550 class, we have also made the decision to include an Outstanding Coursework award.

The Programme Committee has selected the paper *Neural Machine Translation for Restaurant Reviews*, by William Ho, for the WNNLP 2022 Best Paper award, reflecting a combination of thoughtful engineering, in-depth experimentation, and solid analysis.

Among the pool of wonderful WNNLP 2022 reviewers, the Programme Committee finds that one deserves a special mention, for providing especially detailed and constructive feedback to their peers. The recipient of the WNNLP 2022 Outstanding Reviewer award is Marie Fleisje.

And finally, while the coursework that led to this workshop may seem like a distant memory, we award the Outstanding Coursework awards to Tellef Seierstad. Congratulations to all award recipients (and runners-up)!

And, last - but not the least - warmest thanks to all participants of this workshop, who spent many a sleepless nights working on the projects that are certain to make WNNLP 2022 an exciting and stimulating event!

Lilja Øvrelid
WNNLP 2022 General Chair
Oslo; June 09, 2022

Programme Committee

Rohullah Akbari
Herman Brunborg
Helene Bøsei Olsen
Ece Cetinoglu
Marie Emerentze Fleisje
Anastasiia Grishina
Syed Zohaib Hassan
William Ho
Matias Jentoft
Liang Jia
Andrey Kutuzov
Anna Palatkina
Roxana Pop
Sigrid Riiser Kristiansen
Egil Rønningstad
David Samuel
Tellef Seierstad
Elina Sigdel
Tim Sigl
Henrik Syversen Johansen
Aksel Tjønn
Lucas Wagner
Annika Willoch Olstad
Alexandra Wittemann
Qinghua Xu
Huiling You
Thomas Richard Zemp
Jan Šamánek
Erik Velldal
Lilja Øvrelid

Neural Machine Translation Chairs

David Samuel

Semantic Parsing Chairs

Roxana Pop

Word Sense Induction Chairs

Andrey Kutuzov

Targeted Sentiment Analysis Chairs

Egil Rønningstad
Erik Velldal

3

Table of Contents

Exploring the Effect of Hyper-parameter Tuning and Variation of Tag Schemes for Targeted Sentiment Analysis

Marie Emerentze Fleisje marieefl@ifi.uio.no
Annika Willoch Olstad annikaol@ifi.uio.no
Alexandra Wittemann alexankw@ifi.uio.no

Abstract

In this paper, we explore how the use of various tag schemes affects the performance of a targeted sentiment analysis (TSA) model. Utilizing hyper-parameter tuning we first identify which values of the tuning of hyper-parameters improve the given baseline (BiLSTM) model the most, before applying the best model to data sets with various BIO-tag schemes. The result is an improved TSA-model, where the performance is clearly dependent on the choice of BIO-tags. In addition, we provide a short error analysis along with a discussion of our findings.

1 Introduction

Whereas the sentiment analysis (SA) task consists of identifying opinions and feelings in textual data (Pang et al., 2008), targeted sentiment analysis aims to detect both a *target* entity towards which a sentiment is expressed, and the polarity of this sentiment (Mitchell et al., 2013). The system performing TSA must thus complete two tasks:

1. Detect targeted text spans in the data.
2. Identify the polarities of sentiment expressed towards the detected entities from step 1. The polarity can be either negative or positive.

Table 1 displays an example sentence from the NoReC$_{fine}$ train set. This illustrates how several sentiments can be expressed in the same sentence, with the opinions having various targets. TSA captures these differences in sentiment, as opposed to more general SA approaches which would not provide such fine-grained analysis. Being able to more precisely identify the sentiment expressed in text is thus a motivation for the use of TSA.

(1) *Uten Susanne Sundfør hadde ikke dette vært all verdens låt .*
Without Susanne Sundfør had not this been all world's track .
'Without Susanne Sundfør the track would not be all that great .'

Token	Gold label
Uten	O
Susanne	B-targ-Positive
Sundfør	I-targ-Positive
hadde	O
ikke	O
dette	O
vært	O
all	O
verdens	O
låt	B-targ-Negative
.	O

Table 1: Example sentence from NoReC$_{fine}$ train set. See section 3.1 for explanation of tags.

In this work, we compare the performance of a baseline model[1], provided by the lecturers of the IN5550 course at the University of Oslo, to the results obtained by tuning its hyper-parameters. The baseline model is a Bidirectional LSTM (BiLSTM) and is further described in section 3.2. Furthermore, we experiment with various tag schemes (see section 3.1), to determine whether the choice of tags affects the performance of our model.[2] Finally, we perform an error analysis, aiming to identify central errors of the models, which in turn will allow for improvement of the models in future work.

2 Data

In this paper, we make use of the NoReC$_{fine}$ data set, which is annotated with a BIO-format for the use

[1] Available here: https://github.uio.no/in5550/2022/tree/main/exam/targeted_sa

[2] The code for this paper is available here: https://github.uio.no/annikaol/IN5550-exam

	Train	Dev.	Test	Total
Sentences	8634	1531	1272	11437
Targets	5044	877	735	6656

Table 2: Distribution of sentences and targets as number of examples in the NoReC$_{fine}$ data set (based on Øvrelid et al. (2020) and the numbers provided in the exam assignment text[3]).

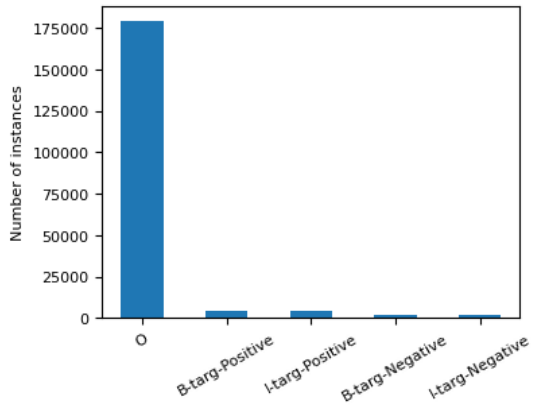


Figure 1: Distribution of labels in the full NoReC$_{\text{fine}}$ dataset. O is by far most common. Positive labels occur roughly twice as frequently as negative labels.

of fine-grained sentiment analysis (Øvrelid et al., 2020). It thus contains annotations of both the polarity and target of sentiment, as illustrated in table 1. The translation of this sentence is given in example 1. The distribution of sentences and targets across the three data splits (train, dev and test), as reported by the exam assignment text [3], is repeated in table 2.

The data is provided to us in conll-format through the GitHub repository of the IN5550 course. [4]

3 Tag schemes and Architectures

3.1 Tag schemes

The NLP society recognizes several different tag schemes for named entity tasks. They differ in complexity and ability, and thus reveal themselves as ideal for various named entity tasks (Alshammari and Alanazi, 2021). In this paper, we will address and train our model with four different tag schemes.

The provided dataset has a total of 5 labels (B-targ-Positive, I-targ-Positive, B-targ-Negative, I-targ-Negative, O) as seen in table 1, this coincides with the popular IOB tag scheme, also referred to as BIO. Alongside this we will also consider the simple IO scheme which assigns only an **I**nside or **O**utside label to a token, and the more complex BIOL and BIOUL schemes where the labels correspond to respectively **B**eginning, **I**nside, **O**utside, **U**nique and **L**ast. The **U** and **L** labels are assigned to respectively a single-token unit and the last token in a named entity.

The main difference between the BIO, BIOL and BIOUL schemes and the IO, is that the IO scheme is unable to recognize consecutive entities. So even though the IO outperforms most other tag schemes (including BIO, BIOL and BIOUL) in terms of metrics, Alshammari and Alanazi (2021) claims it's not fair to compare it to the others due to the skewness in ability.

We want to explore not only the measureable difference, but also the qualitative variation within and between the different tag schemes with differently tuned models and will thus also include the IO scheme in our experiments.

3.2 BiLSTM models

The LSTMs, long short-term memory models, as introduced by Hochreiter and Schmidhuber (1997) aimed to solve the vanishing gradient problem seen with RNNs (Pascanu et al., 2013), where the gradient in gradient based learning methods gets so small that the weights do not update and the training stops as the model is unable to learn correlation between events that occur over time because the information fades or *vanishes*.

The benefit of applying the LSTM architecture is the controlled gating mechanism which allows to regulate the flow of information into and out of the cell, where gradients can proceed even unchanged into the next state. There are three gates in the LSTM architecture that each handle the information passed from one step to the next; the *forget-*, *input-* and *output gate*.

[3] Available here: https://github.uio.no/in5550/2022/blob/main/exam/targeted_sa/5550_TSA.pdf

[4] Available here: https://github.uio.no/in5550/2022/tree/main/exam/targeted_sa/data

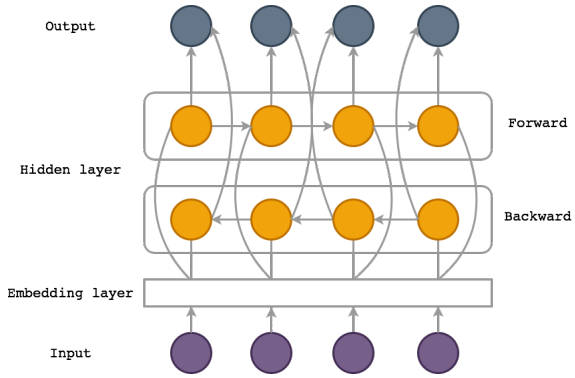


Figure 2: Bidirectional LSTM architecture based on the one proposed in Huang et al. (2015).

When passed through the forget gate the information is controlled in terms of deciding on how much of the previous memory should be kept, and what can be ignored (or *forgotten*). The input gate decides what relevant information can be added from the current step, and the output gate finalizes the next hidden state.

A bidirectional LSTM (*BiLSTM*), whose architecture is illustrated in figure 2, consists of two LSTMs where one handles the forward phase and one handles the backwards phase. The baseline model for this project is a simple BiLSTM with one hidden layer. The benefit of using a bidirectional over a unidirectional LSTM is that each component processes information going both forward and backwards, thus producing a more fitting prediction.

So even though BiLSTMs can be computationally expensive to train (Goldberg, 2017), they have proven to perform especially well with sequence classifying tasks like this one.

4 Evaluation

For this specific task we will benefit from relying mostly on proportional F_1-score in the evaluation for the most fair and balanced measure of performance. This assumes a measure of precision and recall as well. Proportional F_1-score, as opposed to binary F_1-score, paints a better picture of how large a part of the predicted entities coincide with the gold entities while binary F_1-score tell us only whether there is overlap between the predicted entities and the gold entities, or not.

The main argument for using such metrics instead of accuracy is the large class imbalance of this dataset, see figure 1. Using accuracy as main form of evaluation would disturb the impression of performance a lot due to the large number of **O**utside labels in the dataset. Thus if most **O**utside labels are assigned correctly, the accuracy would be artificially high, even if all the other labels are assigned incorrectly.

As for the models, we will run each model five times to calculate average performance. We use proportional F_1-score to select the best model as it is a stricter measure than binary F_1-score. However we calculate and report both for a more nuanced analysis.

5 Experiments

Initially, we run the baseline model with its provided default hyper-parameters. This model has one hidden layer with a dimension of 100. It is trained with a learning rate of 0.01 and batch size of 50 for 50 epochs. Dropout is set to 0.01 and the embeddings are not tuned. The embeddings used for the training task are embeddings number 58 from the NLPL word embeddings repository[5]. The embeddings have been trained on the Norwegian-Bokmaal CoNLL17 corpus using the Word2Vec Continuous Skipgram algorithm, a window size of 10 and a dimension of 100. The vocabulary is non-lemmatized and contains approximately 1.18 million word types.

We tune the baseline model with the goal of finding a combination of hyper-parameter values that yields better results than the baseline. This tuning process consists of two parts. In the first round, we tune the size of the hidden layer, the number of hidden layers, learning rate and number of epochs. We keep the parameters values yielding the best results from this round. These are used in the second part of tuning, where batch size is tuned together with the option of training the embeddings or keeping them frozen. We again keep the best settings, and these define our best model with the standard BIO tag scheme.

Last, we investigate the effect of the alternative tag schemes BIOUL, BIOL and IO. In these experiments, we use the settings yielding the best results for the BIO tag scheme. Our hypothesis is that the simple IO tag scheme will outperform the others

[5] http://vectors.nlpl.eu/repository/#

because its low number of tags (3) makes it more likely for the model to choose the correct tag by chance. Correspondingly, one can expect BIOUL to perform more poorly due to its high number of unique tags (9). Note, however, that this is partly due to the metrics we use. A metric only counting exact entity-level matches as correct might benefit from the more fine-grained information contained in the BIOUL labeling scheme.

In all experiments, every combination of parameter values is run 5 times. The results for both evaluation metrics are averaged across these runs. Standard deviation is computed and reported for the best model. The baseline model and the tuned models with different tag schemes are evaluated on the held-out test set for comparison.

6 Results

The following subsections report, and briefly discuss, the results of the experiments described in section 5. The results are presented in tables, according to the evaluation measures from section 4. A further analysis and discussion of the results is provided in section 7.

6.1 First round of tuning

For the first round of tuning, we tuned the following parameters:

- **Epochs**: the number of epochs the model run.
- **Hidden dim.**: the number of hidden dimensions.
- **LR**: the learning rate.
- **Layers**: the number of layers.

The other parameters are the same as for the baseline model.

The performance of each model in this first round of hyper-parameter tuning is presented in table 3. The columns **Binary $\mathbf{F_1}$** and **Prop. $\mathbf{F_1}$** report the binary F_1-score and proportional F_1-score respectively. Considering the best proportional F_1-scores for each value of epochs, we observe that configuring hidden dimensions to at least 50 and using only one hidden layer is favorable. The parameters yielding the best proportional F_1-score (0.250) has the following values:

$$\begin{aligned}\text{Epochs} &= 30\\ \text{Hidden dim.} &= 75\\ \text{LR} &= 0.001\\ \text{Layers} &= 1\end{aligned}$$

These are the hyper-parameter values used for further tuning.

6.2 Second round of tuning

In the second round of hyper-parameter tuning we use the best model from section 6.1 to tune the following parameters:

- **Batch**: the batch size.
- **Train emb.**: 1 or 0, respectively indicating whether we train the embeddings or not.

Similarly as in section 6.1, we report both the binary and proportional F_1-score for each adjustment. The results are presented in table 4. We observe that the model yielding the best proportional F_1-score (0.2614) has the following adjusted parameter values:

$$\begin{aligned}\text{Batch} &= 10\\ \text{Train emb.} &= 0\end{aligned}$$

6.3 Experiments with tag schemes

As seen in table 5, the model using the IO tag scheme clearly outperforms the others with respect to proportional F_1, yielding an average score of 0.285 (standard deviation 0.00918) on the dev set and 0.287 (standard deviation 0.0169) on the test set. Considering the test set, this is an improvement of 0.038 compared to the baseline. For the BIO model, where the tag scheme is the same as in the baseline, the increase in proportional F_1 on the test set is 0.015.

For binary F_1, the baseline model and the BIOL model have the best scores on dev and test respectively. The IO model is not far behind with an average binary F_1 of 0.425 (standard deviation 0.00808) on dev and 0.403 (standard deviation 0.0157) on test. In general, we observe the tendency that the F_1-scores of a model decrease as the number of unique tags in its tagset increases. This is most prominent for proportional F_1.

An example of predicted labels with various tag schemes is provided in table 6. The translation is available in example 2.

(2) *« Det store spranget » er en spenstig nyvinning :*
« Det store spranget » is a resilient innovation :
'« Det store spranget » is a refreshing new series :'

Performance					
Epochs	Hidden dim.	LR	Layers	Binary F_1	Prop. F_1
15	25	0.001	1	0.407	0.235
			2	0.434	0.235
		0.01	1	0.408	0.233
			2	0.428	0.212
	50	0.001	1	0.425	0.244
			2	**0.453**	0.229
		0.01	1	0.419	**0.246**
			2	0.436	0.223
	75	0.001	1	0.433	0.243
			2	0.447	0.214
		0.01	1	0.422	0.245
			2	0.436	0.218
30	25	0.001	1	0.401	0.229
			2	0.427	0.238
		0.01	1	0.411	0.246
			2	0.418	0.224
	50	0.001	1	0.420	0.234
			2	**0.443**	0.224
		0.01	1	0.425	0.245
			2	0.429	0.224
	75	0.001	1	0.431	**0.250**
			2	0.440	0.240
		0.01	1	0.421	0.247
			2	0.441	0.240
50	25	0.001	1	0.408	0.239
			2	0.426	0.241
		0.01	1	0.392	0.227
			2	0.415	0.218
	50	0.001	1	0.419	0.248
			2	0.432	0.234
		0.01	1	0.411	0.239
			2	0.415	0.223
	75	0.001	1	0.416	0.245
			2	**0.442**	0.240
		0.01	1	0.426	**0.247**
			2	0.437	0.241

Table 3: Binary and proportional F_1-scores on development set for first round of hyper-parameter tuning.

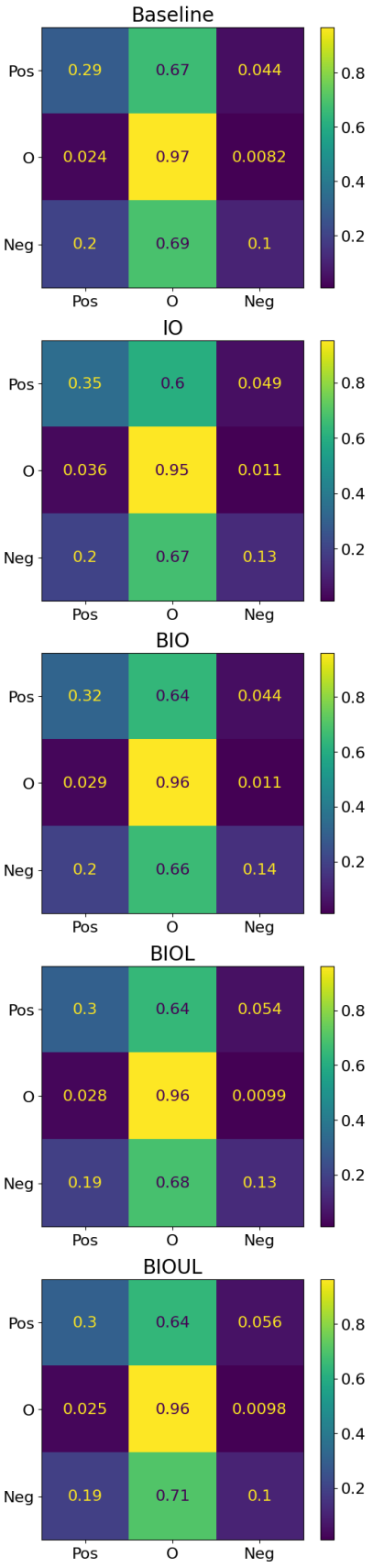


Figure 3: Proportional distribution of predictions with respect to polarity. For each gold polarity on the y-axis, the x-axis shows the frequency by which it is classified as Positive, Outside and Negative. Purple indicates a low frequency, while yellow corresponds to a high frequency.

Performance			
Batch	Train emb.	Binary F_1	Prop. F_1
10	0	0.4226	**0.2614**
	1	0.4138	0.2522
25	0	0.4242	0.2472
	1	0.4166	0.2506
50	0	0.4232	0.2450
	1	**0.4236**	0.2460

Table 4: Binary and proportional F_1-scores on development set for second round of hyper-parameter tuning.

Performance				
	Binary F_1		Prop. F_1	
Model	Dev	Test	Dev	Test
Baseline	**0.434**	0.395	0.255	0.249
IO	0.425	0.403	**0.285**	**0.287**
BIO	0.423	0.403	0.261	0.264
BIOL	0.414	**0.408**	0.243	0.245
BIOUL	0.414	0.404	0.243	0.241

Table 5: Binary and proportional F_1-scores on dev and test set for different tag schemes.

7 Discussion and Error Analysis

As found by Alshammari and Alanazi (2021), the model using the IO tag scheme obtains the best result. Contrary to Ratinov and Roth (2009), we observe that the BIOUL model yield worse results than the BIO model. We note, however, that both these papers experiment with these tag sets in the context of named entity recognition. In this paper, our targets are not necessarily *named* entities, and their labels contain an additional tag describing polarity.

In figure 3, we have kept the O label and merged all other labels into the two main categories 'Pos' and 'Neg', referring to their polarity. This illustrates the frequency by which each category is classified correctly, and how frequently it is misclassified as one of the other categories. We normalize the counts by the number of occurrences of each gold label. We observe that all five models in particular tend to struggle with labeling gold negative tokens correctly. The proportion of correctly classified tokens with negative polarity ranges from 0.1 to 0.14. For positive polarity, the corresponding numbers vary from 0.29 to 0.35. According to our

Token	Gold label (original)	BIO	IO	BIOL	BIOUL
«	B-targ-Positive	O	I-targ-Positive	O	O
Det	I-targ-Positive	O	O	O	O
store	I-targ-Positive	I-targ-Positive	I-targ-Positive	O	I-targ-Negative
spranget	I-targ-Positive	I-targ-Positive	I-targ-Positive	B-targ-positive	L-targ-Negative
»	I-targ-Positive	I-targ-Positive	O	L-targ-Positive	O
er	O	O	O	O	O
en	O	O	O	O	O
spenstig	O	O	O	O	O
nyvinning	B-targ-Positive	O	O	O	O
:	O	O	O	O	O

Table 6: Example sentence from NoReC$_{fine}$ test set, with gold labels from the original data set (BIO-format). Correct predictions are marked with green.

interpretation, this is a result of the class imbalance between positive and negative labels in the dataset, see figure 1. We believe that the models become better at identifying positive labels because there are more positive examples to learn from.

We note that gold positive labels are rarely predicted as negative, i.e. in the cases of misclassification, the predicted label is most frequently O. While this is also the case for gold negative labels, we observe that a large proportion of the wrong predictions belong to the positive class. Negative labels are more likely to be wrongly predicted as positive than be predicted correctly.

Furthermore, our results show that all models to a high degree label non-target labels (O) correctly. This suggests that the models are very good at classifying O. As previously noted, however, O is by far the most frequent label in the dataset (see figure 1), so these results are not surprising.

Figure 3 also highlights some differences between the five models. We observe that IO is superior to the others with respect to recall for positive labels. When it comes to negative labels, BIO shows the highest recall. The negative label recall for BIOUL is unfortunately not better than in the baseline.

8 Conclusion

In this work, we have shown that the choice of tag scheme in targeted sentiment analysis has an impact on the performance of the system. The best result was obtained using the IO tag scheme. However, as discussed in section 3.1 this tag scheme has certain weaknesses, and which tag set to choose depends on the task to be carried out. In addition, we have improved our baseline BiLSTM model by adjusting various hyper-parameters. Finally, we also provided a brief discussion and error analysis in section 7.

9 Future work

For future work, we suggest several modifications to improve our models:

Data Verification The adjustment of the original data set to other tag schemes like IO, BIOL and BIOUL is done automatically by our own script. We have looked through several samples in all converted data sets to verify that it works as expected. However, as these data sets contain many sentences, we were not able to consider each and everyone. To ensure the quality of the data, future work should seek to verify that all tags are indeed correctly annotated. In addition, one should count the number of consecutive entities in the dataset in order to see how much information is lost when applying the IO tag scheme.

Conditional Random Fields Modifying the BiLSTM-model to include Conditional Random Fields may improve the overall architecture resulting in improvements.

BERT The use of language models like BERT have in other NLP applications lead to improved performance and promising results. Thus, it should be considered used in this application as well.

Tag schemes Though we experimented with several tag schemes, we believe that even further exploring of other constellations of tags may lead to better results. For instance, the BIOU

tag scheme was not included in the experiments of this paper, but may be considered in future work.

More tuning Similarly to the tag schemes, the hyper-parameter tuning can also be expanded to include even more hyper-parameters. For instance, it may be relevant to tune the *dropout*-parameter. We did however not experiment with this, as it caused us inexplicable problems we were not able to solve.

Acknowledgements

We thank the WNLLP 2022 conference for the opportunity to carry out this task. In addition, we would like to thank the track chairs Egil Rønningstad and Erik Velldal for their guidance and advice on this study, as well as the rest of the IN5550 team at the University of Oslo for making this paper possible.

References

Nasser Alshammari and Saad Alanazi. 2021. The impact of using different annotation schemes on named entity recognition. *Egyptian Informatics Journal*, 22(3):295–302.

Yoav Goldberg. 2017. *Neural Network Methods for Natural Language Processing*, volume 37 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool, San Rafael, CA.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2020. A fine-grained sentiment dataset for Norwegian. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5025–5033, Marseille, France. European Language Resources Association.

Bo Pang, Lillian Lee, et al. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in information retrieval*, 2(1–2):1–135.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155, Boulder, Colorado. Association for Computational Linguistics.

IN5550 Exam: Targeted Sentiment Analysis

Qinghua Xu
Simula Research Laboratory, Kristian Augusts gate 23, Oslo, Norway
University of Oslo, Oslo, Norway
qinghua@simula.no

Abstract

Sentiment analysis is a traditional task in natural language processing, which aims at mining opinions or sentiments from these texts. With the success of neural network methods in various domains, researchers now has shifted their attention to a more fine-grained task called Targeted Sentiment Analysis (TSA), which classifies sentiments at target level. In this paper, we explore multiple ways of handling a TSA task by varying label encoding, learning procedures (Joint or Pipeline) and model structures (LSTM, GRU and transformer). To evaluate our method, we use both binary metrics and proportional metrics to compare our method with the baseline model provided by the course. Our best model improves the baseline by 0.047 in terms of binary F1 and 0.058 in terms of proportional F1. **The code is in https://github.uio.no/qinghuax/exam.**

1 Introduction

With the rapid growth of social media on the Web, text data such as reviews, forum discussions, blogs, micro-blogs, Twitter etc. We are now faced with an unprecedented huge volume of opinionated data recorded in digital forms. Opinions are central to almost all human activities and are key influencers of our behaviors.Identifying and classifying opinions from massive text data is challenging yet highly rewarding[8].

Sentiment analysis or opinion mining is the computational study of people's opinions, sentiments, emotions, appraisals, and attitudes towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes [7]. Target level sentiment analysis is a more fine-grained sentiment analysis task, which aims at identifying sentiment polarity towards a specific target in its context. For example,given a review "great food but the service was dreadful",the polarity towards food and service would be positive and negative respectively[9].

Various methods has been proposed to address target-level sentiment analysis task. The typical way is to train a machine learning model by supervised learning. Among these models, there are mainly two different types. One is to build the machine learning model on manually created features. The other type replace traditional machine learning models with neural networks to perform end-to-end feature extraction without any prior knowledge[9]. Neural models have shown great improvement in terms of accuracy and efficiency.

More recently,pretrained neural language models, such as Elmo[12], OpenAI GPT[13] and Bert[2] have shown their effectiveness in alleviating the effort of feature engineering. Therefore Sun et al. proposed a novel deep learning model,successfully utilizing Bert in target-level sentiment classification task and achieved the state-of-art results. We argue the key to the success of these models lies in the introduction of large amount of unsupervised data.

In this paper, we do not aim to surpass the state-of-art in TSA. Therefore, we only explore several options of performing this task and compare the experiments results with the baseline provided by the course. We explore in the following three directions:

- Label encoding. The original dataset provided by the course is encoded as combined BIO labels, containing information of target and sentiment at the same time. In this paper, we plan to explore other label encoding, such as combined BIOUL label encoding and split labels (both BIO and BIOUL).
- Learning procedure. In this paper, we consider two common types of learning procedure: joint learning and pipeline learning. In

our context, joint learning denotes a learning procedure that train a single model for the prediction of both targets and sentiments at the same time. On the other hand pipeline learning denotes learning procedures that learns how to perform target classification first and then learns to perform sentiment classification.

- Model structure. Text data is inherently in sequence form. In this report, we explore three different sequential neural network model structures, namely Gated Recurrent Networks (GRU) [1], Long Short Term Memory Networks (LSTM) [5] and transformers [16].

We will elaborate on methods and experiments of our work. In Section 2 we introduce related works in the targeted sentiment analysis field. We demonstrate details about the changes we made towards the baseline code in Section 3. In Section 4 and Section 5, we demonstrate the experiment design and analysis.

2 Related work

Target level sentiment analysis is a branch of sentiment analysis,which aims at identifying sentiment polarity towards certain target based on its context. Early works use rule based methods for target level sentiment classification,such as [3][10].Nasukawa et al. first perform dependency parsing on sentences, then they use predefined rules to determine the sentiment about targets [10]. Jiang et al. further improve this method by introducing target-dependent rules regarding sentences' grammar structure[6]

Later,various neural networks has been proposed to better represent context and target.TD-LSTM using two LSTM networks to model the left and right contexts of target words. The concatenation of last hidden states of these two LSTM are used for the final prediction. [15]. Recently, more researches has been focused on modeling the relationship between context and target using attention mechanism.Ma et al model the context and target words with two separate LSTMs [9]. They further use the hidden states generated from sentences to calculate attentions to target targets by a pooling operation, and vice versa. Hence their Interactive Attention Network (IAN) can attend to both the important parts in sentences and targets. Huang et al. use Attention over Attention module to better capture interactions among word-pairs between sentences and targets,overcoming the defects brought by pooling module in IAN [9].

Another line of research on target level sentiment analysis using neural network methods focuses on the employment of transfer learning techniques. Sun et al. successfully introduce Bert into target level sentiment classification task and achieved state-of-art results. They convert target level sentiment classification task into a sentence-pair classification problem which can be directly addressed by Bert models.[14] He et al. [4] pretrain their model on document level dataset,and then fine-tune it on target level sentiment classification tasks. Transfer learning techniques like this greatly enrich the original datasets which is too small for complicated neural network model to train.

3 Methodology

In this report, we build our model based on the baseline code provided by course IN5550. In this section, we will briefly introduce the structure of baseline code in Section 3.1. In Section 3.2, Section 3.3, and Section 3.4 we presents the changes we made on label encoding, learning procedure and model structure respectively.

3.1 Baseline

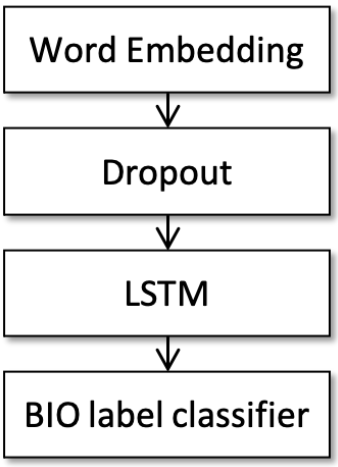


Figure 1: Model structure of baseline

The baseline provided in course IN9550 is a simple LSTM-based classification model. We do not use the baseline code directly. We re-code the baseline ourselves to make it fit in our code base, but we did not change any configurations of the model structure. Figure 1 shows the structure of baseline code. We first feed the input text into a word

embedding layer, transforming raw text into word embedding vectors. These vectors are then fed into a dropout layer to prevent over-fitting. An LSTM layer is used to capture the sequential information from the text. Finally, the last layer is a BIO label classifier, which transforms the output vectors into the label space likelihood vectors. BIO encoding labels words as three categories, namely B (begin of target), I(Inside the target) and O (others). Sentiments for each target is binary: positive or negative. So this classification is a 5 class classification, where labels are: B-targ-Positive, B-targ-Negative, I-targ-Positive, I-targ-Negative and O.

3.2 Different label encoding

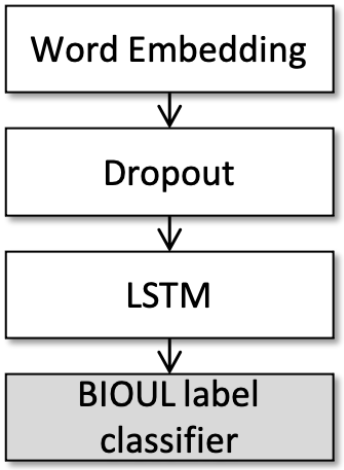


Figure 2: Model structure for using BIOUL label encoding

The first change we explore is label encoding. The baseline code uses BIO encoding. We explore another encoding called BIOUL. BIOUL encoding is an extension of BIO encoding. Besides beginning of target (B), inside the target (I) and others (O), BIOUL encoding introduce two more tags: unit target (U, targets whose length is 1) and last of target (L).

Figure 2 shows the new model with BIOUL label classifier. Instead of a 5-class classifier, it is now a 9-class classifier, whose labels are: B-targ-poistive, B-targ-negative, I-targ-positive, I-targ-negative, U-target-positive, U-target-negative, L-targ-positive, L-targ-negative and O.

3.3 Different learning procedure

Regarding learning procedure, we explore two different types: joint and pipeline learning. We are

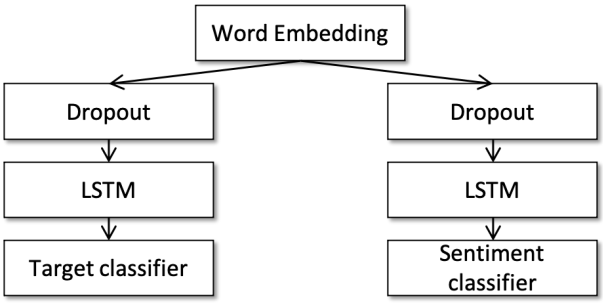


Figure 3: Model structuring for joint learning

aware that the baseline code is in joint learning fashion already, given that it uses a combined label that integrates both sentiments and targets. Predictions on this label allows us to improve the capability of sentiment prediction and target prediction at the same time. In this paper, however, we explore another method of joint learning by separating the sentiment and target labels and build separate models for each of them. Figure 3 illustrate the model structure of joint learning procedure. The word embedding layer is shared by both target and sentiment prediction model. Regarding other layers, these two models has their own set of parameters, including dropout layer, LSTM layer and final classifier layer. We train these two models at the same time in a multi-task learning fashion.

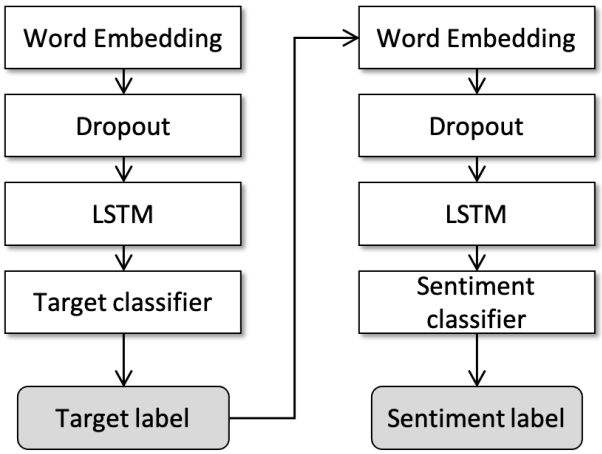


Figure 4: Model structure for pipeline learning

As for pipeline learning, we build also two separate models for targets and sentiments prediction as in Figure 4. But different from joint learning, we use the output of target prediction as an input of sentiment prediction in pipeline learning.

For both joint and pipeline learning, we employ the same BIO labeling encoding. Target classifier is 3-class classifier, with labels of : B-targ, I-targ

and O. Sentiment classifier is a binary classifier, with labels of : positive and negative.

3.4 Different model structure

Another exploration in this paper is to vary the sequential model structures. In the baseline code, LSTM is used to capture the sequence characteristics. We experiment with two other sequence models: GRU [1] and transformer [16].

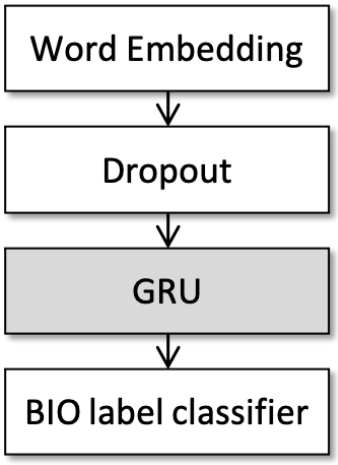


Figure 5: Model structure for using GRU

Figure 5 depicts our model with GRU as sequence model. GRU is an extension of LSTM model. Both LSTM and GRU are based on gating mechanism to combat the problem of vanishing gradient. However, these two models are different in several aspects:

- LSTM has three gates, namely input gate, forget gate and output gate, while GRU only has update and reset gate.
- GRU does not possess any internal memory, unlike LSTM which calculates a candidate vector for this purpose.
- The reset gate of GRU is applied on previous hidden state directly instead of calculating a candidate value.

In Figure 6, we can see that a transformer encoder (Multi-head Attention Module) consists of five layers: positional encoding layer, attention layer, residual layer, feed-forward layer and another residual layer. Let U be the text embedding vectors. Multi-head Attention Module takes U' as the input and uses positional encoding to incorporate position information as shown in Equation 1.

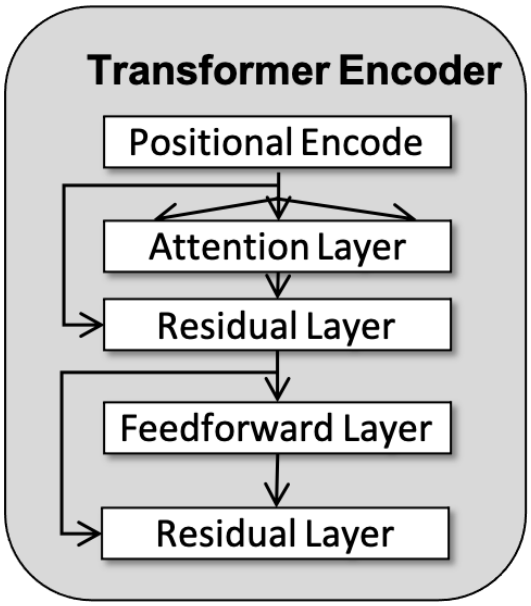


Figure 6: Architecture of Attention Mechanism

$$U' = U' + position\_encoding(U') \quad (1)$$

We duplicate this vector into three and perform linear transform on these three l: U_k, U_v and U_q. U_k and U_v are for calculating the attention weight for each vector. With this weight, we then calculated a weighted sum of U_q. Equation 2 shows the calculation of attention.

$$attention\_weight = U_q U_k^T \quad (2)$$

$$U_{attention} = attention\_weight U_v^T \quad (3)$$

Then a residual connection between the input vectors and the output of multi-head attention is built, as shown in Equation 4.

$$U_{attention} = normalize(U' + U_{attention}) \quad (4)$$

Feed-forward Module takes the output of attention module as its input and feeds it into a feed forward network (Equation 5).

$$U = W_{attention} U_{attention} + b_{attention} \quad (5)$$

$$U_{feed} = relu(U) \quad (6)$$

Then a residual connection between the attention output and the feed-forward output is built (Equation 7.

$$U_{out} = normalize(U_{attention} + U_{feed}) \quad (7)$$

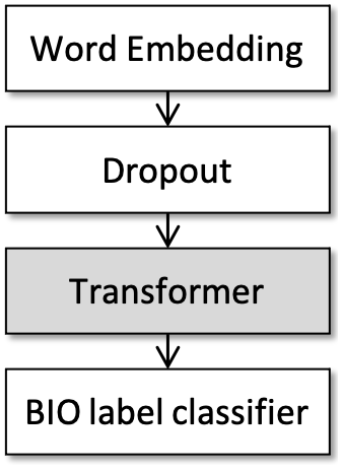


Figure 7: Model structure for using transformer

Figure 7 shows our model with transformer as the sequence model. With transformers, we can take advantage of attention mechanism for capturing context information. The original transformer model consists of two component: encoder and decoder. In this paper, we only utilize the encoder to incorporate context information into the token vectors.

4 Experiment Design

In this section, we presents our design of experiments. In Section 4.1, we propose three research questions to evaluate the changes we make to the baseline code. Section 4.3 demonstrate the evaluation metrics we use in our experiments. We show the parameters and execution environment.

4.1 Research Questions

In this paper, we propose three research questions as follows:

- **RQ1:** Is using BIOUL label encoding effective in TSA compared to BIO label encoding?
- **RQ2:** Is joint learning effective in TSA compared to pipeline learning?
- **RQ3:** Is GRU or transformer effective in TSA compared to LSTM?

With RQ1, we plan to evaluate the effectiveness of using BIOUL label encoding. We compare the experiment results with BIO label, while keep the model structure unchanged. With RQ2, we evaluate to effectiveness of joint learning. We split the combined BIO label (combining both targets and sentiments) into targets labels and sentiments labels. We then perform joint learning and pipeline learning and compare the results. With RQ3, we aim to evaluate the effectiveness of different neural sequence models. We compare the results of baseline with GRU-based model and transformer-based model.

4.2 Dataset

	Train	Dev.	Test	Total	Avg.len
Sents	8634	1531	1272	11437	16.8
Targets	5044	877	735	6656	2.0

Table 3: Number of samples in the dataset

In this paper, we use the dataset named $NoReC_{fine}$, which is provided by the course. This is a fine-grained Norwegian sentiment analysis dataset. To facilitate researching, this dataset has been annotated with polar expressions and targets. The text are from a corpus of professionally authored reviews in various domains, such as literature, games,music, products, and movies. We present the number of samples in Table 3.

4.3 Evaluation Metrics

We follow the instructions given by IN5550 and evaluate our method with two types of metrics: binary metrics and proportional metrics. Binary metric counts any overlap of predicted and gold span as correct, while proportional precision denotes the ratio of the overlap with predicted span and proportional recall denotes the ratio of the overlap with gold span, which reduce to token-level F1. Proportional metric is more strict than binary metric.

4.4 Parameters and execution environment

This project is coded with Pytorch library [11]. We run all of our experiments on one node of SAGA cluster. This node use HPE 16GB (12 x 16GB per node) Dual Rank x8 DDR4-2666 CAS-19-19-19.

Hyperparameter tuning is not the focus of this paper. Therefore for most experiments, we follow the hyperparameter settings of the baseline code, only varying the parameters that we want to compare. Also, we use a much smaller learning rate (2e-5) with transformer model, in case the learning rate is too large for it to learn.

5 Experiment Analysis

In this section, we presents the experiment results of RQ1-RQ3.

Setup	Label	Procedure	Model	Lr	N	H	Others
baseline	BIO	joint	LSTM	0.01	50	100	bidirectional;1 layer
Label	BIOUL	joint	LSTM	0.01	50	100	bidirectional;1 layer
Procedure	BIO	joint	LSTM	0.01	50	100	split labels
	BIO	pipeline	LSTM	0.01	50	100	split labels
Model	BIO	joint	GRU	0.01	50	100	-
	BIO	joint	transformer	2e-3	50	100	dmodel=100; head=1

Table 1: Parameters in different experiment setups.Lr, N and H denote learning rate, batch size and hidden dimension respectively.

	Binary			Proportional		
Setup	Precision	Recall	F1	Precision	Recall	F1
Baseline	**0.436**	0.251	0.319	0.315	0.139	0.193
BIOUL label	0.415	0.254	0.315	0.301	0.194	0.236
Joint learning	0.432	0.336	**0.378**	0.313	0.210	0.250
Pipeline Learning	0.399	**0.349**	0.372	0.267	**0.235**	0.250
GRU model	**0.436**	0.315	0.366	**0.326**	0.218	**0.261**
Transformer model	0.326	0.321	0.324	0.159	0.104	0.126

Table 2: Experiment results

5.1 Result and Analysis for RQ1

RQ1 evaluates the effectiveness of label encoding. We experiment with BIOUL label. The second row of Table 2 shows the result of using BIOUL label. We can observe that BIOUL label does not improve the binary metric very much (only binary recall by 0.003). However, proportional recall and f1 are improved by 0.055 and 0.043 respectively. This results show that BIOUL label performs better with proportional metrics, which means it is good at identifying the boundary of target spans.

5.2 Result and Analysis for RQ2

RQ2 evaluates the effectiveness of learning procedure. We experiment with both joint and pipeline learning procedure. We can observe from Table 2 joint learning (modified from the baseline code) improves the baseline with both binary f1 (0.059) and proportional f1 (0.057). Pipeline learning also improves the baseline, but the binary f1 is lower than joint learning (0.006). This result shows that pipeline performs slightly worse than joint learning. Pipeline learning predicts targets first and use predicted targets to further predict sentiment. We argue that this can induce cascading errors if the target prediction is not accurate enough, which can lead to worse performance.

5.3 Result and Analysis for RQ3

RQ3 evaluates the effectiveness of model structures. In Table 2, we can observe that GRU model has the best performance on binary precision, proportional precision and proportional f1. This shows that GRU has a better performance than LSTM in this code. The reason for this improve is because GRU has few gates than LSTM and fewer parameters. In this TSA task, we do not have a large amount of data. Therefore, our model benefits from a simpler model, i.e. GRU model. However, transformer model surprisingly has bad performance. The result for proportional f1 is even lower than the baseline code. We think this poor performance is because transformer is more complicated in terms of parameter number, which can have better performance with large amount of data. Also, transformer might need more hyperparameter tuning for it work well, which is not in the scope of this paper.

6 Conclusion and future work

In this paper, we make various changes to the baseline code to handle the TSA task. We varied the label encoding, learning procedure and model structure in our experiments. Experiment results show that BIOUL label encoding, joint learning, GRU and Transformer are effective in TSA task. We plan to investigate more in this field, including using contextualized embeddings such as Elmo and

Bert, performing hyperparameter tuning to search better hyperparameter settings, etc.

References

[1] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Xiaowen Ding and Bing Liu. The utility of linguistic rules in opinion mining. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 811–812. ACM, 2007.

[4] Ruidan He, Wee Sun Lee, Hwee Tou Ng, and Daniel Dahlmeier. Exploiting document knowledge for aspect-level sentiment classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 579–585. Association for Computational Linguistics.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 151–160. Association for Computational Linguistics, 2011.

[7] Bing Liu. *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press, 2015.

[8] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM, 2017.

[9] Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Interactive attention networks for aspect-level sentiment classification. *arXiv preprint arXiv:1709.00893*, 2017.

[10] Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[12] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[13] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[14] Chi Sun, Luyao Huang, and Xipeng Qiu. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 380–385. Association for Computational Linguistics.

[15] Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Effective lstms for target-dependent sentiment classification. *arXiv preprint arXiv:1512.01100*, 2015.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Comparing cross-lingual and mono-lingual models for Norwegian targeted sentiment analysis

Tellef Seierstad
tellefse@ifi.uio.no

Abstract

This paper analyses and compares the results on targeted sentiment analysis for Norwegian from different model architectures. These results show that there is a significant difference between the types of errors GRUs and transformer models get for this task. This research also found that a multi-lingual model performed better than the mono-lingual NorBERT for targeted sentiment analysis on the data set $\text{NoReC}_{\text{fine}}$.

1 Introduction

The task this paper deals with is targeted sentiment analysis (TSA), which aims not only to find the sentiment in a given text, but also the target of that sentiment. Sentiment analysis is a field within natural language processing (NLP) that attempts to determine the sentiment of a text, where sentiment is a positive or negative evaluation expressed through language (Taboada, 2016). Where standard sentiment analysis outputs one sentiment for the whole input, TSA is a form of token labeling, like named entity recognition (NER), where each token is assigned a label showing whether it is part of an entity or not.

In this paper, we experiment on several model architectures and hyperparameters to find out which performs best for the specific task of TSA for Norwegian. Furthermore we attempt to compare the results between different models to see what the difference in performance consists of, and what strengths and weaknesses different models have for this task, which means that the most extensive part of this paper will be the error analysis.

2 Data set

The data set used is $\text{NoReC}_{\text{fine}}$, which is a data set for fine-grained sentiment analysis in Norwegian (Øvrelid et al., 2020). $\text{NoReC}_{\text{fine}}$is annotated with respect to polar expressions, targets and holders of opinion. For this paper, a specifically curated version of $\text{NoReC}_{\text{fine}}$with only the targets and their sentiment is used. Fine-grained in this case means that the data is annotated on the token level using the BIO format, not that the sentiment can take several values, as seen in Munikar et al. (2019). This means that the data set has in total five different labels: O, B-pos, I-pos, B-neg and I-neg.

3 Experimentation

3.1 Baseline

For the baseline model a bidirectional gated recurrent unit (GRU) with static embeddings was used. The embeddings used were case-preserved and not lemmatized Gensim CBoW embeddings for Norwegian[1]. During training of this baseline model, the embeddings were trained along with the GRU, albeit with a significantly lower learning rate, a factor of 50 to 200 lower than the GRU.

Then the baseline model was extended with a conditional random field (CRF), which has been shown to produce good results on bidirectional LSTMs (Huang et al., 2015) and transformers (Chernyavskiy et al., 2021). Extending the baseline model with a CRF did not give better results in this case, but further tuning of that model might have been necessary to increase the scores.

3.2 Transformer models

When it comes to models used for the experimentation, the main focus has been transformer models using contextual embeddings, since that has been the state of the art since Devlin et al. (2019) introduced BERT (Chernyavskiy et al., 2021). The architectures that were tried during the experimentation were BERT and RoBERTa (Robustly Optimized BERT Approach). The specific models used were NorBERT, NorBERT2, XLM-RoBERTa, XLM-RoBERTA-large and an XLM-RoBERTA-

[1] http://vectors.nlpl.eu/repository/20/92.zip

large model fine-tuned for named entity recognition (NER) on the the English part of the CoNLL - 2003 data set (Sang and De Meulder, 2003).

NorBERT was first presented in 2021 by Kutuzov et al. and is trained from scratch for Norwegian. NorBERT 2 was released a year later and is trained on a much larger corpus. XLM-RoBERTa (XLM-R) is a cross-lingual model introduced by Conneau et al. (2020) that has shown good results, even outperforming monolingual BERT models in some cases. The experimentation on these models were partly intended to see whether the cross-lingual model could outperform NorBERT for the specific task of targeted sentiment analysis. Since targeted sentiment analysis has some similarities with NER, an XLM-R model already fine-tuned for NER was also tested. Using models already trained for NER in TSA has also been suggested by Rønningstad (2020).

3.3 Labeling scheme

$\text{NoReC}_{\text{fine}}$is annotated using BIO-labels, and this has been the labeling schema used for most of the experimentation. However, both IO-labels and BILOU-labels have also been tried. IO-labeling means that the only thing that is annotated is whether a token is part of an entity or not, and gives no information about the position of the token within the entity. Thus, IO-labeling loses some information present in the original annotations, such as a B - B - I - sequence, which indicates two separate entities, would become I - I - I, with no way to infer that there is actually two entities within the sequence. Nonetheless, it was found that the validation set contains very few entities where IO-labeling would lead to such ambiguity, so that was tried as well.

The BILOU-scheme has in addition to the BIO-labels a "Last label" (L) for tokens that end an entity, and a "Unit label" (U) for entities consisting of a single token. This BILOU-scheme has not been conclusively shown to give better results than the more widely used BIO-scheme (Chernyavskiy et al., 2021; Cho et al., 2013), but it can be useful to get more fine-grained information for error analysis, e.g. by making it possible to analyse whether single-token entities are easier to predict than longer entities. The BILOU-scheme was tested during the experimentation, but the performance fell significantly and no further analysis was done on the BILOU-scheme.

4 Results

The evaluation metrics used in this paper will be binary f1-score and proportional f1-score. The binary score counts any overlapping predicted and gold spans as correct, whereas the proportional score uses the ratio of overlap between the gold and predicted spans. That means the proportional score is the stricter of the two. Proportional f1 score was the metric used for hyperparameter optimizing.

	search space		XLM		
param	min	max	large	base	BERT
lr	1e-6	1e-4	1.4e-5	6.4e-5	3.8e-5
wd	1e-12	1e-1	7.1e-6	3.6e-7	1.0e-8
eps	1e-10	1e-6	3.9e-9	1.6e-7	1.3e-9
warm	0.0	0.5	0.137	0.266	0.025
seed	1	42	6	42	24
h drop	0.0	0.25	0.221	0.185	0.097
c drop	0.1	0.3	0.226	0.253	0.248
sched			pol	cos	const
best			0.381	0.365	0.317

Table 1: Hyperparameter search space and their results. This was a randomized hyperparameter search running 100 different configurations using the Optuna Framework[2]. The hyperparameters that were optimized, in order of appearance in the table, were learning rate, weight decay, epsilon for the Adam optimizer, warm-up ratio, the seed (only for reproducibility), the dropout in the transformer models' hidden layers and the dropout for the input of the transformer model to the final classifier layer. In addition, the scheduler for the optimizer was one of linear, polynomial, cosine and constant with warm-up. The models used in the search were standard versions of XLM-R large, XLM-R base and NorBERT2.

As can be seen in Table 1, the optimized large XLM-R model gets a higher proportional f1 score with a notably lower learning rate, especially compared to the XLM-R base model. The NorBERT model did not achieve the same scores as the cross-lingual model, and this is also the case in the results below.

It is interesting to see in Table 2 that the monolingual Norwegian models performed worse than the cross-lingual models with regards to the proportional F1 score, while being on par with them when looking at the binary scores.

The best model trained and tested on the IO scheme originally got an f1 score of 0.452. However, it would be somewhat misleading to use that

[2]https://optuna.readthedocs.io/en/stable/

Model	Labels	Bin F1	Prop F1
GRU	BIO	0.551	0.384
GRU-CRF	BIO	0.497	0.370
XLM-R	BIO	0.543	0.394
XLM-R-large	BIO	0.501	0.399
XLM-R	BILOU	0.463	0.312
XLM-R-large	IO	0.516	0.382
XLM-R-conll3	BIO	**0.575**	**0.430**
NorBERT 2	BIO	0.529	0.215
NorBERT 1	BIO	0.502	0.232
NB-BERT large	BIO	0.564	0.271
M-BERT	BIO	0.486	0.247

Table 2: Results (binary and proportional f1 score) on the test set for different model configurations and label schemes

score in the table since after converting the predictions back from the IO to the BIO scheme and testing on the original data, the proportional f1 score fell by 7 % to 0.382. The BILOU scheme did not lead to good results either, but it may be that using that labeling scheme would require changing the hyperparameters, noting that no hyperparameter search were made specifically for the BILOU scheme.

The model already fine-tuned for NER on the Conll-3 data set performed best. That could mean that even though the classification head is changed for the TSA task, the pretrained RoBERTa model itself retains some knowledge of NER that is useful for TSA as well.

Even though the transformer models slightly improve the score, it is also noticeable how well the baseline GRU and GRU with CRF trained on static embeddings perform compared to the much larger transformer models. These simpler models can be trained on a laptop and do not require access to a GPU cluster to be able to do experiments.

5 Error analysis

Much of this paper will deal with error analysis, going through what the model classified correctly and what it classified errantly. The best models were trained on the large version of the cross-lingual language model XLM-R (Conneau et al., 2020). The best of these two was also fine-tuned for named entity recognition (NER) on the the English part of the CoNLL - 2003 data set (Sang and De Meulder, 2003). The analysis below will mostly focus on XLM-R-large fine-tuned for NER, and unless specified otherwise, the tables will describe results from that model.

5.1 Quantitative error analysis

The error analysis will start at a quantitative level, and then delve into more qualitative analysis later. Overall, there were 876 entities in the evaluation set, and 663 entities were predicted for XLM-R, 609 for the GRU and 791 for NorBERT 2.

Sentiment	XLM-R	GRU	BERT	gold
Negative	186	89	175	256
Positive	477	520	616	620
POS ratio	0.72	0.85	0.78	0.71

Table 3: Number of predicted and gold entity types as well as the ratio of positive entities for models XLM-R-large-conll3, the baseline GRU and NorBERT 2.

The distribution of negative and positive entities was nearly equal for the predicted and true entities in XLM-R, but not for the other models as shown in Table 3. The GRU seems to be a lot more affected by the label imbalance in the data set, and leans severely towards positive labels, with only 15% of its predictions as negative. NorBERT is also skewed towards positive sentiment, but not as heavily as the GRU.

However, when it comes to the actual scores, they are higher for positive entities predicted by the XLM-R model as well, as can be seen in Table 4 below.

Entity	precision	recall	f1-score
Negative	0.31	0.23	0.26
Positive	0.40	0.30	0.34
micro avg	0.37	0.28	0.32
macro avg	0.35	0.27	0.30
weighted avg	0.37	0.28	0.32

Table 4: Metrics for the two entity types and aggregated metrics computed using the seqeval library[3] for the XLM-R-large-conll3 predictions.

In the entity-level classification report for the predictions on the validation set in Table 4, one can see that the positive targets have both higher precision and recall than the negative targets. This can be attributed to the discrepancy in target distribution in the training set, given that the model was

[3]https://github.com/chakki-works/seqeval

trained on 1558 negative targets and 3486 positive targets. Even though that distribution is the same in the evaluation data set, it still means that the model has had less examples to learn to distinguish the negative targets in the text.

The picture is similar when you look at it at the token level as in Table 5. There is a clear difference between the positive and negative tokens, but also between tokens beginning entities and tokens that are only part of the entity. The I-labels have lower precision but the same or higher recall than the B-labels. This could be because an I-label has to come after a B-label, which means that the classifier has some information that can make it more likely to predict an I-label. It is worth noting here that the scores are a bit higher than for the entity-level scores because parts of an entity could be classified correctly, but not all of it, which would increase the token-level score here but not the entity-level score. Micro average f1 score here is the same as the proportional f1 score.

token label	precision	recall	f1-score
B-Pos	0.59	0.37	0.45
I-Pos	0.53	0.47	0.49
B-Neg	0.45	0.26	0.33
I-Neg	0.37	0.26	0.31
micro avg	0.52	0.37	0.43

Table 5: Token-level metrics with micro average for XLM-R-large-conll3 predictions.

When considering just the extracted entity spans, and not their polarity, the performance increases slightly. The f1-score goes from 0.32 to 0.37 on the entity-level metrics as shown in Table 6.

entity	prec	recall	f1-score
XLM-R spans	0.431	0.321	0.368
XLM-R targets	0.373	0.282	0.321
GRU targets	0.384	0.267	0.315
NorBERT 2 targets	0.307	0.277	0.291

Table 6: Entity-level metrics for different models as well as only entity spans without sentiment for XLM-R-large-conll3.

On the token-level, shown in Table 7, scores are increased even further, but still only 6% past the scores that include sentiment as well.

The above analysis suggests that the hardest part for the classifier is to extract the correct spans. The sentiment analysis itself is arguably easier, because once an entity is extracted, the binary sentiment analysis only needs to find the most probable sentiment. This might indicate that a pipeline approach, where focus can be put on the entity extraction first, and then the sentiment extraction might be a good approach, as was indeed found to be the case by (Pereira et al., 2021). However, Li et al. (2019) suggest that for sub-tasks with a strong coupling, like extracting entities and their sentiment, a more integrated model is usually more efficient than a pipeline model.

token label	precision	recall	f1-score
B-target	0.63	0.38	0.48
I-target	0.55	0.46	0.50
micro avg	0.59	0.42	0.49

Table 7: Token-level metrics with only target spans, not sentiment, for XLM-R-large-conll3.

When predicting the entity spans and the sentiment jointly, the label space increases from 3 to 5 compared to just extracting the entity, using BIO labels. In general, no matter the label scheme used, the joint model for TSA increases the label space by a factor of 2, minus 1 because the outside label stays the same.

The statistics also show that the classifier predicts shorter entities than what's actually present in the training set and the validation set. The following Table 8 shows some statistics for the number of tokens in each entity for the training and validation set, as well as for the predicted true positives. The most interesting part here is that the GRU is a lot better at predicting long entities, and the GRU's entity length distribution is similar to the training and validation data, except for some very long outliers.

5.2 Error types

From here the focus will be on what kind of errors are occurring in the predictions. For this job, we have used the Python library NER Error Analyzer[4], which gives decent overall statistics as well as good visual output for sentence and entity-level error analysis.

In the general error type summary in Table 9 it is interesting to see that the GRU is in the middle and performs better than NorBERT but worse than XLM-R when it comes to false negatives/positives.

[4]https://github.com/ciaochiaociao/ner_error_analysis

Stat	Train	Val	XLM	GRU	BERT
count	5044	870	247	234	243
mean	1.97	1.96	1.45	1.92	1.39
std	1.89	1.80	1.05	1.49	0.94
min	1	1	1	1	1
25%	1	1	1	1	1
50%	1	1	1	1	1
75%	2	2	1	2	1
max	35	15	11	8	7

Table 8: Statistics for entity lengths among the train data, validation data and predicted true positives for XLM-R-large-conll3, the baseline GRU and NorBERT 2

Error type	XLM-R	GRU	NorBERT
false_error	662	715	793
type_error	50	80	57
span_error	129	61	124

Table 9: An overview of different error types for three chosen model predictions. False_error means false negatives/positives, type_error means that the sentiment was changed and span error means that there was some but not total overlap between predicted and gold span.

It is also clear that the transformer models are better at getting the correct sentiment for a given entity span. However, the GRU is a lot better at getting the correct span, with less than half the number of span errors as the other models. This could mean that the GRU retains more positional information than the transformer models, whereas the contextual embeddings are better than the static embeddings when it comes to the sentiment analysis itself. A more detailed summary of the different types of errors is shown in Table 10, sorted by most common to least common.

Some of the error types are self-explaining, like the false negatives/positives, but others might require some explanation. Diminished means that some tokens with a label other than 'O' in the gold data, were classified as 'O', either on the right or left side of the entity, or both. Crossing means that the predicted entity is shifted to one direction compared to the true entity. An illegal tag sequence is an I-label that occurs after a label of another entity type or after an outside label, including I-labels at the start of a sentence. Examples of these error types will come in the qualitative analysis section below. In Table 10 we can also see that diminishing

Error type	XLM	GRU	BERT
False Negative-POS	302	332	299
False Positive-POS	150	182	265
False Negative-NEG	142	168	147
Illegal tag sequence	130	103	138
False Positive-NEG	68	33	82
NEG -> POS	21	55	36
Right Diminished	38	20	52
POS -> NEG	22	20	16
Left Diminished	16	26	12
Right Expansion	15	1	21
Right Left Diminished	14	7	14
Left Expansion	9	11	10
Right Crossed	14	0	9
Left Crossed	15	1	6
Right Left Expansion	5	0	0
POS -> NEG|POS	2	1	0

Table 10: Number of occurrences of different error types for each model: XLM-R-large-conll3, the baseline GRU and NorBERT 2

occurs more often than expansion.

As noted above, the GRU has fewer span errors, especially fewer right reductions and right expansions, but more left reductions and expansions. Also worth noting is how many gold negatives the GRU turns to positives compared to the other models, as the GRU was more affected than the others by the label imbalance. The GRU also has fewer illegal tag sequences than the transformer models.

5.3 Qualitative error analysis

5.3.1 Examples

In the examples below, red means wrongly predicted entity and yellow means true entity that was not predicted correctly. Green means a correctly predicted entity. These examples come from the predictions of the XLM-R-large-conll3 model.

1. Det starter mørkt og intenst med [[«]Pos [Gosh »]Pos før]Pos det går videre til den dypeste , delikate [house]Pos i [« [Sleep Sound »]Pos til en skitten]Pos , men fengende [drum'n'bass]Pos i [« SeeSaw »]Pos.

2. Det [repetitive]Pos er [albumets]Pos store styrke , på [« Evil »]Pos , [« Surrender »]Pos og nevnte [« I need something new »]Pos.

3. [[Resten]Pos av gjengen]Pos , [Viggo Krüger på gitar]Pos , [Roar Ruus Finsås på bass]Pos

, [Tor Harald Rødseth bak tangentene]Pos og [trommis Martin Smith-Sivertsen]Pos leverer solid og stødig uten å overdrevent briljere.

4. [Diskanten]Pos var finkornet , og selv om [de]Neg ikke var like nøytrale som [Momentum-modellen]Pos opplevde vi [bassen]Pos som detaljert nok - til tross for at [de]Pos ga oss en høyere grad av morofaktor.

5. Men selv om [[låtmaterialet]Pos]Neg kan fremstå noe forvirrende , er det ingen tvil om at Øyakonsertens eneste gåsehud-øyeblikk , [« [Alias]Pos »]Pos , er en fabelaktig låt , spesielt når [Fridéns]Pos nesten growlende vokal over den hakkete riffingen og det nydelige [midtpartiet]Pos glir sømløst inn i hverandre.

6. Det morsomste er at [den]Pos [ikke]Pos har et eneste dødpunkt.

5.3.2 Translations:

1. It starts dark and intense with [[«]Pos [Gosh »]Pos before]Pos it continues to the deepest, delicate [house]Pos in [« [Sleep Sound »]Pos to a dirty]Pos , but catchy [drum'n'bass]Pos in [« SeeSaw »]Pos.

2. The [repetitive]Pos is [the album's]Pos great strength, on [« Evil »]Pos , [« Surrender »]Pos and mentioned [« I need something new »]Pos.

3. [[The rest]Pos of the group]Pos , [Viggo Krüger on guitar]Pos , [Roar Ruus Finsås on bass]Pos , [Tor Harald Rødseth behind the keys]Pos and [drummer Martin Smith-Sivertsen]Pos deliver a solid performance without excelling.

4. [The treble]Pos was fine-grained , and even though [they]Neg were not as neutral as the [Momentum model]Pos we experienced [the bass]Pos as detailed enough, despite [them]Pos giving us a higher degree of fun.

5. But even though [[the song material]Pos]Neg can seem somewhat confusing , there is no doubt that The Øya Concert's only goosebump moment , [« [Alias]Pos »]Pos , is a fabulous song, especially when [Fridén's]Pos almost growling vocals over the choppy riffing and the beautiful [middle part]Pos glide seamlessly into each other.

6. The most fun part is that [it]Pos does [not]Pos have a single dead center.

5.3.3 Sentence analysis

The first error in [1] can be a bit hard to read, but the yellow brackets cover "[« Gosh »]", which means that "« Gosh »" is the correct entity. There are two sets of red brackets, which means that "«" was predicted as a separate entity, and then the rest as another entity, simply because both of the tokens "«" and "Gosh" were assigned B-labels by the model. The rest of the errors are mostly false negatives, except for "« Sleep Sound »" which was predicted as "Sleep sound » til en skitten" ("Sleep Sound » to a dirty"), an example of a right cross error. In this case the predicted entity is not a phrase by itself, one of several such errors which will be described further below.

In general, the model was not very good at understanding that if one quotation mark is part of the entity, then the other quotation mark is as well. Chernyavskiy et al. (2021) describe this lack of consistency for sequential tag predictions as a common problem for all transformer-based models, and thus they are not able to learn the general rule for punctuation and quotes.

In [2] there is one false positive and some false negatives. We can see that the model predicts the noun "albumets" ("the album's") as an entity, and not the adjective "repetitive", which was the gold entity, but it got the sentiment right. As we can see in both [2] and [3], the enumeration of different names, each split into an entity, is hard to predict and are all false negatives in these examples.

The first error in [3] is an example of a recurring error; An entity which is a noun phrase containing several words is only partly predicted by the model, like here where only "Resten" ("The rest") is predicted from the true entity "Resten av gjengen" ("The rest of the group"). This error also goes the other way in some cases, like when "Pixars store film om følelser" ("Pixar's big movie about feelings") is predicted from the true entity "Pixars store film" ("Pixar's big movie"), but the predicted entity is more often diminished than expanded.

A further issue with a significant amount of the predicted entities, like when quotation marks are missing, is that they are not correct noun phrases at all. Examples of such predictions are "album med" ("Album with") instead of "album" and "de tre" ("the three") instead of "tre historiene" ("three histories). The last one should arguably be "De tre

historiene" ("The three histories") to be a complete noun phrase, but that is besides the point. An interesting approach to this problem could be to use a model that's trained on syntactic parsing, specifically constituency parsing, and fine-tune that on this data set. Another possibility could be to use the output from a constituency parser as input to the TSA model, and train all of it end-to-end. That could help to predict only entities that are complete, correct phrases.

In [4] we see one correct prediction "Diskanten" ("The treble") among a lot of false negatives. In general, the model predicted fewer entities in total than the amount present in the validation set. The model also has a tendency to predict nouns that start a sentence as an entity at a comparatively higher rate than other nouns. In addition to that, it seems like most predictions occur at the start of the sentence, whereas the gold entities are spread throughout the whole sentence. In [5] we also see an example of a NEG -> POS error, the only type error among these examples. It is hard to say how much weight the model gives to the positive words occurring later in the sentence compared to what describes "låtmaterialet" ("the song material") directly, i.e. "kan fremstå noe forvirrende" ("can seem somewhat confusing"). However, this description does not carry very strong sentiment, and might have been drowned out by the strong positive adjectives later on in the sentence, even though those in reality describe something else.

In [6] there is an off-by-one error, which is somewhat strange since "ikke" ("not") is never an entity by itself and is only part of two entities in the training set. This could mean that "ikke" ("not"), being a negator but not an entity by itself, is still seen as important by the model and given a lot of attention since it may flip the sentiment predictions.

5.3.4 Analysis of predicted entities

When it comes to entities the models did not manage to predict, "jobben min" ("my job") and "Dette" ("This") come out on top. The five times "jobben min" was not predicted was in practice just three times, since the last of the following sentences was repeated three times.

1. Som regel trenger jeg ikke denne koppen for å minne meg på at jeg elsker [jobben min]Pos.
2. Noen ganger hater jeg [jobben min]Neg.
3. Jeg elsker [[jobben]Pos min]Pos.

Given what we saw earlier about first words in a sentence being easier to predict for the classifier, it could be that the classifier finds it easier to classify phrases that are the subject of the sentence, possibly with a describing adjective as well. Detecting an entity based on being the object of a verb may be harder. In [3] we also see again that the model predicts shorter entities than the gold entities, often just including the noun and not the entire phrase.

Looking statistically at the entities' textual representation, there are in total 646 different entities in the validation set. 464 of them i.e. 72 % have no true predictions. 37 of the entities with no predictions occur more than once in the validation set. On the precision side of the entities, 182 of the 505 entities predicted were true positives, i.e. 36 %, while 323 (64%) of the predicted entities were not correct gold entities.

What we can see in Table 11 is that especially the transformer models predict words starting with an uppercase letter more often than lowercase words. It seems like those words are easiest to get right, like e.g. "Grafikken" ("The graphics") and "Sistnevne" ("The latter"), which have perfect recall and precision on all models, while a more common word like "hun" ("she") has very low recall overall, and 0 for NorBERT. The same goes for "de" ("they") and many of the pronouns that do not start a sentence, whereas "Hun" ("She") and "De" ("They") with upper-cased first letter score a lot better, with recall and precision mostly in the range 60 to 100 %. When they do not start the sentence, these pronouns are probably used a lot without being part of an entity, whereas when a pronoun starts a sentence it is more often the subject, which has a higher chance of being an entity. On the other hand the lowercase word "filmen" ("the film") has somewhat higher precision than "Filmen" ("The film") for XLM-R and the GRU, but it still has a lot lower recall, indicating that an uppercase first letter is seen a strong feature by the models.

The determiners also seem pretty hard for the models, with very low recall for "Det", "det", "Dette" and "dette" ("That", "that", "This" and "this"). These words are so common that it is probably hard to understand exactly where they are an entity for targeted sentiment and where they are not, whereas less common words like "konserten" ("the concert") "Diskanten" ("The treble"), "musikken" ("the music"), "Sistnevnte" ("The latter") and "Konsollen" ("The console") all have very high precision

	XLM-R		GRU		BERT		
Entity	TP	P	TP	P	TP	P	GP
Den	9	14	4	8	8	13	13
Han	9	11	7	9	7	11	11
filmen	7	7	8	12	5	9	16
Filmen	7	8	3	6	7	7	8
de	1	5	2	3	5	14	14
han	4	7	1	1	4	5	12
den	4	7	0	1	4	5	12
Hun	3	5	3	5	3	4	5
De	3	5	2	3	3	8	3
film	2	3	3	5	2	3	7
Grafikken	3	3	3	3	3	3	3
Hot Chip	2	3	3	3	2	2	4
hun	1	2	2	4	0	0	6
musikken	2	2	2	2	2	2	3
Det	1	6	0	0	0	3	4
det	0	3	0	0	0	3	2
Dette	0	1	0	0	0	1	4
dette	0	1	0	0	0	0	2
Jeppe	2	3	1	2	1	2	3
Sistnevnte	2	2	2	2	2	2	2
Konsollen	2	2	1	1	2	2	2
Metallica	2	4	0	1	1	1	3
konserten	2	2	0	0	2	2	4
Diskanten	2	2	0	1	2	2	2
dem	2	2	0	0	2	2	3
konsollen	1	2	1	4	0	1	2
,	0	2	0	0	0	14	0
.	0	3	0	0	0	5	0
«	0	1	0	2	0	3	0
»	0	0	0	3	0	2	0
«	0	1	0	0	0	1	0

Table 11: The entities with most predicted occurrences among the models XLM-R-large-conll3, the GRU and NorBERT 2. TP means true positives, P means predicted and GP means gold positives. The table is ordered by the sum of each row, except for the quotation marks, which are put at the bottom.

and quite high recall. Another point is that all of these are definite noun phrases, and when they are mentioned in a review, it is usually because one is describing them in some way, which could include sentiment. The mentioned words actually score better than many of the names, like "Metallica", "Jeppe" and "Hot Chip".

It also seems like the precision for the entities in the table is better than the recall, which is in line with the overall precision being higher than the recall. The number of predictions is usually lower or equal to the number of gold entities. Quotation marks has been mentioned earlier in the paper as something transformer models do not deal with very well, and that is especially notable with NorBERT, which predicted 14 commas to be an entity as well as 5 periods. The GRU and XLM-R also made some such predictions, but not as many.

6 Conclusion

These experiments found that the cross-lingual model XLM-R-large, trained on English NER performs best for the Norwegian TSA task on the $\text{NoReC}_{\text{fine}}$data set, outperforming NorBERT and the baseline GRU. However, the simpler GRU was not too much shy of the transformer models and performed relatively well on the span extraction, while lacking more in the sentiment analysis part.

When it comes to these models' predictions, they consistently predict shorter entities than the gold entities, and tend to predict the first word of a sentence as an entity at a higher rate than is the case for the gold data. Words that occur seldom are also easier to predict for the tested models than words that occur often. The transformer models are also bad at understanding punctuation rules, which is reflected in their predictions.

7 Future work

As mentioned earlier, it would be very interesting to see if using a model trained on constituency parsing, or even parsing the constituents of the text as a sub-task before doing the TSA, would increase the performance. Having parsed the constituents would also enable one to do even more interesting error analysis on what kind of constituents the models manage to predict correctly or not. For the cross-lingual models it would also be very interesting to see how much they can generalize across languages for the TSA task, e.g. by training it on an English data set and testing it on $\text{NoReC}_{\text{fine}}$.

References

Anton Chernyavskiy, Dmitry Ilvovsky, and Preslav Nakov. 2021. Transformers: "The End of History" for NLP? Technical Report arXiv:2105.00813, arXiv.

Han-Cheol Cho, Naoaki Okazaki, Makoto Miwa, and Jun'ichi Tsujii. 2013. Named entity recognition with multiple segment representations. *Information Processing & Management*, 49(4):954–965.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. *arXiv:1911.02116 [cs]*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. Technical Report arXiv:1508.01991.

Andrey Kutuzov, Jeremy Barnes, Erik Velldal, Lilja Øvrelid, and Stephan Oepen. 2021. Large-Scale Contextualised Language Modelling for Norwegian. *arXiv:2104.06546 [cs]*.

Xin Li, Lidong Bing, Piji Li, and Wai Lam. 2019. A Unified Model for Opinion Target Extraction and Target Sentiment Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6714–6721. Number: 01.

Manish Munikar, Sushil Shakya, and Aakash Shrestha. 2019. Fine-grained Sentiment Classification using BERT. *arXiv:1910.03474 [cs, stat]*.

Fábio Rodrigues Pereira, Per Morten Halvorsen, and Eivind Grønlie Guren. 2021. Applying Multitask Learning to Targeted Sentiment Analysis using Transformer-Based Models. In *Proceedings of the Third IN5550 Workshop on Neural Language Processing*.

Egil Rønningstad. 2020. Targeted Sentiment Analysis for Norwegian text. Master's thesis, University of Oslo.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *arXiv:cs/0306050*.

Maite Taboada. 2016. Sentiment Analysis: An Overview from Linguistics. *Annual Review of Linguistics*, 2(1):325–347.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2020. A Fine-grained Sentiment Dataset for Norwegian. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5025–5033, Marseille, France. European Language Resources Association.

IN5550 - Final Exam
Neural Machine Translation

Henrik Syversen Johansen

Abstract

This paper focuses on low-resource translation from Norwegian to English, with an emphasis on the importance of model and training parameter tuning in such a setting.

1 Introduction

One of the most researched fields in natural language processing these days is neural machine translation. State of the art approaches to neural machine translation, like transformer based architectures (Vaswani et al., 2017), often take large amounts of data to train. The lack of parallel language data in smaller languages can thus serve as a bottleneck for the performance of translation models that include those languages.

In this paper, rather than focusing on architectural, or data oriented solutions to the bottleneck problem as in other work (Chen et al., 2019; Chaudhary et al., 2019), I focus on the importance of parameter tuning.

2 Background

2.1 Sequence to Sequence Translation

The general approach to neural machine translation is to employ a sequence to sequence neural architecture. That is, a neural network architecture that takes as inputs a sequence and also outputs a sequnce. While most network architectures can be persuaded to fit this definition in principle, the most commonly used are: Recurrent neural networks of varying types, convolutional neural networks and Transformers (Vaswani et al., 2017).

Transformers are particularly interesting as they serve as the backbone of several of the current state of the art natural language processing architectures such as GPT-3 (Brown et al., 2020). In this paper, a transformer based architecture is used.

2.2 Generating Translations

When generating translations using a transformer based sequence to sequence model, there are several approaches. In this paper two will be considered, both of which are autoregressive, meaning the model iteratively generates outputs based on the models previous output. The difference between the two appraoches lies in how the next output is determined.

The first approach is termed Greedy Search and simply selects the most likely token at any given point. The second approach is called Beam Search and is more complex. It keeps track of the n most likely sequences generated, and selects the most likely sequence among them as a whole at the end of generation. This n is referred to as the "beam width". The latter approach usually gives better results, but also takes longer to compute.

2.3 BPE Tokenization

When encoding language information as quantifiable numeric data that is appropriate for use in neural networks there are different availble approches. A very rudimentary approach simply maps each different word in the corpus to a unique number. The downside to this is that unless you are comfortable with a large vocabulary size, which in turn usually means a massive embedding layer, you are bound to see a decent number of words not represented by models the vocabulary.

An alternative approach, BPE tokenization (Sennrich et al., 2015), instead iteratively grows a set of subwords of increasing length until a defined vocabulary size is reached. The subwords added at each iteration are determined by frequency of occurence. When tokenizing, the algorithm splits words according to their largest subword components, thus making it able to represent more words with a smaller vocabulary.

2.4 BLEU metrics

Measuring the quality of a translation is not a solved problem. A given sentence in one language may very well have several valid translations in another; or no precise translations at all. The ideal way to measure the quality of translations, would be to ask bilingual speakers what they think, and then average their opinion in some way. While this may be a good way to quality test a model that is about to be used in some production pipeline, it is much to time consuming when developing the model. Instead a metric called BLEU score is often used (Papineni et al., 2002). BLEU scores have been shown to roughly correspond to scores given by humans, and thus serves as a good automatic metric for measuring the quality of machine translations.

3 Datasets

In short, data from four sources is used. The first consists of sentence pairs from two publicly availble subcorpora. One being the bilingual English-Norwegian parallel corpus from the Office of the Auditor General website. And the other being from the Public Bokmål-English Parallel Corpus. Together these two corpora comprise our first dataset, which consists of high quality translation, but mostly of offical language, rather than everyday speech. This dataset is referred to as the *Government* dataset.

The second dataset used is openly sourced movie and TV subtitles from opensubtitles. The translations here are generally of worse quality, as many of them are submitted by users who aren't necessarily expert translators. However, the dataset is much larger, and the language used is more common, as opposed to the professional language in the Government dataset. This dataset is hereby referred to as *Subtitles*.

The third dataset is used only for testing. It is based on the sentence by sentence translation of Arthur Conan Doyle's book, "Hound of the Baskervilles". The translation is sourced from FarkasTranslations.com. This dataset is referred to as the *Book* dataset. The language, and sentence structure used in the book, is quite different from the data the model will be trained on, so it serves as a test to see how well the model generalizes to different domains.

3.1 DIY Test Set

Finally I use a custom test set consisting of translated lyrics from The Beatles' "Sgt. Pepper's Lonely Hearts Club Band" album. They were translated using google translate, before being manually corrected to make sure the lyrics made some sense also in Norwegian. The resulting test set contains 394 sample sentences, with 89 being duplicates due to the nature of song lyrics. The sentences are split according to lines of the music, not according to how real sentences are structured (as punctuation and sentence structure is often dubious in lyrics).

This test set is interesting because musical and poetic lines, while often relatively simple and short, also regularly violate the rules of grammar and sentence structure for the benefit of lyricism. Also, non-words such as F'Ooh", "Aaah" and "Ho!" are often used, and some sentences do not even make much semantic sense, such as "Lucy in the Sky With Diamonds". This differs greatly from the complex terminology and non abstract language found in government text, and to a lesser extent translated subtitles.

4 Model Overview

The translation system consist of a BPE tokenizer from the tokenizers library with a vocabulary size of 10,000 tokens, a basic pytorch embedding network, a transformer (Vaswani et al., 2017) modified by David Samuel as described in the assignment text, and a linear classifier head. In the base configuration the weights of the embedding network and the classifier head are shared.

4.1 Implementation

The system is implemented as a basic pytorch model, but for training, validation and logging purposes the model is wrapped in a Pytorch Lightning module.

In order to generate text sequences the model, as a default, uses a beam search with a beam width of 3. As this is done iteratively rather than in parallel however, it takes much longer than simply calculating the cross entropy between the predicted and actual tokens as during training. Calculating cross-entropy loss is 62 times faster to be exact. Therefore, during validation as well as training, no search or generation is made. Instead just cross-entropy loss is used in both cases. For testing however, sequences are generated using beam search and BLAU scores are calculated using the torchmetrics library.

5 Baseline Model Search

Initially to find a decent baseline model, increasingly complex models were trained on the smallest government training set and tested on the book validation set. The results of these experiments are in Table 2.

All of these modelse were trained using cross entropy loss and the AdamW optimizer (Loshchilov and Hutter, 2017), with the default weight decay of 0.01. In all the baseline examples the learning rate is not scheduled, instead it is kept constant. This constant learning rate, is found using pytorch lightning's lr_finder function. It works by trying an exponentially increasing learning rate for a set number of mini-batches, and measuring the loss score. The resulting learning rate vs. loss plot, can guide a reasonable learning rate (often some experimentation is required), or even a range to pass through when using a scheduler. The models were all trained for 16 epochs with a batch size of 4096.

Among the model configurations that performed best on just the small government set, two sufficiently different ones (in bold) were selected to be trained on the larger combined dataset. In addition, two more complex models were also trained to evaluate the extent to which more complex models benefitted from the increase in data. The results of this experiment is in Table 3.

The most complex model, with 6 layers, performed the best but also took twice as long to train as the second best with 3 layers, for only a marginal benefit in BLEU score. As such the second best model was selected as the baseline. This baseline was subsequently trained on the three different training sets and the corresponding BLAU scores logged on both the book and DIY test sets. These scores are in Table 4.

Dataset	Train	Validate	Test Size	Avg. NO Length	Avg. EN Length
Government	50,000	2,500	–	13.85	16.81
Subtitles	250,000	2,500	–	6.16	6.76
Book	–	–	2,500	13.53	13.83
DIY (Beatles Album)	–	–	394 (305 unique lines)	6.34	6.50

Table 1: Dataset information

Runtime	d_model	layers	num_heads	# trainable params	4-gram-BLEU (book)
12 min	**32**	**6**	**4**	457 K	**0.028**
15 min	32	8	1	500 K	0.028
9 min	**256**	**3**	**4**	6.5 M	**0.028**
9 min	32	3	4	393 K	0.026
9 min	256	3	4	6.5 M	0.024
10 min	256	3	8	6.5 M	0.024
8 min	128	3	4	2.3 M	0.023
7 min	64	2	2	817 K	0.020
14 min	256	6	4	10.5 M	0.019
6 min	32	1	1	351 K	0.019

Table 2: Comparison of model sizes trained on the government dataset

d_model	layers	num_heads	Runtime	4-gram-BLEU
256	6	8	82 min	0.111
256	3	4	39 min	0.110
256	3	8	40 min	0.104
32	6	4	27 min	0.081

Table 3: Two of the better models from the initial config search, as well as a more complex one as a sanity check trained on the combied dataset.

train set	Runtime	4-gram-BLEU (book)	4-gram-BLEU (DIY)
combined	39 min	0.110	0.276
subtitles	28 min	0.096	0.201
government	9 min	0.032	0.051

Table 4: Baselines for the best performing model trained on all three datasets.

6 Experiments

With a decent baseline established, several experiments were performed in an attempt to improve the model's performance. The training and validation loss of each of these experiments are shown in Figure 1. Also the 4-gram-BLEU scores of each experiment is listed in Table 5.

Rather than performing an exhaustive grid search with each modification tested in all combinations with the others, modifications were added cumulatively in the order described in the following sections and retained for subsequent experiments only if they showed an improvement. This was done to save time and resources.

As the baseline model contains 6.5 M trainable parameters, all of the experiment models also do, except for the model with free classifier weights, it has 9.1 M trainable parameters. The models without any dropout were trained for 16 epochs, while the ones with were trained for 20 epochs. In all cases the batch size used was 4096.

6.1 Small Initial Embedding Weights

The first experiment is a trick I discovered in the github repo of an independent AI researcher using the moniker BlinkDL (their results are not published apart from on their public github). They had observed that initializing

the embedding weights with a uniform distribution from $-1-e4$ to $+1e-4$, rather than the default normal distribution $N(0, 1)$, improved convergence. After trying this the model converged much faster, and settled at a lower final validation loss, seemingly confirming BlinkDL's observation (this can be seen clearly in Figure 1b).

The reasoning provided by BlinkDL as to why this works, is mostly empirical. They observed that when training a transformer the embedding weights often move slowly, by their reasoning: making it difficult for the model to move from the initial noisy embedding. I have not validated this for myself, but the improved convergence at the very least shows the usefulness of the trick in my particular setting.

6.2 LR Scheduling

An LR scheduler was then added, namely the OneCycleLR scheduler (Smith and Topin, 2017). It works by initially increasing the learning rate to a peak, before decaying it until the last epoch, following a cosine curve. This furter increased BLEU scores, but the model begun overfitting earlier.

In the paper describing the OneCycleLR (Smith and Topin, 2017), they describe something they call super-convergence, where the model converges in much fewer iterations than normal. This does not seem to be what is happening in the case of this system. However: the "warm-up, decay" curve of the OneCycleLR scheduler is reminicient the phases of the scheduler proposed in the original transformer paper (Vaswani et al., 2017). An added benefit of having a moving learning rate is the decreased need of finding a good initial one, instead we can settle with finding a good range and let the scheduler do the rest.

6.3 Dropout

After introducing the learning rate scheduler the model now started to overfit, Dropout was added both in the transformer itself as well as in the tokenizer (in the form of BPE dropout). With the concrete dropout probabilities being 0.1 in both cases. This reduced overfitting but slowed down training. Because of this training time was increased by 25%, giving the model time to settle. With dropout the BLEU scores increased significantly, especially in the last added training iterations.

6.4 Label Smoothing

Label smoothing was then added to the cross entropy loss function, however, performance actually decreased a bit compared to the previous configuration (Szegedy et al., 2015). It is possible that, given even more time to train the model would improve past the previous iteration, however training time was already becoming a limited resource so the modificatino was discarded in the next experiment.

6.5 Free Classifier Weights

In the base configuration the weights of the embedding network and the linear classifier head are shared. However, as an experiment the weights of the classifier layer were split from those of the embedder, allowing them to tune on their own. Performance did not increase however. The model performed slightly worse than with the previous configuration, and took longer to train due to the extra parameters.

Again, as in the case of label smoothing, it is possible that given more time this configuration would outperform the others; in this case it would be prudent to validate whether or not this is simply due to the additional trainable parameters by comparing it to a deeper model (with as many trainable parameters) with non-free classifier weights.

6.6 Best Configuration

As can be seen in Table 5, the best performing model was the one with small intial embedding weights, the OneCycleLR learning rate scheduler and dropout. As such, this model configuration is hereby referred to as $M_{+dropout}$.

7 Comparing Search Methods

In the default system configuration, inference is made using a 3 beam width beam search. As can be seen in Table 6, there is a marginal improvement in BLAU score when increasing the beam width to 6, but also a two-fold increase in inference time. The greedy search, despite being twice as fast as the 3-Beam search, looses out quite substantially in performance. As such 3-Beam was selected as a good tradeoff between efficiency and prediction quality.

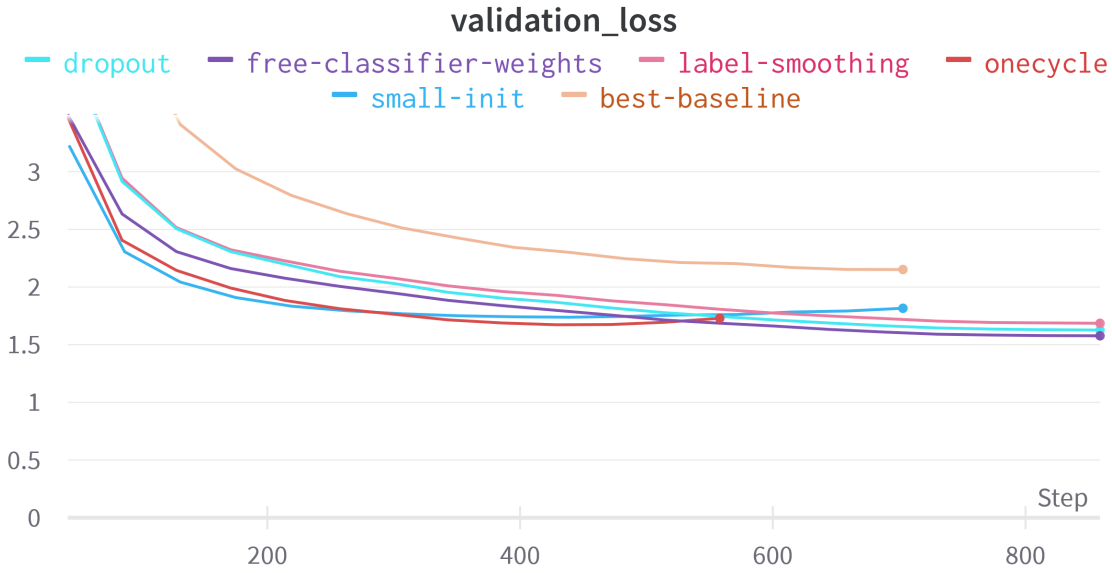
8 Example translations

Several translation examples are listed in Table 7. The best translations are, not surprisingly, from the government dataset on which the model was trained. The model seems to be able to translate not only from relatively complex Norwegian compound words to pairs of English ones (such as from "økonomibestemmelsene" to "financial provisions"), but also between Norwegian and English sentence structure, which often differ.

The model's successful translation of the Beatles lyrics, may be because they are fairly short sentences, containing mostly simple words that are likely to be within the vocabulary.

Some of the worst translations are from the out-of-domain book dataset, which also makes sense as none of the samples were seen during training, and that most of the samples are from an entirely different context from what the model has trained on. The model appears to make its worst errors when attempting to make sense of words it has not seen, or only seen part of (due to the BPE tokenizer splitting off common parts of words).

(a) Training loss of the experiments



(b) Validation loss of the experiments

Figure 1: Training and validation loss of all the experiments listed in Section 6. The steps on the x axis denote optimizer steps, which in this case is equal to batches.

Small init	OneCycleLR	Dropout	Label Smoothing	Free cls weights	4-gram-BLEU (book)	4-gram-BLEU (DIY)
✓	✓	✓			0.164	0.357
✓	✓	✓		✓	0.161	0.336
✓	✓	✓	✓		0.161	0.381
✓	✓				0.138	0.332
✓					0.131	0.348
					0.110	0.276

Table 5: 4-gram-BLEU scores for each experiment.

Method	Runtime	4-Gram-BLAU (book)
Greedy	11 sec	0.1568
3-Beam	24 sec	0.1640
6-Beam	43 sec	0.1641

Table 6: Comparison of search approaches during inference.

For example: “årstallet” is translated to “annual figure”. Both “års” to “annual” and “tallet” to “figure”, make some sense on their own, but not in this context. The translation of “overgår” to “overlooking”, is also interesting in a similar way. The model clearly understands that “over” translates to “over”, but then guesses at what the rest of the word might mean.

This misunderstanding of portions of words is likely due to the BPE tokenizer splitting them. Thus a reasonable path for further experiments (given enough resources) would be an increased vocabulary size.

9 Conclusion

In this paper I opted to focus on the importance of hyperparameter tuning and training parameter tuning. I’ve shown that just dialing in good parameters can take a baseline model from a poor BLAU score of 0.11 to a better, but still modest, 0.16.

A possible future direction is to combine the hyperparameter tuning approaches used in this paper with the architecture experiments from papers such as (Chen et al., 2019). As well as with data augmentation techniques such as synonym replacement.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Vishrav Chaudhary, Yuqing Tang, Francisco Guzmán, Holger Schwenk, and Philipp Koehn. 2019. Low-resource corpus filtering using multilingual sentence embeddings.

Peng-Jen Chen, Jiajun Shen, Matt Le, Vishrav Chaudhary, Ahmed El-Kishky, Guillaume Wenzek, Myle Ott, and Marc’Aurelio Ranzato. 2019. Facebook ai’s wat19 myanmar-english translation task submission.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Leslie N. Smith and Nicholay Topin. 2017. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the inception architecture for computer vision.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Norwegian	**English, Manual Backtranslation**	**English**
Some good examples:		
I slike tilfeller må du derfor straks melde fra til NAV.	In such cases you must immediately notify NAV.	In such cases, you must therefore notify NAV immediately.
Mange av avtalene ivaretar dermed ikke kravene til rapportering i økonomibestemmelsene.	Thus, many of the agreements don't meet requirements for reporting in the financial provisions.	Thus, many of the agreements don't fulfil the requirements for reporting in the financial provisions.
"Nei, langtfra, min kjære Watson — på ingen måte.	No, far from, my dear Watson, in no way.	" No, far from, my dear Watson, in no way.
"Det er interessant, ikke sant?"	It's interesting, isn't it?	" It's interesting, isn't it?"
Jeg pleide å være sint ung mann	I used to be angry young man	" I used to be angry young man
Jeg gjemmer hodet i sanden	I'm hiding my head in the sand	I'm hiding my head in the sand
Stopper tankene mine fra å vandre	Stop my thoughts from wandering	" Stop my thoughts from wandering
Some bad or mixed examples:		
Vold i familien	Violence in the family	Wild in the family
"Nokså fornuftig!" sa Holmes.	"Quite reasonable!" "Holmes"	" Enoughly reasonable!" " Holmes ".
"De overgår virkelig Dem selv, Watson," sa Holmes, skjøv stolen tilbake og tente en sigarett.	"You've truly outdone yourself, Watson", said holmes, pushed the chair back and lit a cigarette.	" They're really overlooking yourself, Watson," said Holmes, roof the chair back and thought a cigarette.
Jeg setter årstallet til 1730."	I'm putting the year at 1730.	I'm putting the annual figure to 1730."
Dumme mennesker løper rundt	Stupid people running around	Runny people run around

Table 7: A table of example translations, both from the in domain government dataset and the out of domain book dataset and DIY dataset. These results were produced using the best performing experiment model $M_{+dropout}$, described in Section 6.3.

Task-Oriented Semantic Parsing with Pointer-Generator Network

Huiling You
University of Oslo, Norway
huiliny@ifi.uio.no

Abstract

Semantic parsing is key for virtual assistants (e.g. Amazon Alexa, Apple Siri, and Google Assistant) to understand users' utterances and take corresponding action(s). Traditionally, the task-oriented intent and slot filling is solved with rule-based or statistical systems. Semantic parsing with shift-reduce parsers has shown promising results on hierarchical representations of queries. More recently, sequence-to-sequence models with pointer-generator module have achieved state-of-the-art results in parsing both simple and complex queries.

In this work, we demonstrate the usefulness of pointer-generator network in task-oriented semantic parsing with sequence-to-sequence models. We experiment with sequence-to-sequence models w/o pointer-generator module, and show that adding pointer-generator module greatly improves the results of naive sequence-to-sequence models.

1 Introduction

Virtual assistants such as Amazon Alexa, Apple Siri, and Google Assistant have played an important role in people's everyday life. At the core of these voice assistants is the ability to process users' utterances and perform corresponding tasks, such as play song and set reminder. Traditionally, a slot-filling system would classify the *intent* first and then tag the necessary slots (Mesnil et al., 2013; Liu and Lane, 2016). A more complex query with multiple intents and nested slots would make such slot-filling systems fail easily.

A Task Oriented Parsing (TOP) representation for intent-slot based dialog systems was recently introduced by Gupta et al. (2018) to capture complex nested queries. Figure 1 shows such a complex query with intent `IN:GET_RESTAURANT_LOCATION` nested inside slot `SL:DESTINATION`. The structure of TOP representation is so similar to standard constituency parses that the modelling techniques of constituency parsers can be easily adapted to solving this problem. Gupta et al. (2018) use linear-time Recurrent Neural Network Grammar (RNNG) (Dyer et al., 2016) to significantly improve the accuracy compared with strong sequence-to-sequence (seq2seq) models, and Einolghozati et al. (2019) further propose three approaches to improve their model. A seq2seq model with pointer-generator network has recently been proposed by Rongali et al. (2020) as a unified architecture to handle complex queries and has achieved state-of-the-art (SOTA) on various public datasets.

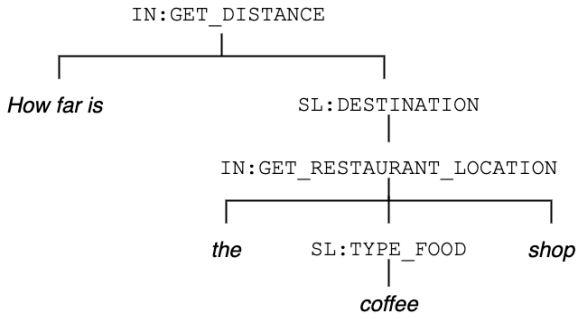


Figure 1: Semantic tree for utterance "How far is the coffee shop".

In this paper, we examine the usefulness of pointer-generator network in task-oriented semantic parsing with seq2seq models. We experiment with seq2seq LSTMs on the TOP dataset. We find that a seq2seq LSTM model with the pointer-generator module greatly improves the results generated by the same model without the pointer-generator module.

2 Related Work

Task-Oriented Semantic Parsing has been a well-researched field since the 1990s since the release of the ATIS dataset (Price, 1990). The task was traditionally approached by jointly classifying the intent and tagging the slots with se-

quence tagging models (Mesnil et al., 2013; Liu and Lane, 2016). Another line of work features neural shift-reduce parsers (Gupta et al., 2018; Einolghozati et al., 2019) to handle more complex *compositional* queries. More recently, Rongali et al. (2020) propose a seq2seq model using a BERT encoder (Devlin et al., 2019) and a transformer (Vaswani et al., 2017) decoder with a pointer-generator module, and present SOTA results on three public datasets, which are TOP (Gupta et al., 2018), SNIPS (Coucke et al., 2018), and ATIS (Price, 1990).

Pointer-Generator Network was first introduced by Vinyals et al. (2015) and later widely used in NLP tasks where some words in the source texts also appear in the target texts such as summarization and style transfer (Paulus et al., 2018; Prabhumoye et al., 2018; See et al., 2017). Jia and Liang (2016) adopts a seq2seq RNN with an attention-based copying mechanism for efficient data recombination to boost accuracy on three semantic parsing datasets.

3 System Description

We use simple seq2seq LSTMs as our baseline model for parsing a source sentence into a target semantic tree. For the pointer-generator model, we add a pointer-generator module to the LSTM decoder. In this section, we describe how the queries are formulated and the corresponding parses with pointers for our model, and continue by describing the encoder and decoder components.

3.1 Query Formulations and Parses with Pointers

A seq2seq model is trained on samples with paired source-target sequences. In order to implement the pointer-generator module, we replace words that are "copied" from the source texts in the target sequences with corresponding *pointers* identifying their positions in the source texts.

Take the example query from Figure 1. In our model, the target sequence is reconstructed by combining the intents with the corresponding slots, where each slot also contains its sources. The source-target pair for this example is then transformed into the following.

```
Source:
How far is the coffee shop

Target:
[IN:GET_DISTANCE @ptr0 @ptr1 @ptr2
[SL:DESTINATION [IN:GET_LOCATION
[SL:CATEGORY_LOCATION @ptr3 @ptr4 @ptr5
]SL:CATEGORY_LOCATION ]IN:GET_LOCATION
]SL:DESTINATION ]IN:GET_DISTANCE
```

Here, *@ptri* is a *pointer* to the i_{th} word in the source sentence. Therefore, the span "@ptr0 @ptr1 @ptr2" points to "How far is" in the source text. We also replace the unified closing bracket `]` with custom closing brackets for each intent and slot. Hence, the intent `[IN:GET_DISTANCE` will have a custom closing bracket `]IN:GET_DISTANCE`.

3.2 Encoder

We use a single-layer bidirectional LSTMs as encoder. We use a static word embedding model pretrained with word2vec skip-gram (Mikolov et al., 2013) on English Wikipedia to encode input queries.

3.3 Decoder with Pointer-Generator

Our decoder is a single-layer unidirectional LSTMs. Since the target sequences have been pre-processed into the aforementioned special formats, the decoder embedding is initialized and trained together with the model. The number of entries in the decoder embedding is the sum of tag set size (intents and slots), pointers (to the source text), and special tokens (e.g. "<pad>").

At each decoding step, attention scores of the input query are calculated to further generate the context vectors. These unnormalized attention scores are later concatenated with the decoder output to produce final prediction. The context vectors are then concatenated with the decoder hidden states to generate scores for each word in the tag set vocabulary. The tag set vocabulary contains the necessary semantic parse symbols (i.e. intents and slots), as well as some special tokens.

Finally, the unnormalized attention scores (over the source sequence) and the scores over the tag set vocabulary are combined and fed through a softmax layer to obtain the final probability distribution.

In the example in Figure 2, we are trying to predict a target word after token "@ptr0". As shown in the figure, we compute the scores over the source tokens (light blue, left) and the scores (dark green, right) over the tag set vocabulary. The final distribution is calculated by combining both scores and parsing them through a softmax layer, and we expect the model to predict "@ptr1", corresponding to "far" in the source query.

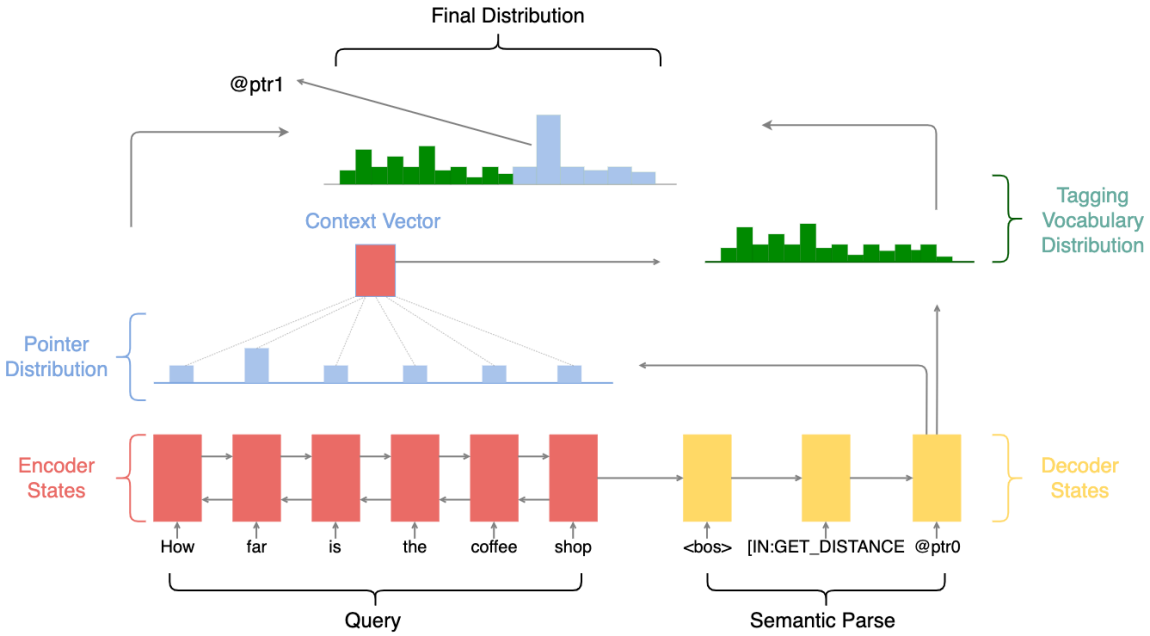


Figure 2: Our seq2seq model with pointer-generator module (SEQ2SEQ-PTR). The model is currently decoding the symbol after "@ptr0", and by looking at the scores over the tagging vocabulary and the attentions over the source pointers, it generates "@ptr1", which points to the second word, namely "far" in the query.

4 Dataset and Evaluation Metrics

Dataset We use the TOP (Gupta et al., 2018) dataset for all our experiments. It contains 44783 annotations with 25 intents and 36 slots. The dataset is randomly split into 31279 training, 4462 validation, and 9042 test utterances.

Evaluation Metrics There are three metrics to evaluate the systems. **Exact match** measures the number of utterances whose parse trees are fully predicted by the systems. **Labeled bracketing** is commonly used in syntactic parsing (Black et al., 1991), and we also use the pre-terminals (i.e. intents and slots) in the calculations. **Tree-Labeled** (TL) F_1 compares sub-tree structures, instead of just the token span. **Tree Validity** measures the percentage of predicted trees which can form valid trees via bracket matching.

5 Experimental Setup

In this section, we describe our detailed experimental setups.

LSTMs For both encoder and decoder, we use a fixed hidden size of 200. Both encoder and decoder contain a single layer. The encoder is bidirectional, but the decoder is unidirectional.

Encoder embedding The embedding model has a vocabulary of size 199,807 and vector dimension of 300.

Decoder embedding The decoder embedding has a vocabulary of size 226 (tag set of size 122, 4 special tokens, and 100 pointers) and vector dimension of 200.

Teacher-forcing We experiment with different teacher-forcing rates during training.

Beam-search We experiment with different beam sizes during decoding, besides the greedy decoding method.

Learning rate We use a fixed learning rate of 0.0006.

Optimization We use Adam to optimize model parameters.

6 Results and Discussion

We present the results of our best Seq2Seq-PTR model in Table 1. Compared with the baseline model, our Seq2Seq-PTR model generates better results over all evaluation metrics, where the exact match percentage is almost doubled. Results also show huge improvements on Tree-Labeled scores, with roughly 30 percentage points on F_1, precision, and recall. In terms of labeled bracketing and tree validity, there are also considerable improvements. With tree validity of 90.68%, the baseline model also presents the strong ability of

Model	Exact match	F_1	Precision	Recall	TL-F_1	TL-Precision	TL-Recall	Tree Validity
Seq2Seq-PTR	75.14	87.82	88.71	86.94	81.41	82.24	80.60	99.58
Baseline	38.56	72.61	62.51	67.18	54.69	47.08	50.60	90.68

Table 1: Results of the baseline model and our best Seq2Seq-PTR model (with teacher-forcing rate of 0.8 and decoing beam size of 3).

Teacher-forcing	Exact match	F_1	Precision	Recall	TL-F_1	TL-Precision	TL-Recall	Tree Validity
0.3	71.26	84.37	89.66	79.66	79.07	84.03	74.66	93.86
0.5	72.38	86.33	88.87	82.06	79.62	82.93	76.57	95.61
0.8	73.63	86.03	88.06	84.09	80.07	81.96	78.27	98.50
0.9	73.35	85.70	87.88	83.64	79.75	81.77	77.82	98.67

Table 2: Results of our Seq2Seq-PTR model with different teacher-forcing rates.

Beam size	Exact match	F_1	Precision	Recall	TL-F_1	TL-Precision	TL-Recall	Tree Validity
2	73.51	86.98	88.20	85.79	80.37	81.51	79.27	99.39
3	75.14	87.82	88.71	86.94	81.41	82.24	80.60	99.58
5	74.50	87.87	88.65	87.10	81.36	82.09	80.65	99.48

Table 3: Results of our Seq2Seq-PTR model using different beam sizes at decoding time.

a basic seq2seq LSTM model. Adding the pointer-generator module indeed greatly boosts the results of a naive seq2seq LSTM model.

In seq2seq modelling, teacher forcing is a useful strategy in training RNN networks. We experiment with different teacher forcing rates in training our Seq2Seq-PTR models. As shown in Table 2, neither a low teacher forcing rate (e.g. 0.3) nor a high one (e.g. 0.9) is the most beneficial choice. In our experiments, using a teacher forcing rate of 0.8 during training produces overall best results. Results also show that using higher teacher forcing rates increase tree validity, but the scores of other evaluation metrics are compromised.

Beam search is a useful strategy at decoding stage of a seq2seq model to generate better outputs. In this work, we experiment with different beam sizes with our Seq2Seq-PTR models. As show in Table 3, using a moderate beam size of 3 produces the best overall results, exact match and tree validity in particular.

The most profound disadvantage of the baseline model is that it generates an output token from the whole output vocabulary. For this reason, the generated semantic parses often contain tokens that do not exist in the source sequences. We take a close look at the semantic parses generated by the baseline model on the test set, and find that out of 9042 instances, 3932 parses contain words that are not present in the source sequences. However, for the Seq2Seq-PTR model, since the output vocabulary is bound by the the source sequences and the tag set, it never generates any token that is not present in the source sequences, apart from the intent and slot labels. As shown in the following example, the semantic parse of the baseline model contains out-of-scope tokens "volunteer opportunies", but the Seq2Seq-PTR model produces the an exact match to the gold tree.

```
Source:
What breakfast locations within
5 miles of me open at 6 am

Baseline parse:
[IN:UNSUPPORTED What [SL:CATEGORY_EVENT
volunteer opportunities ] open at 5 miles
of me Saturday at 6 am ]

Seq2Seq-PTR parse:
[IN:UNSUPPORTED What breakfast locations
within 5 miles of me open at 6 am ]

Gold parse:
[IN:UNSUPPORTED What breakfast locations
within 5 miles of me open at 6 am ]
```

7 Conclusion and Future Work

In this work we present the usefulness of pointer-generator network in task oriented semantic parsing using seq2seq LSTM models. Adding a pointer-generator module greatly improves the results of a simple seq2seq LSTM model over all evaluation metrics. Our experimental results agree with the SOTA model by Rongali et al. (2020), where they incorporate pointer-generator network in seq2seq models and their models outperform previous SOTA shift-reduce parsers.

For future work, we would like to carry on the same line of research and explore pretrained language models and transformers with pointer-generator network. It is also interesting to experiment with different embedding models to encode the target sequences.

References

E Black, S Abney, D Flickenger, C Gdaniec, R Grisham, P Harrison, D Hindle, R Ingria, F Jelinek, J Klavans, et al. 1991. A procedure for quantitatively comparing the syntactic coverage of english grammars. In *Proceedings of DARPA Workshop on Speech and Natural Language*.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer. 2019. Improving semantic parsing for task oriented dialog. *arXiv preprint arXiv:1902.06000*.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In *Interspeech 2016*, pages 685–689.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech 2013*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

Romain Paulus, Caiming Xiong, and Richard Socher. 2018. A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. 2018. Style transfer through back-translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 866–876, Melbourne, Australia. Association for Computational Linguistics.

P. J. Price. 1990. Evaluation of spoken language systems: the ATIS domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. *Don't Parse, Generate! A Sequence to Sequence Architecture for Task-Oriented Semantic Parsing*, page 2962–2968. Association for Computing Machinery, New York, NY, USA.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in neural information processing systems*, 28.

Targeted Sentimental Analysis using Norbert with Layer-wise Learning Rate Decay

Syed Zohaib Hassan
SimulaMet
syed@simula.no

Abstract

The task of targeted sentimental analysis is typically designed as sequence labelling problem. We study the effect of the contextualized and non-contextualized embedding models on extending the work on a baseline bidirectional LSTM. We also introduce a layer-wise learning rate decay during the fine-tuning phase of embedding models. Results show that using large contextualized neural language models can improve the results significantly compared to non-contextualized models. In addition, layer-wise learning rate decay which allows to fine-tune each layer of the pre-trained model with different learning rate. This adds to the performance of contextualized embedding models, that lead to better results compared to using global learning rate for all the layers.

1 Introduction

Sentiment analysis aims to analyze and extract opinions, views, and perceptions about an entity (e.g., product, service, or an action). It has been widely adopted by businesses helping them to understand consumers' perceptions about their products and services. Understanding and discovering the opinions in this online generated data could be useful for lot of applications. Analyzing the opinions in the reviews of the customers on an E-commerce platform could help product deliverer to improve their product or marketing strategy.

The idea of sentiment analysis has been researched in the Natural Language Processing (NLP) widely. Several methodologies and techniques has been developed to understand and extract the sentimental opinion from the stream of text. Positive, negative and neutral are basic sentiments, but there are more complex sentiments on which work has been done as well. As there is a gigantic amount of textual data available, it is impossible to manually adsorb the information about the opinions in the text. Hence, it creates a need for developing an computational framework which can automatically analyze and understand the sentiments in the complex texts. This resulted in the development of various opinion mining sentiment analysis frameworks (Sadr et al., 2019; Liu, 2012).

Traditional sentiment analysis mainly focused on the identifying the sentiment polarity for a sentence or document level (Turney, 2002; Pang et al., 2002; Yu and Hatzivassiloglou, 2003), hence predicting the opinion of the whole document. It is assumed that the complete document or sentence conveys a single sentiment while making a prediction, pragmatically it is not correct to make this assumption.

Research in sentimental analysis for the last few years has focused on the targeted sentiment analysis. On contrast to conventional sentiment analysis, targeted sentiment focus on the target entities and determine the sentiments for them, which makes it more informative analysis.

Targeted sentiment analysis has two underlying sub tasks, namely sentiment classification and entity recognition for each mention of the entity that is applicable to both of these cases. Sentiment analysis is the basic classification task and entity recognition is a pattern matching task. (Mitchell et al., 2013) has introduced an approach where these two tasks are joined together as an extension to the named entity recognition task. It has been shown in the figure 1, where in (a) they are shown as two separate tasks and in (b) it has been collapsed together.

2 Dataset

The dataset used for this task is $NoReC_{fine}$ (Øvrelid et al., 2020), which is fine-grained sentiment analysis dataset in Norwegian. Source of this dataset is review articles across a wide range of domains from multiple news-resources which include movies, products, games, music and many more. In this dataset, text is annotated for sentimental po-

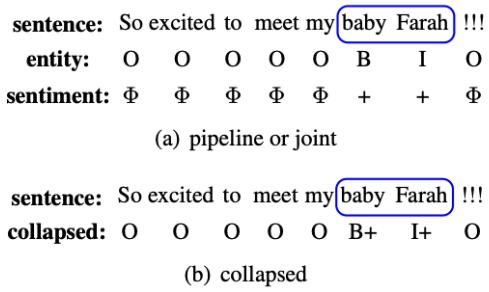


Figure 1: Targted sentimental analysis presented as joint task of sentiment classification and entity recognition (Zhang et al., 2015)

larity, target and holders of the expression at token level. We only focus on the sentiment polarity and target of the sentiment in this final exam submission for IN5550. Table 1 presents the dataset and its annotated targets. The dataset is distributed with pre-defined train, development and test splits.

id	**Train**	**Dev**	**Test**	**Avg. len**
sents	8634	1531	1272	16.8
Targets	5044	877	735	2.0

Table 1: Number of sentences and annotated targets across the data splits

2.1 Data format

The dataset provided for final exam is in conll-format. Each line in the dataset have a token and label which are separated by the tab. Sentences are separated by a new line. The labels are BIO + polarity (Positive, Negative) for a total of 5 labels (B-targ-Positive, I-targ-Positive, B-targ-Negative, I-targ-Negative, O).

Table 2

Labels	**Train**	**Dev**	**Test**
B-targ-Positive	3215	561	516
I-targ-Positive	3389	615	541
B-targ-Negative	1537	252	206
I-targ-Negative	1419	238	155

Table 2: Label distribution in each split

3 Methodologies

3.1 Pre-trained embedding

We use pre-trained embedding from three different models to extract token embedding namely: Word2Vec Continuous Skipgram, NorBERT (Kutuzov et al., 2021) and NorBERT2. Word2Vec model is trained on *Norwegian-Bokmaal CoNLL17* corpus. NorBERT is trained on *Norsk Aviskorpus*[1], *Norwegian Bokmål Wikipedia Dump of September 2020*[2] and *Norwegian Nynorsk Wikipedia Dump of September 2020* corpuses. NorBERT2 is trained on *Norwegian Colossal Corpus (NCC)*[3] and *C4 Web Corpus*[4] corpuses. Embedding dimension for Word2Vec is 100 and 768 for BERT based models.

3.2 Layer-wise Learning Rate Decay

(Zhang et al., 2020) describe layer-wise learning decay as setting the higher learning rates for the higher layers and lower learning rates for lower layers of the language model. It is also termed as discriminative fine-tuning by (Howard and Ruder, 2018) who describe this method as fine-tuning using different learning rates for each layer instead of using a global learning rate for all the layers. Experiments with the learning rate have revealed two insights. First, NorBERT (and BERT in general) is very sensitive to the learning rate, especially when it is fine-tuned. In such cases, the learning rate should be set to 1e-5(that is 0.00001) or a smaller value. Secondly, the set up with the gradually increasing learning rate described in (Howard and Ruder, 2018) has resulted in the best performing model over all experiments we have conducted. The core idea of (Yosinski et al., 2014). is that the lower layers of BERT encode general and common information from the input, while higher layers encode more specific and localized information for the downstream task in hand. Following (Howard and Ruder, 2018), we divide NorBERT layers in three groups and set up different learning rates for each group. BERT has one embedding layer, 12 hidden layers and one pooler layer. We reuse the scaling factors for groups and initialise our learning rate with 1e-5, the default value that has worked well in other experiments. The groups and learning rates are shown in Table 3.

3.3 Evaluation metrics

Evaluation metrics used for this task are Binary Overlap and Proportional Overlap (Katiyar and

[1] http://avis.uib.no/avis/om-aviskorpuset/english

[2] https://dumps.wikimedia.org/

[3] https://huggingface.co/datasets/NbAiLab/NCC

[4] https://aclanthology.org/2021.naacl-main.41/

Group	Layers type	Layers number	learning rate
Group-1	bert.embeddings and bert.encoder	0 to 3	1e-5
Group-2	bert.embeddings and bert.encoder	4 to 7	2e-5
Group-3	bert.embeddings and bert.encoder	8 to 11	3e-5
Group-4	bert.pooler		3.6e-5

Table 3: Fixed set of hyperparameters used for Bi-LSTM experiments

Cardie, 2016). Binary overlap determines the correctness of the overlapping between gold and predicted span. Proportional span determines the overlap with the predicted span as precision and overlap with the gold span as recall, which makes it a more strict metric than binary overlap. We just focus on the binary and proportional for the targets.

4 Experimentation

Baseline provided is a simple Bi-LSTM model which takes in embedding from a pretrained word2vec model. In this section, we discuss the experimental setup of methodologies defined in the section 3.

4.1 System Specifications

The system used for training was a HP Pavilion Gaming Desktop TG01-0 model with an AMD Ryzen 7 3700X processor, 16 GB physical memory, an 8 GB NVIDIA GeForce RTX 2060 Super graphics processing unit (GPU), and a 512 GB SSD.

4.2 Baseline Experimentation

We experimented with fine-tuning embedding models word2vec (58) and use Bidirectional LSTM with different number of layers while keeping the fixed set of hyperparameters shown in Table 4

Hyperparameter	Output
Epochs	20
Dropout	0.2
Batch size	50
Learning rate	1e-2
Hidden layer	100
Loss function	CrossEntropyLoss
optimizer	AdamW

Table 4: Fixed set of hyperparameters used for Bi-LSTM experiments

4.3 NorBERT

We run the experiments for both Norbert models with frozen BERT embedding layers and fine-tuning them by keeping all the other hyperparameters static. We choose the default model to compare different design choices with the results of this default model. These choices are largely inspired by the original BERT paper (Devlin et al., 2018). The following choices have been made for the NorBERT default model:

- If the NorBERT tokenizer outputs more than one token for one input word then first (sub)token is tagged with the true named entity tag while other subtokens are tagged with -100 (padding that will be omitted when computing the loss);
- 1 linear classification layer is used on top of the NorBERT representation;
- last hidden-layer output is used as the representation of the input token;

We crop or pad all sentences up to 128 tokens and fill the longer sentences with an "O" tag for cropped words. We explore the lengths of the sentences in the train dataset and found that all sentences are shorter than 103 CoNLL-U tagged tokens, so we assume that after BertTokenizer we will have 128 tokens at maximum. In addition, we pad the labels with the value -100 that is the default value to ignore by BERT-like models and set the parameter of the loss function to ignore the label -100 so that the padding output does not affect the loss and training progress.

We assign the initial label (corresponds to the non-tokenized full word) to the first subtoken (subword). The rest of the subtokens are labelled as padding. The default choice is to use the first token if tokenization splits the word into more than one token. This option is also used in the original paper by (Devlin et al., 2018). We hypothesise that for named entities the first token is a meaningful

one, because prefixes are not as common in named entities as in general words.

We run experiments with frozen NorBERT layers and train only the classifier layers rate of 1e-2. We have two learning rate strategies; (i) Experiments with fine-tuning the NorBERT embeddings using a global learning rate of 1e-5 for all the NorBERT layers and the classifier layers. (ii) Experiments with discriminative learning rate strategy while fine-tuning the NorBERT embeddings.

We run experiments for this discriminative layer-wise learning rate strategy for the default setting of all the other hyperparameters. To reiterate, the default model uses one linear classification layer, the first subtoken and the last layer output from NorBERT.

5 Results

In this section, we discuss the results quantitatively and qualitatively for our experiments. We also discuss the computational aspect of the training

5.1 Quantitative Results

Table 5 and 6 shows the results for all the experiments conducted. Classifier column shows the type of neural model used after getting embeddings and number of layers. Strategy shows the learning rate strategy. Table shows that we improve the results considerably over the baseline. Best results for each metric are highlighted, where it can be observed that we improve each metric more than double for both target binary and proportional overlap.

We will go through the results step by step moving up from the baseline. Baseline model represents the configuration of word2vec embedding model with single layer BiLSTM without fine-tuning. When we finetune the word2vec embedding model and increase the number of the BiLSTM layers, we observe a slight improvement over the base line as number of layers are increased. Since the training data is not that big and word2vec is non-contextualized embedding model, hence we don't see significant gains for sequence labelling task after the fine-tuning.

When we move from non-contextualized model to BERT contextualized embedding models we start to see a decent improvement in the results. Even without fine-tuning both BERT models (NorBERT and NorBERT2), we see an improvement over fine-tuned word2vec model. Fine-tuning BERT models give us significant improvements compared to all previous experiments.

Results show that the discriminative learning rate strategy tend to perform better than when we use global learning rate for all the BERT layers. We also observe that NorBERT2 tends to perform better than NorBERT, it could be associated with the fact that it has been trained on the significant large corpus than NorBERT.

Figure 2 shows the confusion matrix for the best model, NorBERT2 with BiLSTM followed by one linear layer classifier. As most of token are labeled "O" in the text, model tend to learn to maximize the objective to get prediction of "O" correct. It can be observed that it has least portion of false negatives and it is most falsely predicted class when true label is other than "O". Analyzing the dataset showed us that there are more target labels with positive polarity than negative, as shown in table 2. Hence model is able to predict positive polarity better than negative. There is almost even split between true positives and true negatives for **B-targ-Positive** and **I-targ-Positive** labels. On the other hand, we observe twice true negatives compared to true positives for **B-targ-Negative** and **I-targ-Negative** labels.

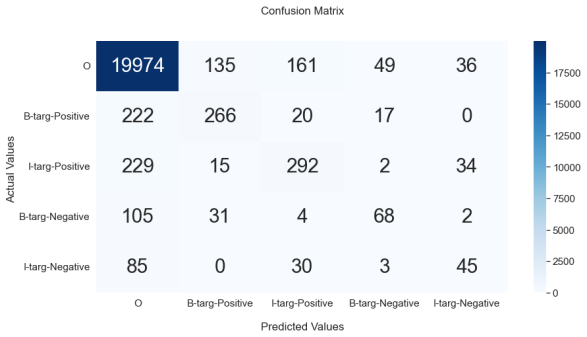


Figure 2: Confusion matrix for the best model that is Norbert2-BiLSTM-Unfreeze-1

5.2 Qualitative Results

Table 7 shows some examples of the text passed into the model for the targeted sentiment analysis. First column shows the input text, target and polarity in the input text is highlighted and second column shows the predicted polarity of the identified target/entity. The first example text is about the good quality of **"Garmin watches"**(target) , our model manages to identify target, it's span and polarity correctly. In the second example text says that **"Garmin"** (target) brand are not that hated

Embedding	Classifier	Fine-tuned	Strategy	Precision	Recall	F1
Word2Vec (Baseline)	BiLSTM-1	False	Global	0.384	0.328	0.354
Word2Vec	BiLSTM-1	True	Global	0.384	0.409	0.396
Word2Vec	BiLSTM-2	True	Global	0.386	0.377	0.382
Word2Vec	BiLSTM-3	True	Global	0.348	0.506	0.412
Norbert	Linear-1	False	Global	0.544	0.405	0.464
Norbert2	Linear-1	False	Global	0.526	0.538	0.532
Norbert	Linear-1	True	Global	0.642	0.584	0.612
Norbert2	Linear-1	True	Global	0.632	0.628	0.630
Norbert	Linear-1	True	Discriminative	0.654	0.606	0.629
Norbert2	Linear-1	True	Discriminative	**0.719**	0.577	0.640
Norbert2	BiLSTM-1	True	Discriminative	0.674	**0.623**	**0.648**

Table 5: Binary Overlap for targets in test data

Embedding	Classifier	Fine-tuned	Strategy	Precision	Recall	F1
Word2Vec (Baseline)	BiLSTM-1	False	Global	0.288	0.204	0.239
Word2Vec	BiLSTM-1	True	Global	0.264	0.229	0.245
Word2Vec	BiLSTM-2	True	Global	0.272	0.195	0.227
Word2Vec	BiLSTM-3	True	Global	0.194	0.223	0.207
Norbert	Linear-1	False	Global	0.449	0.274	0.341
Norbert2	Linear-1	False	Global	0.375	0.263	0.263
Norbert	Linear-1	True	Global	0.534	0.399	0.457
Norbert2	Linear-1	True	Global	0.526	0.439	0.479
Norbert	Linear-1	True	Discriminative	0.538	0.415	0.469
Norbert2	Linear-1	True	Discriminative	**0.599**	0.422	0.495
Norbert2	BiLSTM-1	True	Discriminative	0.555	**0.456**	**0.501**

Table 6: Proportional Overlap for targets in test data

and also they are not stylish, annotated sentiment is negative to this but our model predict it as positive as two parts of sentences convey different polarity about the target. Third text mention **"evil ogres"** as a target entity with negative polarity which our gets it write. In the fourth and fifth example our model gets the target entity right but with wrong polarity.

5.3 Computational Results

The system used was for training was a HP Pavilion Gaming Desktop TG01-0 model with an AMD Ryzen 7 3700X processor, 16 GB physical memory, an 8 GB NVIDIA GeForce RTX 2060 Super graphics processing unit (GPU), and a 512 GB SSD.

Table 8 shows the computational cost for running these experiments for each model configuration. We observe that time taken to finish one epoch for NorBERT models is considerably more than the baseline model. Among BERT models, NorBERT2 is computationally more expensive as it is trained on 15 billion token compared to the 3.5 billion token for NorBERT. Time for NorBERT with bidirectional LSTM followed by a linear layer as classifier is not shown in the table 8. The reason being saga GPU resources are scarce and the PC used for the training was running out of memory, so experiments were run on CPU. It took more than 1 and half an hour for each epochs.

5.4 Conclusion and Future Work

We explore different contextualized and non-contextualized embedding model and, different finetuning approaches for embedding model for downstream task of targeted sentiment analysis. We conclude that contextualized embedding models tends to perform better than non-contextualized models for sequence labelling task like named entity recognition. We used layer-wise learning rate decay strategy for fine-tuning and it gave better results than global learning rate for all the layers of the embedding model. Future work could be including character level information and using conditional random fields with contextualized em-

Input text with gold label of target and polarity	Predicted Polarity
Garmin satser på urmaker kvalitet **[Garmin Fenix Chronos]$_{\textbf{Positive}}$** er meget god tur-og treningskamerat ,og mye flottere enn vanlige smartklokker.	**Positive**
Riktignok har ikke **[Garmin]$_{\textbf{Negative}}$** vært blant de mest forhatte, men de har neppe samlet noen stilpoeng blant de klokkebevisste.	**Postive**
Blir jeg engasjert slik at jeg bryr meg om hvordan det går ? På plussiden er at **[enkelte onde orker]$_{\textbf{Negative}}$** ,til tross for at de ser ti ganger verre ut enn i " Ringenes herre "	**Negative**
[" My Cousin Rachel "]$_{\textbf{Positive}}$ ikke kan sies å være helt på høyde med den gamle mesterens adapsjoner , er det ganske enkelt fordi de avgjørende vendingene i plottet ikke kommer overraskende nok.	**Negative**
Les anmeldelser av sesong 4 : For mye moro Det interessante med **[tidsgrepet]$_{\textbf{Negative}}$** , er at det burde gi seerne en enda mer perfekt unnskyldning til å binge hele sesongen.	**Positive**

Table 7: Examples of the output from the model performing targeted sentiment analysis

Model	Time(sec)
Word2Vec-BiLSTM-Freeze-1	4
Word2VeC-Bi-LSTM-Unfreeze-1	9
Word2Vec-Bi-LSTM-Unfreeze-2	11
Word2Vec-Bi-LSTM-Unfreeze-3	13
Norbert-Linear-Freeze-1	39
Norbert2-Linear-Freeze-1	50
Norbert-Linear-Unfreeze-1	108
Norbert2-Linear-Unfreeze-1	127

Table 8: Time taken per epoch for each model configuration

bedding models.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *a0rXiv preprint arXiv:1810.04805*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Arzoo Katiyar and Claire Cardie. 2016. Investigating lstms for joint extraction of opinion entities and relations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 919–929.

Andrey Kutuzov, Jeremy Barnes, Erik Velldal, Lilja Øvrelid, and Stephan Oepen. 2021. Large-scale contextualised language modelling for norwegian. *NoDaLiDa'21*.

Bing Liu. 2012. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open domain targeted sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2020. A fine-grained sentiment dataset for Norwegian. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5025–5033, Marseille, France. European Language Resources Association.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.

Hossein Sadr, Mir Mohsen Pedram, and Mohammad Teshnehlab. 2019. A robust sentiment analysis method based on sequential combination of convolutional and recursive neural networks. *Neural Processing Letters*, pages 1–17.

Peter D Turney. 2002. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. *arXiv preprint cs/0212032*.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27.

Hong Yu and Vasileios Hatzivassiloglou. 2003. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 129–136.

Meishan Zhang, Yue Zhang, and Duy Tin Vo. 2015. Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2020. Revisiting few-sample bert fine-tuning. *arXiv preprint arXiv:2006.05987*.

Everyday transformer-based translations between Norwegian and English is all you need

Lucas Wagner
University of Oslo
lucas.wagner@uol.de

Tim Sigl
University of Oslo
tim.sigl@uol.de

Abstract

This paper explores the possibility of a lightweight, fast and transformer-based model for translations from Norwegian into English. The aim is to enable fast and mobile translation from a language spoken only by a few people into English. Special focus is given to the handling of sparse vocabulary and small datasets and the size of the model. The experiments carried out show that even with small data sets and small models, measured by the number of parameters and inference time, reasonable to good translations can be achieved. Experiments are performed with both smaller and larger models. A trade-off is made to show that even a reduced number of parameters does not significantly worsen the translation. The BLEU score serves as a benchmark for translation quality.

1 Introduction

Machine Translation has seen a tremendous increase in performance through the additions of neural-based encoder-decoder architectures (Cho et al., 2014). In concept, the encoder encodes the source sentence into a representation for the decoder to use. The decoder uses the encoded representation together with the already predicted sub word tokens to produce a translation of the input sentence (visualized in Figure 1). An emerging issue when using RNNs as the underlying base architecture in the encoder-decoder is that translation performance suffers when translating longer sentences because the whole sentence is just represented by a single state. This problem is solved by the attention mechanism and transformer architecture (Bahdanau et al., 2014; Vaswani et al., 2017). Attention in transformers is facilitated with the help of queries, keys, and values. A key is a label of a word and is used to distinguish between different words. The query checks all available keys and selects the one, that matches best. So it represents an active request for specific information. Key and values always come in pairs. When a query matches a key, not the key itself but the value of the word gets propagated further. A value is the information a word contains.
This work focuses on light-weight models for mobile systems like smartphones. The aim of this paper is not to produce the best performing models in terms of BLEU score. Rather, we propose a trade-off between quality of translation and inference speed.

2 Related Work

The architecture introduced with the transformers has greatly advanced the quality of machine translation. Mao et al. (2021) shows how transformers can also be successfully used on mobile devices by means of downsizing procedures. This is in line with the goal of this work to develop a lightweight model. The work of Murray et al. (2019) shows that a transformer with reduced size does not necessarily result in poorer quality in the translation. This also depicts the general aim of this work. Ranathunga et al. (2021) deals with the problem of the lack of larger, aligned datasets for machine translation. The lack of training data is also a researched topic of this work.

3 Dataset

Three different datasets serve as the data basis for training the model. The first is derived from the Bilingual English-Norwegian parallel dataset from the Office of the Auditor General (Riksrevisjonen) website and the Public Bokmål- English Parallel Corpus (PubBEPC).[1] Thus, this is very formal language with little colloquial vocabulary. From now on it will be referred to as the government dataset. The second dataset is translated subtitles from the website opensubtitels.org,[2] compiled by Lison and Tiedemann (2016). Thus, this dataset reproduces

[1] www.riksrevisjonen.no/, www.nav.no/no/person, www.nyinorge.no/

[2] http://www.opensubtitles.org/

the structures and vocabulary of films and series, so it is closer to colloquial language. From now on, it will be referred to as the subtitle dataset. The government dataset and the subtitle dataset were also merged into a combined dataset. The third dataset is a compilation of everyday sentences. This also reflects the use case of the model. It is intended to be used for translating everyday situations. Thus, the dataset consists of simple, predominantly short and colloquial sentences.The data set was derived from the vocabulary of the online language tool Babble.[3] The samples from the data set mainly represent colloquial language. When selecting the sentences, care was taken to include simple, short sentences as well as longer and more complex sentences. Two sentences are given below as examples. 1) "*Han kan hjelpe deg.*" with the expected translation "*He can help you.*" and 2) "*T-banen går hvert sjette minutt om dagen, og hvert tiende minutt om kvelden.*" with the expected transaltion: "*The subway runs every six minutes a day, and every ten minutes in the evening.*". From now on, it will be referred to as the DIY dataset. It is also the first of two datasets used for evaluation purposes. The second dataset to be evaluated is the translation of the book "Hound of the Baskervilles" by Arthur Conan Doyle. The sentence structures contained in the data are from the year 1902; older and literary language structures are predominantly used.

3.1 Dataset statistics

The data sets used differ in size. For example, the data for training is significantly larger than the two data sets used for validation. However, the average length of the sentences does not differ very much. The combined data set has 7.9 words and the DIY data set 6.7. The book data set has 13.6. Thus, shorter but also longer sentences are tested during the validation.

Name	samples	avg. length
DIY	101	6.7
government	50000	15.3
combined	300000	7.9
subtitles	250000	6.4
book	2500	13.6

Table 1: Overview over the sample size and average sample lenght in the used datasets.

[3] https://my.babbel.com/dashboard

4 Experimentation

To establish a baseline (now called experiment 1 subsection 5.1), a model was trained based on the transformer architecture. The available datasets except the DIY dataset and book dataset served as the training basis. A transformer model was trained on each dataset. It was tested against a validation set of each dataset (in-domain) and the newly assembled DIY dataset and book dataset, out-of-domain. The hyperparameters of the baseline are based on the related work, default values in the PyTorch and Hugging Face library and will be explained in the following paragraph.[4] Adam with a default learning rate of 0.003 was used throughout all experiments. The tokenizer was generated on the basis of the combined training data and the vocabulary size was set to 30.000 subword tokens. The model is composed on an encoder and a decoder with a hidden size of 100. For the baseline, 50 epochs were trained with a batch size of 4096. At the start of the training, the model had a "warm-up phase" of 25 steps. Here, the learning rate is successively increased to the target value of 0.003. This ensures a more stable training at the beginning.

After the baseline was established, variations of the training were started. All variations are trained on 50 epochs. A more efficient training would have been possible since many models loss plateaued before the 50th epoch but for simplicity reasons we choose 50 epoch as a standard for all experiments. The first variation implemented was parameter sharing (experiment 2 subsection 5.2). This ensures that layers share their parameters and the models are reduced in size. The model is optimized for use on mobile devices such as smartphones. A small, low-computing model is therefore the goal of this work. The complexity of the model can only be increased as long as the operability on mobile devices is guaranteed. The sharing of parameters is maintained over all experiments except the baseline.

The most promising step is to increase the number of encoder and decoder layers in the model (experiment 3 subsection 5.3) to allow for more complexity. The number of epochs, the warm-up phase and the learning rate remains unchanged.

The next step is to increase the hidden size (exper-

[4] https://pytorch.org/, https://huggingface.co/docs/tokenizers/api/tokenizer

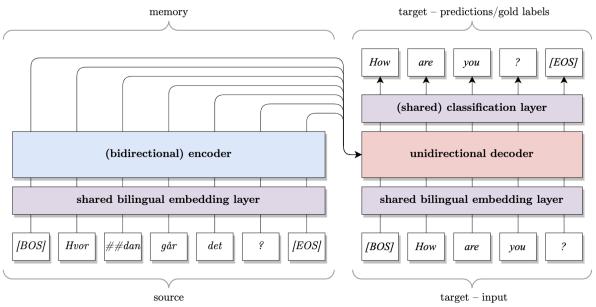


Figure 1: Neural Machine Translation Architecture

iment 4 subsection 5.4) to reduce the difference to the classification layer to prevent a potential information bottleneck between the default hidden size of 100 and the classification output size of 30.000. With an increased hidden size of 200, it was doubled compared to the previous experiment. The next experiment was realized with a reduction in vocabulary size (experiment 5 subsection 5.5). The focus of this paper is on simple basic vocabulary for everyday situations. Therefore, a limitation to a smaller vocabulary is justified. The vocabulary size was reduced from initially 30,000 to 5,000 subword tokens for both languages combined, i.e. to $\frac{1}{6}$ of the size.
The last experiment (experiment 6 subsection 5.6) is a combination of previous experiments with the addition of multiple attention heads. In this run, the hidden size was increased to 300. This size also allows the number of attention heads to be increased to 3. It is hoped that this will lead to a better understanding of the context. The number of encoders and decoder layers is set to 3 and allows for more complexity.

All model trainings were performed on the computing cluster Saga with dedicated GPUs.[5] The model inference was run on a local computer with an Intel i7 CPU to make performance difference between experiments more significant. So all time measurements are based on this particular machine. The unit 'seconds per batch' used in this paper refers to the time measured on this exact local machine and should not be seen as absolute time measurements but rather as numbers in relation to each other. Despite the focus of this work on low-computational mobile devices no actual experiments were carried out on smartphones. Performance improvements achieved on desktop machines will be proportional similar on mobile devices so the experiments done on desktop computers will still stay relevant to the goal of this paper.

[5] https://documentation.sigma2.no/hpc_machines/saga.html

5 Evaluation

Previously described experiments are evaluated in the next sections. The section order follows the numbering of the experiments from one to six. During the generation of the predictions, there are two algorithms that both search for the most likely translation, greedy search and beam search. Greedy search is the faster one, and beam search is more sophisticated. During the evaluation, random samples of both algorithms were tested. It turned out that greedy search is significantly faster, but therefore delivers a poorer result. In direct comparison with the model from experiment 6, trained on the combined dataset and evaluated against the DIY dataset, greedy search achieves a BLEU score of 41.2 (presented in section 5.6). Beam search with a beam size of four, on the other hand, delivered a BLEU score of 46.9 with an inference time of 1.984 batches per second. The better BLEU score of (5.7 points) does not compare to the significantly increased batch inference time (5.2 times), since the aim of this work is to create a particularly fast model. Therefore, Beam search will not be considered any further.
The last section (5.7) summarizes interesting sentence structures and translations.

5.1 Baseline

The metric focus in the comparison of the different models is on the BLEU score and its interpretation (Lavie, 2010). The baseline training results on the combined dataset with the DIY dataset as validation set amount to a BLEU score of 34.8 after 50 epochs (see number one in Figure 3). This value indicates comprehensible to good translations. In comparison, a validation using the book dataset results in a score of 10.1, which, interpreted, means that the model has difficulty understanding the essentials. The big difference in the BLEU score between the DIY dataset and the book dataset should carry through the further experiments. This is explained by the fact that the DIY dataset consists of simple, everyday sentences that are contained in this way or very similarly in every larger dataset, including in the combined dataset. The mixture of the English version of the Norwegian government website and the subtitles depicts formal language

and colloquial speech. The model has no difficulty translating simple sentences. In comparison, the book dataset consists of literary sentences, which are not the same as colloquial or formal language in the combined dataset. This is the reason for the lower score.
If the model is trained on the government dataset only, it achieves a BLEU score of 2.2 when validated by the book dataset, so the translation is almost unusable. The formal sentences of the training dataset seem to be very different from the literary sentences. A validation by the DIY dataset also only achieves a score of 6.17 with the same interpretation. Thus, apparently no sentences of everyday use are found in the training dataset, their structure is unknown to the model.
If the model is trained on the subtitle dataset, it achieves a score of 8.6 in the validation with the book dataset, and 30.1 with the DIY dataset. These results confirm the previous assumption. Subtitles and colloquial language are very close, literary language deviates too much in its structure. Figure 3 depicts the BLEU scores of all models versus their number of parameters. This is a valuable metric since the goal of this paper is to find the best translation while being constrained by low computational power. The optimal model would have a high BLEU score while still having a low amount of parameters. So in Figure 3 this model would be located in the upper left part. A bad model would be located in the bottom right part with a low BLEU score while still having a high amount of parameters.

5.2 Shared Parameters

The parameters of the model can be shared between the embedding layers and the classification layer, since input and output sizes are the same (visualized in Figure 1). Sharing the parameters in embedding and classification layers worsened the BLEU score enormously and was therefore not further considered. Therefore this experiment only examines shared parameters between the embedding layers and not between embedding and classification layers. The BLEU score with shared parameters is a little worse than in the baseline and decreased from 10.1 to 9.4 by 7.4% on the book dataset. On the DIY dataset, the BLEU score decreased by 6.4% from 34.8 to 32.7 (see Table 2). The BLEU score thus decreased on average by 6.9% on both validation datasets. The slight decrease in translation

	1	2	3	4	5	6
DIY	34.8	32.7	45.9	32.4	25.6	41.2
Book	10.1	9.4	12.9	10.9	9.6	14
Val.	21.4	18.6	25.7	23	17	15.2

Table 2: BLEU scores of all experiments on DIY, book and validation datasets. $1 = baseline$, $2 = shared\ params$, $3 = encoder/decoderlayer\ inc.$, $4 = hidden\ size\ inc.$, $5 = decreased\ vocab.$, $6 = mixed$

quality is expected and most likely due to parameters having represent more information with the same weight. The inference speed per batch in seconds increased from 0.12 seconds to 0.13 seconds compared to the baseline. The difference can be attributed to natural variations in execution time. The inference time is therefore considered to be equivalent. An improvement of the inference time was also not expected in this experiment, since the number of computations is the same compared to the baseline but now shared parameters in multiple computations. Only the model size could be reduced from 25.3 MB (6.233.132 parameters) to 13.3 MB (3.233.132 parameters). On end devices with little memory or for use on the Internet, this saving can be significant.

5.3 Multiple Encoder/Decoder Layers

In this experiment, the number of encoder and decoder layers was increased from 1 to 2 to give the model more flexibility. Thus, with a training of 50 epochs on the combined dataset and the validation with the DIY dataset, the model achieved a BLEU score of 45.9. The quality of translation is fairly high and sentences are clear and good structured. This is an increase of over 11 points compared to the baseline with the same data. The increment of the layers resulted only in a minor increase in model parameters compared to the shared parameters experiment from 3.2m to 3.4m parameters (6.3%).
Validating the same model with the book dataset, a BLEU score of 12.9 is achieved, which means that the model still has difficulty understanding the essentials. However, there is also an improvement of almost 2 points compared to the baseline. If the validation data is in domain, the model achieves a score of 25.6, the sentence structures are understood, but there are still grammatical errors. Compared to the baseline, this is an increase of just over 5 points. 2 points improvement with an in domain

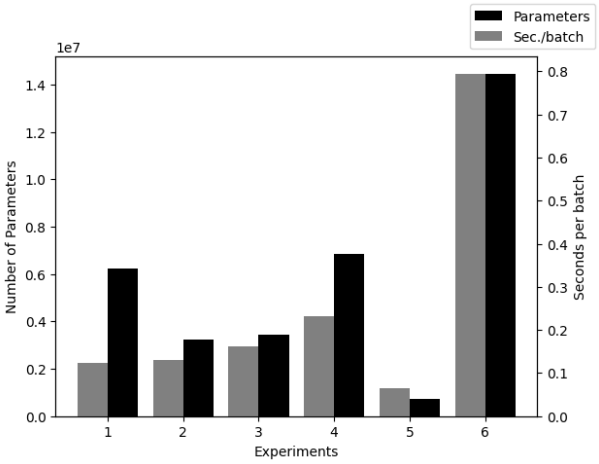


Figure 2: BLEU scores of six different models against their number of parameters. $1 = baseline$, $2 = shared\ params$, $3 = encoder/decoderlayer\ inc.$, $4 = hidden\ size\ inc.$, $5 = decreased\ vocab.$, $6 = mixed$

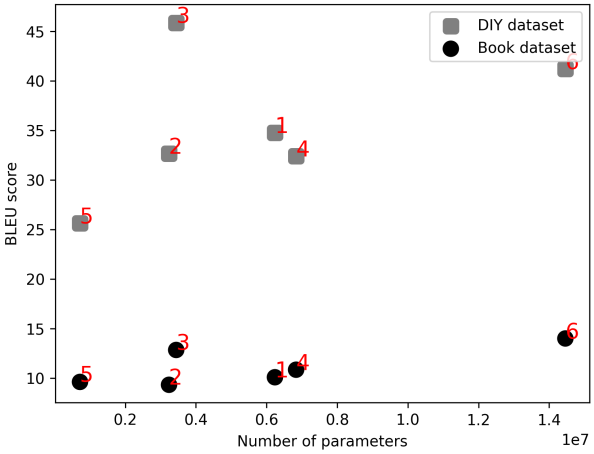


Figure 3: BLEU scores of six different models against their number of parameters. $1 = baseline$, $2 = shared\ params$, $3 = encoder/decoderlayer\ inc.$, $4 = hidden\ size\ inc.$, $5 = decreased\ vocab.$, $6 = mixed$

validation using the subtitles dataset.
In general, it can be said that by doubling the layers, there is a significant improvement in the performance of the model. Thus, this was a useful adaptation.

5.4 Increased Hidden Size

An increase of the hidden size from 100 to 200 results in an increase of all neurons in each layer in the transformer. Thus also increases the number of trainable parameters by a significant amount. The BLEU score of the DIY dataset decreased from 34.8 to 32.4 by 7.4%. In contrast the score of the book dataset increased slightly from 10.1 to 10.9 by nearly 8%. This observation shows that an increase of the hidden size neurons can have a different effect depending on the dataset used for validation. While it worsens the performance on the simple DIY dataset it improves the translations on the more sophisticated book dataset. The increased hidden size also results in an increase of parameters. In comparison to the baseline parameters are increased from 6.2 m to 6.8 m. Parameter sharing is still applied which decreased the parameters before to 3.2 m. So in fact parameters in this experiment nearly doubled (93% increase) compared to the shared parameters experiment. This also has an impact on inference time which increased from 0.12 seconds per batch to 0.23 seconds per batch. The increased is visualized in Figure 2. The needed inference time is the second highest after experiment six (see 5.6) despite number of parameters being very similar to the baseline.

5.5 Decreased Vocabulary Size

In this experiment, the vocabulary size was greatly reduced. On the one hand, a reduction of the vocabulary size affects the number of predictable subword tokens. On the other hand, a smaller vocabulary size also decreases the required parameters in the embedding as well as classification layers. The number of parameters compared to the baseline could be reduced from 6.2 m to 0.7 m and thus to $\frac{1}{9}$ of the original size. The reduction in parameters is not solely due to the smaller vocabulary but is a combination of smaller vocabulary and shared parameters as shown in the experiment 5.2. But even in comparison to the decreased parameters with shared parameters a smaller vocabulary decreases the number of parameters to $\frac{1}{5}$. Reducing the parameters slightly worsens the performance of the model. Compared to the baseline, the BLEU score on the DIY dataset decreased from 34.8 to 32.7 by 6.1% and the BLEU score on the book dataset decreased from 10.1 to 9.6 by 4.7%. The inference speed was halved from 0.12 seconds per batch to 0.06 seconds compared to the baseline. The model size was reduced from 25.3 MB (6.2 m) to 3.2 MB (0.7 m parameters).

5.6 Mixed

The next step was to add another layer, raise the hidden size to 300 and add two more heads of attentions. The layers share their parameters again

and a total number of 14.4 m parameters is reached. When trained on the combined dataset and validated by the DIY dataset, the model achieved a BLEU score of 41.2. Compared to the baseline this is a significant improvement by over 18%. The BLEU score of the book dataset also improved from 10.1 to 14. All those improvements come at a large computational cost. As seen in Figure 2 the number of parameters and inference time per batch are by far the highest. In comparison to the baseline parameters have increased by over 230% and inference time by over 640% to nearly 0.8 seconds per batch. Despite this model generating the best translations for the Norwegian sentences it is not the optimal model measured by our computational constraints.

5.7 Interesting findings

Since language cannot only be assessed on a numerical level, the translations of the results were also examined in terms of content and semantics. For this purpose, the predictions of the model were compared with sentences that had already been translated and evaluated. Some interesting observations are discussed below.

The baseline already provides an interesting result. The sentence: *"You speak Norwegian*" becomes *"You're talking Norwegian"*, so the model chooses the present progressive. The present progressive tense is often overused by non-native speakers of English. It should only be used in the following contexts: To describe an incomplete action which is in progress at the moment of speaking; usually with time expressions such as: now, at the moment, right now. The use of the present progressive to describe current and ongoing things occurs repeatedly in the predictions.

If the sentences become a little more complex, such as: *"I'm going to buy a gift for them."*, the baseline model generates for example the following prediction: "*I'm going to buy them a gift to get to them*". Here, the first part of the sentence is translated correctly, but errors occur in the grammatically more complex second half of the sentence. The proper use of the object "them" was not correctly predicted.

It can be observed that some sentences are simplified. The sentence: "*Have you thought about inviting Jon?*" becomes "*You're going to invite Jon?*" The content remains the same, but the sentence structure is of simpler nature.

The understanding of numbers was only partially learned, "*I have to buy a kilo of sugar.*", becomes "*I'm gonna have to buy a hundred.*". Here the model recognizes that the word kilo represents a number, but cannot correctly assign it, 100 != 1000. With longer, literary and more complex sentences, the baseline model reaches its limits. "*I see in your eyes the same fear that would take the heart of me, A day may come when the courage of men fails, when we forsake our friends and break all bonds of fellowship, but it is not this day*" becomes "*I see in your eyes that fear that would take the heart of me, a day, a day, when we leave our friends and every day we leave all our friends and all day, but we're not all this day.*". But even here the approximate meaning of the sentence is still conveyed to the reader.

However, if the model becomes more complex (experiment 6), it is suddenly able to find synonyms for numbers: the sentence "*In fourteen days, my two sisters will visit.*" becomes "*In a fortnight, my two sisters'll be over - serviced.*". The end of the translation makes no sense here, but the model uses an alternative expression for "fourteen days", namely "fortnight", which in turn is an expression for "fourteen days".

Unlike the baseline model, this model also no longer uses a present progressive when it is not necessary. If "the Island Festival" appears in the labels, the model replaces it with "Øyafreamus festival". Interestingly, this festival really exists.

Noteworthy, there are large differences between the models when they are trained on different datasets. We already know from the experiments that when models are trained on the government dataset, they perform worse when measured by the BLEU score. This finding is also strongly reflected in the translations. Thus, the model's vocabulary is very focused on formal vocabulary and tries to recreate sentences from everyday situations with these words. "*You need a new pair of pants*" becomes "*You need a new set of finances.*". In all its predictions there is a strong influence of its training data, to the point where translations can no longer be understood: "*I am compensated well.*" is the translation for "*I remember it well.*". The original meaning of the sentence is lost.

The last factor to be considered is the quality of the translations during the training. The model from experiment 6 trained on the combined dataset again serves as the basis for the following explanation.

The example sentence is "*The dress code for the wedding is shorts and swimsuit.*". After epoch 20, the content of the sentence can already be guessed: "*The code for the wedding is cargo - shorts and cab.*". Here, misunderstandings could still occur when reading. After epoch 30, however, the message is clearly understandable: "*The code for the wedding is both a shorts and a bathing suit.*". At the end of the training, the translation deteriorates again. At epoch 50 the translation is "*The code for the wedding is shorts and lots of baths"*. Although the sentence is grammatically correct, it deviates strongly from the gold label.

6 Conclusion

This paper compared different transformer-based translation models for translating everyday Norwegian sentences into English. The constraint for this study was a conceptual application running on a smartphone limited in computational resources. Different techniques for decreasing the model's parameters and improving inference time like parameter sharing and a smaller vocabulary size were evaluated. As shown, decent translations can even be generated with a reduced vocabulary size and little parameters. Besides the reduction of computational requirements techniques to make the model larger and thus computationally more heavy were also investigated. Especially adding multiple encoder and decoder layers (see subsection 5.3) increased the translation quality by a lot while only raising the computation burden slightly. With the rather small training data, it was shown that the generated translations are the best in in-domain contexts or similar contexts as the training data. In a real-world application, more training data would be necessary to really bring translations up to a human standard. Another possible experiment is the concept of grid search. This involves the suggestive exploration of the hyperparameter space and can be combined with an evolutionary search algorithm. The BLEU score has already been introduced as a numerical metric. Other rules, such as a maximum number of parameters or a maximum duration of the inference time, are also conceivable. The testing and deployment on smartphones as well as experimentation with larger training data or different base architectures are objectives for future research.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches.

Alon Lavie. 2010. Evaluating the output of machine translation systems.

Pierre Lison and Jörg Tiedemann. 2016. OpenSubtitles2016: Extracting large parallel corpora from movie and TV subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 923–929, Portorož, Slovenia. European Language Resources Association (ELRA).

Jiachen Mao, Huanrui Yang, Ang Li, Hai Li, and Yiran Chen. 2021. Tprune: Efficient transformer pruning for mobile devices. *ACM Trans. Cyber-Phys. Syst.*, 5(3).

Kenton Murray, Jeffery Kinnison, Toan Q. Nguyen, Walter J. Scheirer, and David Chiang. 2019. Auto-sizing the transformer network: Improving speed, efficiency, and performance for low-resource machine translation. *CoRR*, abs/1910.06717.

Surangika Ranathunga, En-Shiun Annie Lee, Marjana Prifti Skenduli, Ravi Shekhar, Mehreen Alam, and Rishemjit Kaur. 2021. Neural machine translation for low-resource languages: A survey. *CoRR*, abs/2106.15115.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Neural machine translation from Norwegian to English

Jan Šamánek, Elina Sigdel, Anna Palatkina
{jansama | elinasi | annapal} @student.matnat.uio.no

Abstract

This paper analyzes and compares several models with various architectures for neural machine translation from Norwegian to English trained on small corpora with different domains and demonstrates the results of evaluation on a newly collected hand-annotated parallel corpus. As baseline we used simple Transformer with a various number of Encoder and Decoder layers. After that we conducted several experiments using NorBERT2 and RNNs.

1 Introduction

Neural machine translation (NMT or simply MT) is one of the most researched subfields of Natural language processing (NLP) these days, which might not be surprising given its direct applicability in practice. Today's state of the art for machine learning translation are Transformers, but Transformers are usually trained on big corpora. What happens if we only have small corpora for training use? Could older RNNs outperformed Transformers on NMT task given only training sets? And could pre-trained components from monolingual NorBERT2 model help? Our focus was mainly based on trying different architectures and using pre-trained components, and creating the best generalizing model possible using only small training corpus.

2 Neural machine translation task

According to paper *Machine translation using deep learning: An overview*[SKD+17], machine translation is one of the subfields in the Natural Language Processing community and its goal is to translate the input sequence from one language to another corresponding sequence in another language. Neural machine translation (NMT) has the same goal but uses statistical models that can be trained directly on the corpus using deep learning. Models used for NMT usually consist of Encoder and Decoder, where Encoder encodes the input sequence, and the Decoder then uses these encoding for context and generating corresponding predictions.

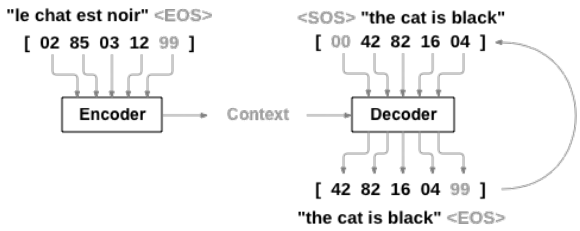


Figure 1: Simplified architecture of the model performing NMT task [PyT]

Today's state of the art for NMT task are Transformers. According to the article by Tomas Wood [Tho], Transformers are the key components in neural networks used today for many tasks like NLP, genome sequence processing, sound signals, and more. Transformers architecture consists of the Encoder and Decoder where Encoder receives the input sequence, encodes it, and gives this encoding to the Decoder which then decodes this encoded sequence to another sequence based on the performed task (in this case translation from Norwegian to English).

For evaluating how satisfactory the machine translation is we used the BiLingual Evaluation Understudy (BLEU)[1] score which is defined as:

$$BLEU = b_p \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

N	order of n-grams,
b_p	sentence brevety penalty,
w_n	positive weights summing to one,
p_n	modified n-gram precisions.

[1] https://cloud.google.com/translate/automl/docs/evaluate#bleu

3 Dataset

As mentioned in the introduction, we worked only with the small corpora. We used 2 datasets for training which differed based on their source, topic, and average sentence length.

The first corpus was made up of two publicly available subcorpora: 1) Bilingual English-Norwegian parallel corpus from the Office of the Auditor General (Riksrevisjonen) website and 2) Public Bokmål-English Parallel Corpus (PubBEPC). In total, this corpus comprises of aligned Norwegian-English sentences built from the public web sites[2][3][4][5]. We consider this corpus of high quality, made by professional translators. This dataset is quite small as it consists of only around 50 000 sentences designed for training. The average sentence length of this corpus is a slightly over 6 words for both Bokmål and English sentences and the topic of this dataset is mostly about a governmental and legal matters. The English level is at an intermediate level at least.

The second corpus has been taken from the OpenSubtitles.org[6] where people provide their translations of movies and TV series. This corpus is about 5 times larger than the Government corpus but also noisier.

As for the evaluation sets, apart from the Government and Subtitle corpora mentioned above, we also used a third small evaluation set as the main metric for evaluating our models. This corpus had around 2,500 sentences and consisted of the sentences from the book of by Arthur Conan Doyle *Hound of the Baskervilles*.

Apart from datasets mentioned above we also created our self-made dataset. We used the CCAligned [EKCGK20] corpus that has been created by the "Commoncrawl Snapshot" organization. This dataset was fairly large but also quite noisy and so we decided to cherry-pick only a few sentences and merge them with translated sentences found on the Duolingo application for learning a new language. Because of the origins of our DIY corpus, it consists mostly of basic everyday sentences.

Total number of data is visualized on the table 1.

[2] www.riksrevisjonen.no
[3] www.nav.no
[4] www.nyinorge.no
[5] www.skatteetaten.no
[6] http://www.opensubtitles.org/.

3.1 Tokenizer

According to a paper from author Jonathan J. Webster [WK92], tokenization is a process of separating a text into smaller units called tokens. These tokens can be whole words, subwords (n-grams), or only single letters. The tokenizer creates the vocabulary of these tokens based on the corpus it has been given. It is possible to specify the size of the vocabulary and therefore it is up to the user to choose the number of tokens.

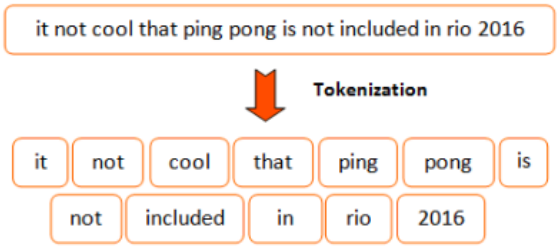


Figure 2: Example of tokenization [MOR].

We trained our tokenizer on the combined training corpus and tried several different sizes of token vocabulary. In the end, we decided to use a small vocabulary of the size of 2000. We decided to use a smaller vocabulary due to the fact that we focused on a smaller corpus and smaller models respectively. We also tried the sizes of 5000, 10000, and 20000 but this experiment did not bring any significant improvements.

4 Experiments

As a baseline for our neural network model we used a simple Transformer architecture with a various number of Encoder and Decoder layers and then we conducted few experiments in sake of improvement of the models performance.

The final model, that we decided to use as our baseline, had 4 Encoder and Decoder layers, used embedding dimension as well as the hidden size of 300, and had 3 attention heads. We set the vocabulary size to 2000 according to conducted experiments mentioned at the end of Tokenizer section 3.1. The results for the baseline model are further described in the Results section 5.1.

Dataset	Train	Valid	Avg. no len.	Avg. en len.
Government	50,000	2,500	13.85	16.81
Subtitles	250,000	2,500	6.16	6.76
Book	-	2,500	13.53	13.83
Custom	-	121	6.39	6.83

Table 1: Statistics of used data for NMT.

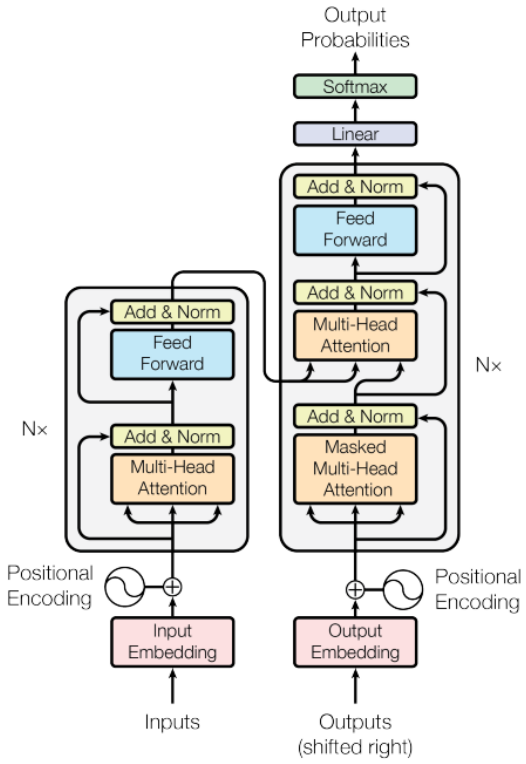


Figure 3: The Transformer - model architecture showed in "Attention is all you need" paper [VSP+17]

4.1 NorBERT2 as Encoder

The first adjustment that we tried was to replace the original Encoder of the Transformer with the pre-trained NorBERT2 [KBV+21] model. Our motivation for this was that even though NorBERT2 is quite a big model for our small dataset, it comes pre-trained on the Norwegian corpus and so if we use this model for creating our input sentence encoding, it could be easier for our Decoder to train and produce more precise translations. For time management purposes we froze the NorBERT2 model weights during training and tried to train only the Decoder side. Because of fixed word embedding layers in NorBERT2 model, we needed to use separate word embedding layers for the Decoder and Encoder which also resulted in an increased number of parameters. Also, one more thing we tried in terms of this experiment was to use hidden states of different layers of the NorBERT2 as the Encoder output. The motivation behind this was that BERT models in general learn different kinds of information in different layers and so the last layer of NorBERT2 does not necessarily need to be the best for the purpose of neural machine translation. Results of this experiment are further described in the Results section 5.2.

4.2 NorBERT2 embeddings

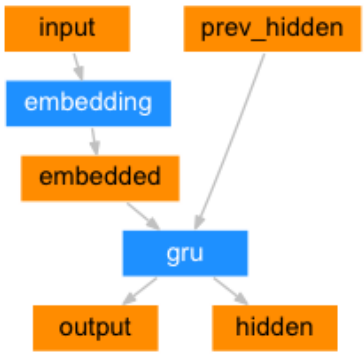
The motivation behind this experiment was that the previous experiment with the entire NorBERT2 model used as the Encoder did not bring more sufficient results than our baseline model. One of the reasons for it is that the architecture was very different on the Encoder and Decoder side.

Because of this, we decided to go back to baseline architecture. We tried to use pre-trained word embeddings of the NorBERT2 model and share them between the Encoder and Decoder. Because the NorBERT2 model was pre-trained only on the Norwegian corpus we decided to let it fine-tune on our corpus because during the translation process the shared word embeddings need to learn not only the features of the input language, but also the target language. This approach also meant to increase the size of model vocabulary to the size of NorBERT2 vocabulary. The results of this approach are described in the Results section 5.3.

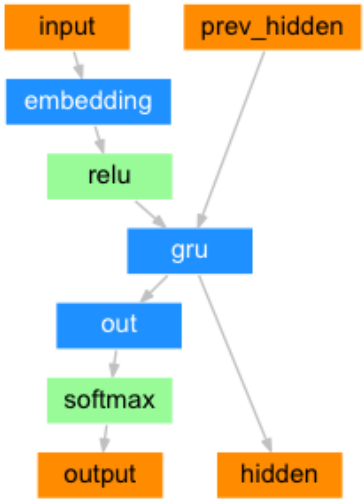
4.3 RNN

In this experiment of changing the architecture of the model to use RNN was provided. The main disadvantage of this approach is that as opposed to the Transformer where the input encoding can be used for each token independently, with RNNs whole input sentence must be encoded to only one vector. For the implementation, we followed the guideline from the official PyTorch site [PyT]. Neural machine translation with RNNs also uses the architecture of Encoder and Decoder. Visualization of the model is shown in figure 4. Results are

described in the Results section 5.4.



(a) Diagram of the RNN Encoder.



(b) Diagram of the RNN Encoder.

Figure 4: Architecture of the RNN Seq2seq model[PyT].

4.4 Data augmentation

In our research, we tried to briefly work not only with the model architecture but also with the data. During our prototyping process, we discover that our models do not work on sentences with all the capital letters. One of the most common ways how to deal with this problem is to transfer input sentences to all lower/upper case, but we decided to experiment and extend our training corpus of these all caps sentences. Results are further described in the results section 5.5.

5 Results

5.1 Baseline results

As the baseline, we trained the same Transformer model architecture on Government corpus, Subtitles corpus, and a combination of both corpora and we evaluated them on all of the validation sets we got earlier. The BLEU scores the models achieved are shown in the results figure 5.

Model parameters	training set score	Book valid. set score
6.7 M.	0.285	0.115
8.7 M.	0.299	0.117
10.7 M.	0.314	0.118

Table 2: Comparison of the model size and model performance (BLEU scores). All the models were trained on the combined corpus.

We trained mostly small models up to 11M parameters and so the generalization of the model translation skills was hard to achieve. This can be seen on the plot 5 and also table 3 where for example the models which were not trained on the Government corpus performed very poorly on the Government validation set. That is because the Government validation set has very specific topic and type of language, so our model is not able to generalize that well from the other training corpora. Second example would be the Book validation set where every model performed poorly for the same reasons as for the Government set.

We also tried to experiment with the size of the model and see how the performance change which is shown in result table 2.

From table 3 it is traced that the larger model did improve the performance on the training and also on the validation sets, but as opposed to the training set, on the validation set the difference between models is negligible.

5.2 NorBERT2 Encoder results

As mentioned in the section 4.1, we used the entire NorBERT2 model as the Encoder in our NMT model.

Please note that to save time we trained these models only on the Subtitles corpus which might affected the models results. Their final results are shown on the plot 5 and table 4.

From the table 4 it is visible that the model using the middle layer of the NorBERT2 achieved better results than the one using the last layer which indicates that the best information for the NMT is not stored/learned in the last NorBERT2 layer but somewhere in between.

Train. set	Book valid.	Govern. valid.	Subtitles valid.	DIY valid.
Government	0.02	0.33	0.03	0.06
Subtitles	0.10	0.08	0.25	0.44
Combined	0.12	0.35	0.25	0.48

Table 3: Different performance of the same model trained on different datasets. The results are in BLEU score units.

NorBERT2 out. layer	training set score	Book valid. set score
7	0.158	0.083
12	0.117	0.052

Table 4: Comparison of the models using NorBERT2 as an Encoder with different layer as the output. Both models were trained on the Subtitles corpus. Units are in BLEU.

We mentioned that this approach did not score very high on the evaluation board and the reasons for this, mentioned below, are mostly just speculation:

- Too big model for such small corpus (over 180M parameters).
- Too few epochs for training (30 epochs).
- The size of Encoder and Decoder were too much different.
- We did not allow fine-tuning of the Encoder.
- We did not share word embedding layers between Encoder and Decoder.
- We tried only 2 different layers of the NorBERT2 model as the output layer for our Encoder. There might be a better performing model.

5.3 NorBERT2 Embeddings results

As for our third experiment firstly mentioned in 4.2. We tried to use the same architecture as for the baseline model but use the pre-trained word embedding layer from NorBERT2, share it between Encoder and Decoder, and let it fine-tune.

We tried to train only one model on the Subtitles corpus which scored 0.09 BLEU on the Subtitles valid set and only 0.023 on the validation Book set. Results of this approach are also shown on the plot 5.

Again, the reasons why this approach did not perform well are up for discussion but we strongly believe that the main reason why this approach failed is because of the size of the embedding layer. NorBERT2 word embedding layer has a size of 50,000 tokens and dimensionality of 768. In this approach we went back to training smaller models and so classifying to over 50,000 classes was probably an overload for our model which had only around 24M parameters.

5.4 NMT using RNNs

As for our fourth experiment mentioned in section 4.3. We tried to replace the Transformer architecture with RNNs to see if by any chance the RNN would work better on small corpora. Unfortunately, this experiment failed when the RNN model was not able to translate almost anything. The training and validation process worked and the loss value during training gradually decreased. But we were not able to measure any BLEU score on this model. We think that one of the reason for lower performance is that Encoder must encode the whole sentence into the single vector which causes more compressed information for the Decoder.

Here are at least some examples of the translations using RNNs.

- **Target**: *What does it suggest?*
  Predicted: *You" s the you to the you?*
- **Target**: *Because I had suggested that he should come over.*
  Predicted: *I'm the a a a a a a a a a a a a a a a a a a.*
- **Target**: *There were several Mortimers, but only one who could be our visitor.*
  Predicted: *You" s the a a a a a a a a a a a a a a a a a a a.*

From the examples shown above, it seems that the translation using RNNs is not stuck only on one most frequent token that it would spam on repeat, but also it does not seems that the model picked out any grasp of the sentence meaning. We used a hidden size of 300 and so our assumption was that this performance was caused by the small encoding dimension size. Because of this assumption we also tried our Encoder to work bidirectionally but that did not help either. It also seems to understand that the target sentence is usually of a similar

length as the input sentence and what more, from the examples we see it seemed that the model even understands the end marks of the sentences like *'.'* or *'?'*. From this evaluation it seems that model was able to learn the very minimum but not enough to perform any kind of translation.

5.5 Data augmentation results

As mentioned in Data augmentation section 4.4, after our first generation of models, we found out that if we give our model the same sentence written in Bokmål but in all capital letters our model breaks. This is described more in the Common mistakes section 5.7. After reading through several articles on how to deal with this, we decided to try instead of transferring all the sentences to lowercase to extend the training corpus by sentences in all caps. Unfortunately, after training our model on such corpus we found out that it actually confused the model and its performance further decreased compared to the baseline models we got.

5.6 Evaluation on validation sets

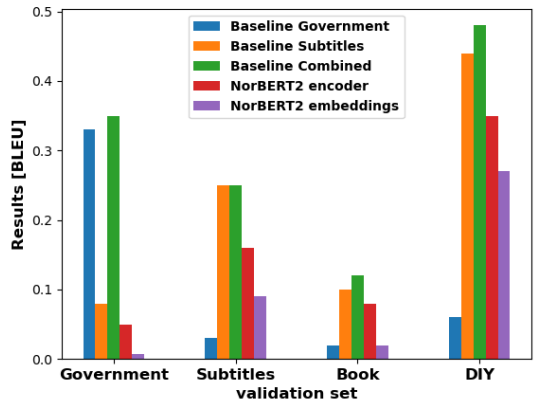


Figure 5: BLEU scores of different models on various validation sets. NorBERT2 models were trained only on the Subtitles corpus. Baseline models were trained accordingly to their names.

Poor generalization of models can be seen from the plot 5 where for example model *Baseline Subtitles* performed quite well on *Subtitle* and *DIY* validation sets it performed poorly on *Government* and *Book* validation sets. The best performing model in our research has been baseline Transformer trained on corpus combined from *Government* and *Subtitles* training corpora.

5.7 Mistakes on validation sets

Machine translation is quite a difficult task due to the linguistic differences between the target and source languages. The problems that one faces may be related to the ambiguity of the words, the existence of multi-words and compounds, some lexical and structural differences between languages, etc. [ABM+01].

There are also some classifications of translation errors, that have been analyzed by researchers. Unfortunately, the description of errors in translation from English to Norwegian or from Norwegian to English could not be found. However, we can rely on the proposed classification and analysis options for the English-Lithuanian [SKH17] and English-Spanish / Chinese-English [VXDN06].

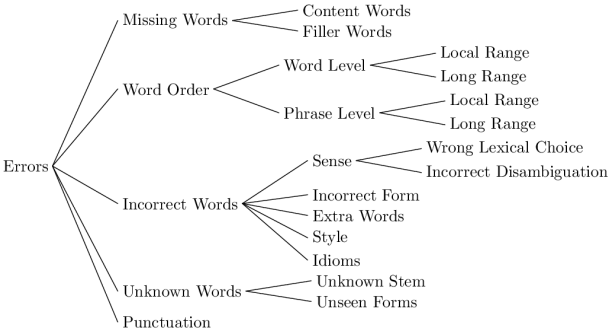


Figure 6: Classification of translation errors [VXDN06]

Based on the existing classifications (Figure 6, e.g.), we would like to describe what errors can be found in our research.

Hence, in this section, we will summarise the most common translation mistakes patterns of our models.

Errors that model provides in the long term:

1. No answer at all.

   Input: *Han er jo arvingen?*
   Target: *He being the heir?*
   Predicted:

2. Model does not know what to do if Norwegian sentence is given in capital letters

   Input: *hva heter du?*
   Predicted: *What's your name?*

   Input: *HVA HETER DU?*
   Predicted: *HETER DU?*

   Interestingly, if only the first letter in the sentence is capitalized, then the model can translate it, on the contrary, if the entire text is capitalized, then our model breaks. This might be

caused not only by the model itself but also by trained tokenizer.

3. Wrong punctuation

   Input: *Er han kommet vekk?*
   Target: *Has he **escaped**?*
   Predicted: *Is he **coming***

   In the current example it also can be seen how model, on the one hand, does the incorrect lexical choice for the translation, and on the other hand, it chooses the incorrect grammatical features such as tense.

The specified linguistic errors that the model provides:

1. Only part of the answer/grammatically incorrect segment.

   Input: *If I had only been there! He cried*
   Predicted: *I was just a*

2. Model can add extra words to translation.

   Input: *Andre kilder*
   Target: *Other sources*
   Predicted: *Other sources **are***

3. Model has troubles translating compound words

   Input: *For mer informasjon om **barnehage**, kontakt kommunen du bor i.*
   Target: *For further information about **pre-school** day care centres, contact your local municipality.*
   Predicted: *more information on the **school**, you contact the house.*

4. Model can predict non-existent word forms. (Close enough to word formation problem)

   Target: *I **haven't** saw him since...*
   Predicted: *I **seven't** said heaven!*

5. Errors in inflection

   Input: *Ifølge FBT utgjør **disse opplysningene** grunnlaget for videre oppføl- ging og befaring.*
   Target: *According to NODCS, **this information** constitutes the basis for further monitoring and site visits.*
   Predicted: *According to the information, **these information**, the reasons, the terms are beginning and the visit.*

   There are some mistakes in how words are inflected in a wrong way. As it can be seen, *'opplysningene'* is a plural form with the plural pronoun *disse*. In the current example to ways of solving the problem are seen: 1) use *'this information'* as a singular form, 2) use *'these informations'* as a plural form of group of information types. It is also can be connect to problems with syntactic government.

6. Model can choose referents in a wrong way in translated texts

   Input: *Med pistolen i hånden styrtet vi alle tre inn i værelset.*
   Target: *Pistol in hand, **we all three** rushed into the room.*
   Predicted: *the gun in my hand, the **three of the gun** was gone.*

7. Using word form in irrelevant collocation

   Input: *De har **ingen** mening om hvem denne L. L. kan være?*
   Target: *And you have **no idea** who L. L. is?*
   Predicted: *I have **any** idea what this is about,*

   'Any' is not the best choice here, as its use is caused by the presence of the word *ingen* and the collocation in English sentence is inappropriate.

8. Predictions include obscene language

   Target: *The **goods/services** must be for use within your enterprise.*
   Predicted: *That **shit** is for you in your life.*

   Target: *Boys, we're straight through to New York, so anyone needs a piss, you take it now.*
   Predicted: *We'll be a little baby, but I'll be a **little bitch**, it'll be a **little bitch**, it.*

All in all, as can be seen from the examples provided above, we were able to find most of the error types in our predictions. We have rather big amount of mistakes in morphological (word formation, inflection, non-existed word forms, etc.), lexical, and semantic (choice of sense, style as using appropriate lexical forms, etc.), and syntactic (government, etc.) fields [VXDN06]. There are also some interesting points that we luckily found in the predictions, and errors in translation compounds [SKH17] are the most important here.

6 Future directions

As the future direction of this project it make sense to work on handling the problem with case sensitivity. Capitalization seems to be a common issue in neural machine translation task [XHPY20] as the translation performance drops significantly when introducing case-sensitive evaluation metrics. Though there are some papers on casing methods for Neural Machine Translation [EU20], at the moment there is no established an optimal pre- and

post-processing methodology for machine translation.

7 Conclusion

The goal of this paper was to try to modify today's state of the art architecture for use on the small corpora. We tried older approach using RNNs as well as using components from pre-trained monolingual model NorBERT2. We also tried experiment with data augmentation and ran performance tests on several validation sets. Unfortunately, We have not been able to outperform our baseline Transformer model on neither of our validation sets. We also found the importance of choosing the target corpus for training as this parameter created the biggest gap in models performance. As for our *DIY* corpus we created this corpus with the goal of seeing if our models are usable for everyday translation which we are happy to confirm as it seems that the best model scored almost 0.5 BLEU which according to BLEU table[7] is considered high quality translations.

References

[ABM+01] Douglas Arnold, Lorna Balkan, Siety Meijer, R. Lee Humphreys, and Louisa Sadler. *Machine Translation: an Introductory Guide*. NCC Blackwell Ltd, 2001.

[EKCGK20] Ahmed El-Kishky, Vishrav Chaudhary, Francisco Guzmán, and Philipp Koehn. CCAligned: A massive collection of cross-lingual web-document pairs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*, pages 5960–5969, Online, November 2020. Association for Computational Linguistics.

[EU20] Thierry Etchegoyhen and Harritxu Gete Ugarte. To case or not to case: Evaluating casing methods for neural machine translation. *LREC*, page 3752–3760, 2020.

[KBV+21] Andrey Kutuzov, Jeremy Barnes, Erik Velldal, Lilja Øvrelid, and Stephan Oepen. Large-scale contextualised language modelling for Norwegian. In *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 30–40, Reykjavik, Iceland (Online), May 31–2 June 2021. Linköping University Electronic Press, Sweden.

[MOR] MORGAN FUNTOWICZ. Hugging Face Tutorials - Training Tokenizer. https://www.kaggle.com/code/funtowiczmo/hugging-face-tutorials-training-tokenizer/notebook. Online; accessed 19 May 2022.

[PyT] PyTorch. NLP FROM SCRATCH: TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION. https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html. Online; accessed 18 May 2022.

[SKD+17] Shashi Pal Singh, Ajai Kumar, Hemant Darbari, Lenali Singh, Anshika Rastogi, and Shikha Jain. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167, 2017.

[SKH17] Gilvilė Stankevičiūtė, Ramunė Kasperavičienė, and Jolita Horbačauskienė. Issues in machine translation. a case of mobile apps in the lithuanian and english language pair. *International Journal on Language Literature and Culture in Education 4(1)*, pages 75–88, 2017.

[Tho] Thomas Wood. What is a Transformer Neural Network? https://deepai.org/machine-learning-glossary-and-terms/transformer-neural-network#:~:text=The%20transformer%20is%20a%20component,area%20of%20natural%20language%20processing. Online; accessed 18 May 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[VXDN06] David Vilar, Jia Xu, Luis Fernando D'Haro, and Hermann Ney. Error analysis of statistical machine translation output. *LREC*, pages 697–702, 2006.

[WK92] Jonathan J Webster and Chunyu Kit. Tokenization as the initial phase in nlp. In *COLING 1992 Volume 4: The 14th International Conference on Computational Linguistics*, 1992.

[XHPY20] Shi X, Huang H, Jian P, and Tang YK. Case-sensitive neural machine translation. *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference*, page 662–674, 2020.

[7] BLEU table

Word Sense Induction for Norwegian with Contextualized Language Models

Thomas Zemp
University of Oslo
thomasrz@uio.no

Abstract

This paper investigates performance on Word Sense Induction tasks in Norwegian, using data from the recently published NorDiaChange data set. The paper looks at recent graph-based approaches used to generate WSI inventories for Norwegian and explores the possibility to improve on this performance by using contextualized embeddings (such as BERT). This paper finds that using contextualized embeddings can outperform more recent graph-based methods, but that these methods have difficulty predicting the ideal number of meanings.

1 Introduction

Polysemous words have long been a challenge in NLP. The resolution of these ambiguities is generally straightforward for humans: whether or not the word "bank" refers to a financial institution or a river bank will in most cases be clear from the word's context. However, this task is more challenging in NLP systems. In recent decades, performance on NLP tasks has been improved by the use of static embeddings, but these embedding models have a shortcoming in that they do not capture polysemy; for example, the embedding for bank is the same regardless of whether it refers to a financial institution or a river bank. Recent innovations with contextualized embeddings, like BERT, have helped with this problem by producing an embedding for a word based on the context. However, these embeddings do not actually determine the sense of the word, but rather vary based on the context of the word.

Using NLP to determine the appropriate sense of the word given its context can be divided into two related tasks: Word Sense Disambiguation (WSD) and Word Sense Induction (WSI). WSD involves assigning words to an appropriate sense given a priori knowledge about the possible senses of the word; in this way, if one has data with correct assignments, it is easy to build a supervised system to assign a word to its appropriate sense. WSI is the process of assigning a words to various senses without any prior knowledge of what the senses actually are. Thus, WSI can be thought of as an unsupervised task whereby a model must determine the appropriate clustering of sense. For example given the sentences "My bank offers good interest rates", "He's been working at Nordea, the bank, for two years" and "The bank was flooded after it rained", the model might determine that there are two meanings of the word bank based on the differences in the underlying contexts.

This paper begins with a description of the data evaluated in Section 2 and a description of the Adjusted Rand Index that will be used to assess WSI performance in Section 3. As a baseline this paper measures performance using graph-based approaches to WSI in Section 4; this section includes an overview of recent work, a discussion of method, and a presentation of results. This paper then begins to look at how contextualized embeddings may improve WSI performance by discussing existing approaches in Section 5; this section includes an overview of recent work, a discussion of method, and a presentation of results before moving on to a presentation of variations to this approach. Other approaches using contextualized embeddings for WSI are then considered, including using BERT hidden layers in Section 6 and average static embeddings based on candidate words generated with BERT in Section 7. A discussion of differentiating monosemic and polysemic words is presented in Section 8. Finally, the paper presents a conclusion and outlines possible ideas for further research in Section 9.

2 Data

This paper will explore WSI in Norwegian using the recently published NorDiaChange data set (Kutuzov et al., 2022). The NorDiaChange dataset was built to investigate semantic change over time

in Norwegian, and consists of two subsets each with 80 lemmatized nouns. This paper uses subset 1 which looks at semantic differences between 1929-1965 and 1970-2013. NorDiaChange used native Norwegian speakers who evaluated pairs of sentences and determined if the chosen word was the same or different (scale from 0 to 4). From these judgements, a clustering of the sentences into possible senses was determined. It is these senses that are used for evaluating the efficacy of the WSI. Note that the NorDiaChange includes examples in both standard written forms of Norwegian-*bokmål* and *nynorsk*–but *bokmål* is predominant.

The NorDiaChange dataset was slightly cleaned up for use in this paper to resolve some cases where the provided offset for the target lemma was incorrect, but is otherwise not changed. In some cases, the provided offsets for the token appeared to be slightly shifted. For example, for the context with identifier 1970-2015_kjemi_642, the provided offsets would give the word " kjem" as opposed to the desired "*kjemi*". In these cases, the offsets were corrected to align properly. In other cases, the offsets appear to reference an incorrect lemma. For example, for the context with identifier 1929-1965_egg_473, the provided offsets identify the word "*er*" (am) as opposed to the desired lemma of "*egg*". In these cases, the first word containing the expected lemma was used for the offsets. This may result in some misidentifications; for instance, if the sentence contained multiple meaning of the word "*egg*" or if the identified word had the lemma as a substring without being related, e.g. "*legg*". However, this method seemed preferable and more reliable than using evidently incorrect offsets, and sample review of the cleaned offsets did not show any of these potential problems. Context sentences with the adjusted offsets are available here: https://github.uio.no/thomasrz/trz-in5550/blob/master/exam/all_data.csv

3 Evaluation metric

To evaluate the effectiveness of WSI performance, this paper uses the Adjusted Rand Index (ARI). ARI is a chance-corrected version of the Rand Index:

$$RandIndex = \frac{AgreeingPairs}{AgreeingPairs + DisagreeingPairs}$$

An ARI score does not consider the actual labels themselves but rather the clustering. For instance comparing the predicted labels [1,1,2] and the target labels [a,a,f] will produce an ARI of 100% because the underlying clustering is the same between the predictions and the targets. ARIs have a maximum of 100% (perfect alignment) and can be negative if the clustering is worse than that expected by random chance. In general, this paper will use the macro-average ARI across the 40 lemmas as a benchmark.

4 Graph based methods (egvi)

4.1 Related work

Some recent work has explored graph-based approaches to handle WSI. A recent study (Logacheva et al., 2020) created an algorithm referred to as Ego-Graph Vector Induction (egvi). This method used static embeddings to determine a given word's nearest neighbours and then used vector subtraction to identify so-called "anti-edges", a graph between near neighbours was then constructed which excludes connections between unrelated neighbours as determined by the anti-edges (for example, the word *ruby* is related to *python* and *opal* but *python* and *opal* are not related to one another). Using these constructed graphs, clusters were determined using the Chinese Whispers Algorithm, and these clusters were determined to be senses for the given word.

4.2 Method

Sense dictionaries for 158 languages, including Norwegian, based on the egvi algorithm have been made public. The sense dictionary represents each sense with a separate line listing the nearest neighbour words of that word. For example, the public sense dictionary[1] shows that egvi determined that the following six senses exist for the word "*ris*":

- *perlehirse ukokt maisgrøt hirse hvetekli hvetegrøt bokhvete 85kle hirsegrøt riskake sorghum avkokt ros hvete kokoskake* [pearl millet, uncooked, polenta, millet, bran, wheat porridge, buckwheat, 85kle, millet porridge, rice cake, sorghum, boiled, praise, wheat, coconut cake]
- *makisushi mangochutney søtpoteter sesampasta blomkålris currypasta sesamfrø mangosalat jasminris sesamolje chili risnudler*

[1] egvi sense inventories calculated from the nearest 50, 100, and 200 neighbours are available. This paper uses the senses derived from the nearest 50 words. The paper uses the public sense as a baseline and so did not fully examine performance variations for each publicly available senses dictionary.

woksaus chiliolje mangosalsa [maki sushi, mango chutney, sweet potatoes, sesame paste, cauliflower rice, curry paste, sesame seeds, mango salad, jasmine rice, sesame oil, chili, rice noodles, wok sauce, chili oil, mango salsa]

- *naanbrød nanbrød Nanbrød maisbolle maiskorn bananbrød maisbrød* [naan bread, nan bread, Nan bread, corn fritter, corn kernel, banana bread, cornbread]
- *soyabønne soyabønner mungbønner soyamel byggryn soyaost* [soybean, soybeans, mung beans, soybean meal, barley, soy cheese]
- *Risen* [The rice / Risen]
- *grønnsaker* [vegetables]

As a baseline, this paper tried to perform WSI using these predetermined senses. Following the method of the original paper, cluster centroids were determined by taking the average static embedding for all the neighbour words[2]. For each sentence, the average static embedding was then calculated (excluding the target word). To determine the appropriate sense, the example was assigned to the sense clusteroid nearest to its average sentence-level vector (based on cosine similarity). For example, for the sentence "*Av korn bidrar hvete og ris hver med en femtedel av verdenskonsumet.*" ("Wheat and rice each account for a fifth of worldwide grain consumption"), the sense is mapped to the first from the sense dictionary, which contains the word "*hvete*" (wheat) and similar terms like "*hirse*" (millet) and "*sorghum*" (sorghum).

4.3 Results

Performing WSI using the predefined sense dictionaries on the NorDiaChange Subset1 lemmas resulted in an ARI of 18.2% (note: "*anfektelse*" was excluded because it did not appear in the static embeddings). This seems like a reasonable result, but it should be noted that as this a macro-average, most of the positive result comes from cases where

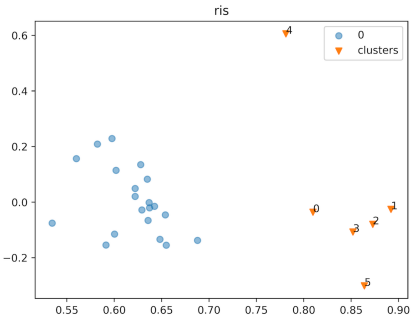


Figure 1: NorDiaChange sentences (dots representing the average sentence embedding) are assigned to the nearest sense. Here average egvi senses are depicted as upside-down triangles. One can see that the average context embedding varies significantly from the available sense embeddings.

the baseline method (correctly) assigned all instances to one class (and thus achieved an ARI of 100% for the class). Since for Subset1 of NorDiaChange, 7 of 40 lemmas have only one class, simply predicting one class for each word would actually produce a macro-average ARI of 35.0% across all 40 lemmas.

It is evident that this approach has certain weaknesses. For example, the egvi senses of the word "*ris*" are all food-related. In reality, the noun "*ris*" has several meanings (loosely translated): rice, ream (of paper), twigs, beating, criticism. The first egvi sense includes word "*ros*" from the expression "*ris og ros*" (praise and criticism) but it is subsumed in a list of primarily grain-related words. The egvi method also produces "*Risen*" as a separate meaning, which is not particularly useful as this could either be the definite form "the rice" or a geographic name (a web search reveals that Risen is a name of some insignificant small hills and islands in Norway).

These limitations become evident when we look at graphic representations of the determinations according to the egvi algorithm, for example in Figure 1. [3]

5 Tf-idf Vectors with contextualized language model candidates

5.1 Related work

Other recent approaches to WSI have tried to leverage the power of contextualized embeddings (like

[2]static embedding model 100 from http://vectors.nlpl.eu/repository/ is used for static embeddings throughout this paper. This model was selected due to its large vocabulary (almost 4.5 million words) and it is non-lemmatized and hence easily applied to NorDiaChange's context sentences without further data processing. Other static embedding models were not tested in this paper, though this could be an area for further research.

[3]Note: this paper uses Singular Value Decomposition for transforming multidimensional vectors to two-dimensional representation

BERT and ELMo). In two recent papers, Asaf Amrami and Yoav Goldberg have explored the approach of using contextualized language models to generate candidate words (that is possible substitutes for the target word), vextorizing the result candidate documents, and then assigning the target to senses based on the clustering of those candidate vectors. (Amrami and Goldberg, 2018) (Amrami and Goldberg, 2019)

For example, one can mask the target lemma in sample sentences:

- My [MASK] offers good interest rates
- He's been working at Nordea, the [MASK], for two years
- The [MASK] was flooded after it rained

then use BERT to generate candidate words:

- [account, credit card, bank, loan, mortgage]
- [bank, company, DNB, insurer, Sparebanken]
- [river, lake, house, town, dam]

these words can then be vextorized (for example using scikit-learn's TfidfVectorizer):

- [0.42 0.32 0.42 0.00 0.42 0.00 0.00 0.00 0.00 0.00 0.42 0.42 0.00 0.00 0.00]
- [0.00 0.36 0.00 0.47 0.00 0.00 0.47 0.00 0.47 0.00 0.00 0.00 0.00 0.47 0.00]
- [0.00 0.00 0.00 0.00 0.00 0.45 0.00 0.45 0.00 0.45 0.00 0.00 0.45 0.00 0.45]

Finally, the vextorized representations can be clustered using an appropriate clustering mechanism (for example K-Means, Agglomerative Clustering, Affinity Propagation, etc.,).

In Amrami and Goldberg's original paper, 20 candidate words were generated for each masked word, and the authors chose to assign to 7 clusters (fixed based on observation of underlying data), with ELMo used as the embedding model (Amrami and Goldberg, 2018); further work, however, showed BERT to achieve better results. (Amrami and Goldberg, 2019). There is good reason to believe that BERT may be particularly suited for an approach based on clustering candidate words. The BERT model is based on a transformer and is trained by masking and then predicting given words (Devlin et al., 2018), thus producing context-dependent embeddings. Hence, one might expect that the embeddings for masked words, and their predicted lexical substitutes, might differ based on the word's sense in its given context.

5.2 Method

This paper applied the method described in Amrami and Goldberg's work, using 20 candidate words, 3 fixed clusters (with clusters determined using agglomerative clustering), and NorBERT2 to perform WSI on NorDiaChange dataset. 3 clusters were chosen rather than 7 as this fit the underlying data better.

5.3 Results

Overall, this achieved a macro-average ARI of 8.3%. This is clearly lower than the egvi method's 18.2%, but the macro-average is a fairly misleading statistic. As mentioned above, the higher score for the egvi method is largely associated with assigning lemmas to only one cluster and achieving 100%. If one looks at the results on the individual lemma level, one can see that ARI actually improves for most lemmas where there is more than one class (see Figure 2). Since this method enforce 3 fixed clusters, all of the monosemic words that previously obtained an ARI of 100%, now obtain a score of 0%.

One can also see this improvement by looking at results of individual lemmas, for example in Figure 3. Here we see clear evidence of clustering for the word "*ris*". Some of the clusters appear intuitive (e.g. the grouping of grain related words–"*mais* (corn)", "*erter* (peas)", "*korn* (grain)"–in the upper left), but some seem less intuitive, e.g. the grouping of points with "*speilet* (the mirror)" as top candidate word in the lower right. In this latter case, the candidate word "*speilet*" is mostly likely predicted because of the presence of the expression "*riset bak speilet*" (literally, "the beating behind the mirror", or a "backup threat"). This does generate a clear clustering difference of the grain/threat meaning for "*ris*", but shows that the challenges of language model predictions with idioms.

Variations to tf-idf approach

A number of variations to this algorithm were investigated. For consistency, the baseline of 20 candidate words and 3 fixed clusters appears at the top in each comparison.

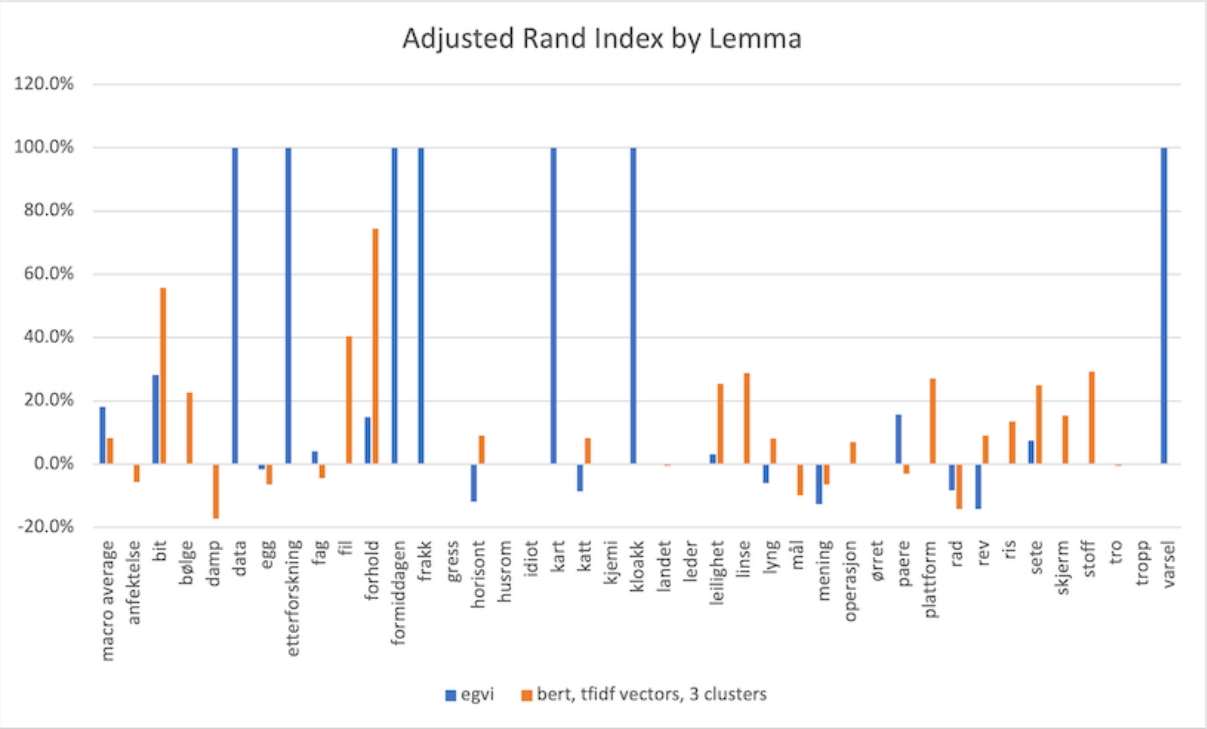


Figure 2: ARI per lemma comparing egvi method and method using tf-idf vector representations of candidate words generated with NorBERT2 (20 candidate words, 3 fixed clusters). For words with more than one sense, ARI score generally is improved using tf-idf approach.

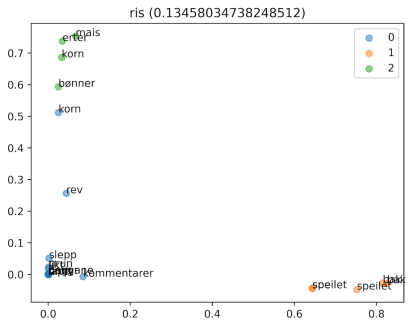


Figure 3: Candidate tf-idf representations are projected into two-dimensional space. They are then clustered into 3 clusters using agglomerative clustering. Annotations at each point indicate the top candidate for each example.

5.4 Number of clusters

A possible variation of this approach is to vary the number of clusters (either different numbers of fixed clusters, or using a non-fixed number of clusters and choosing an appropriate algorithm to decide the appropriate number of clusters). Throughout this paper, unless otherwise specified, agglomerative clustering is used when number of clusters is fixed and affinity propagation is used when clusters are non-fixed. In this case, using a non-fixed number of clusters seems to have improved results notably.

Number of Clusters	ARI
3	8.3%
5	7.8%
7	7.6%
10	7.0%
Non-fixed	10.9%

5.5 Number of candidates

One can also vary the number of candidate words (N) generated by the BERT model. This paper investigated choices of N=10,20, and 50. Best results were obtained using 50 candidate words. This may be because the resultant documents of N candidate words had better overlap with higher values of N and hence the resulting tf-idf vectors were easier to cluster.

Number of Candidates	ARI
20	8.3%
10	6.1%
50	11.5%

5.6 Extra text

Amrami and Goldberg suggested that adding additional text to the model could improve results. For example instead of masking an example sentence of "The bank was flooded after it rained" as "The [MASK] was flooded after it rained", it could be masked with filler words to help guide the candidate predictions. An example would be to add the word and: "The bank and [MASK] was flooded after it rained". This paper tested variations on this approach using *og* (and), *eller* (or) and *eller til og med* (or even). The results here were mixed; *og* and *eller* lead to slightly worse results whereas *eller til og med* improves the macro-average from 8.3% to 9.7%. Due to the mixed performance here, it is difficult to conclude much on the usefulness of adding coordination clauses. These clauses, or indeed other sentence manipulation, might be useful in specific tasks, but care would be needed in selecting them as they could lead to either improved or deteriorated performance.

Extra text	ARI
no extra text	8.3%
og	7.8%
eller	7.6%
eller til og med	9.7%

5.7 BERT Model

Varying the underlying BERT model might also influence performance. NorBERT-2 performed significantly better than NorBERT-1 which might not be unexpected given that NorBERT-2 has better vocabulary coverage.

Extra text	ARI
NorBERT-2	8.3%
NorBERT-1	6.6%

6 BERT Vectors

Another option is to perform clustering directly based on BERT's vector representations of the masked word rather than going through the intermediary step of generating candidate words.

In this approach, one can take the final hidden state from the BERT model and cluster with these vectors. This paper investigated this approach with both a non-fixed number of clusters (using affinity propagation) and fixed clusters of size 3 and 7 (using agglomerative clustering). NorBERT-2 is the BERT model used throughout these experiments.

ARI for NorBERT-2 Vectors

Number of Clusters	ARI
3	21.1%
7	13.9%
Non-fixed	18.2%

We now begin to get results better than the egvi baseline. This might be due to the NorBERT-2 vectors containing richer information than the candidate words themselves.

When we look at individual words we can see some evidence that the clusters may be better defined (see Figure 4)

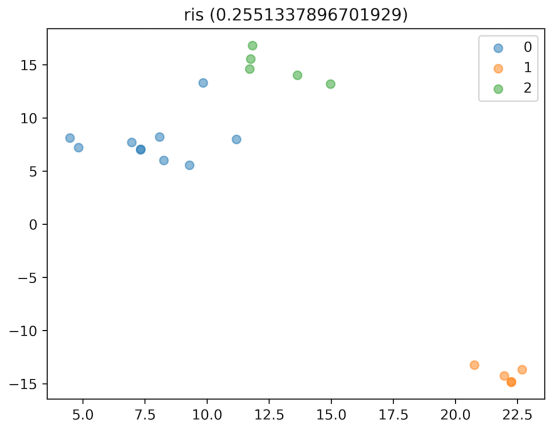


Figure 4: Two-dimensional projection of NorBERT-2 Vectors for individual sentences and assigned clusters using affinity clustering. Points here could be annotated with the original context sentence, but this has been omitted to improve graph clarity/size.

6.1 Hidden layers

Different hidden layers of BERT Models can be thought of as modelling different linguistic phenomena (Belinkov and Glass, 2018). Using this insight, it was investigated whether a specific level of NorBERT-2's hidden layers was particularly suited for the task of WSI.

Interestingly, the macro-average ARI increases as one uses higher layers of the NorBERT-2 model, particularly notably from layer 8 onwards. This may be due to BERT's higher layers learning more about the context and this richer information being more useful for the WSI task (see Figure 5).

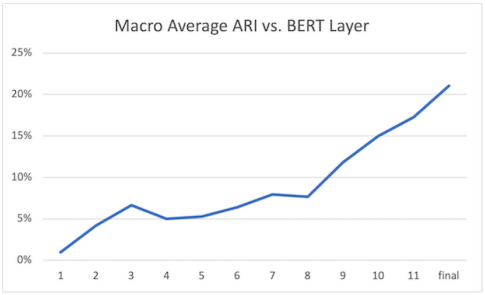


Figure 5: ARI vs hidden-layer level of NorBERT2 model used for creating vectors.

7 Average static embedding vectors based on BERT candidate words

Another approach to clustering that this paper looked at was using candidate words generated by BERT but then vextorizing them with the use of static embedding vector model. The theory behind this approach is that one might get more defined clusters by using static embedding vectors because the tf-idf vextorization process will not be able to capture the relatedness of candidate words (that is, it does not intrinsically know that *Sparebanken* and *DNB* are related to one another and to the concept of bank as financial institution and must rely on their appearing within similar lists of candidate words to aid clustering).

In this approach, candidate words were thus generated and then the average static embedding vector for each of these list of candidate words was generated. These average vectors were then clustered (with affinity propagation for a non-fixed number of clusters and agglomerative clustering for a fixed number of clusters). NorBERT-2 was used as the BERT model and NLPL Model 100 was used for static embeddings.

This approach produced results that were quite similar to the approach of directly using NorBERT-2 vectors.

ARI for candidate average embedding vectors

Number of Clusters	ARI
3	21.1%
7	13.7%
Non-fixed	17.8%

8 Determining a clustering threshold

While the methods of directly using NorBERT-2 vectors and using an average static embedding based on NorBERT-2 candidate words both yield decent results, they perform better with a fixed cluster number of 3, rather than when the affinity propagation algorithm is allowed to choose the number of vectors. This is problematic as the results rely on a somewhat arbitrary choice of clusters which might not be appropriate for a given word; indeed words vary in the number of sense they have.

Secondly, the approach of choosing a fixed number of clusters has the disadvantage of imposing clusters when there is no evidence for clustering, for example when there is only one sense. This is a large handicap when calculating the macro-average ARI as seen in the discussion of the egvi approach. The egvi algorithm with the predefined senses did not do a particularly good job overall but the macro-average ARI was boosted by a number of lemmas where egvi correctly predicted one sense (regardless of whether that sense was itself reasonable).

It might be therefore be possible to improve performance by coming up with a reasonable heuristic to predict number of clusters and then to perform clustering. Concretely, having a heuristic that decides whether a lemma has 1 or several senses would most help.

To approach this, it is useful to investigate various clustering statistics and see if there are any clear trends. This was explored using the average static embedding of candidates approach (chosen in favour of BERT vectors as the average of static embeddings allows one to annotate points with top candidate word).

One approach investigated was the construction of elbow graphs showing decline in inertia (determined with K-means clustering) for each lemma. Ideally, elbow graphs for lemmas with multiples sense would show a sharper initial decline as the number of clusters is increased, and then even off at the ideal cluster. For lemmas with one sense, the decline should be more even as K increases.

There was some evidence of this when examining graphs. For instance, the word "*stoff*" (textile, material, curriculum, etc.) had 3 predicted senses from NorDiaChange; its scatter plot (Figure 6) shows clear signs of clustering and its elbow graph (Figure 7) shows a steep decline from 1 to 2 clusters.

Meanwhile, *etterforskning* (investigation) had 1 sense from NorDiaChange and shows neither clear evidence of clustering in its scatter plot (Figure 8)

nor an initial drop in its elbow graph (Figure 9)

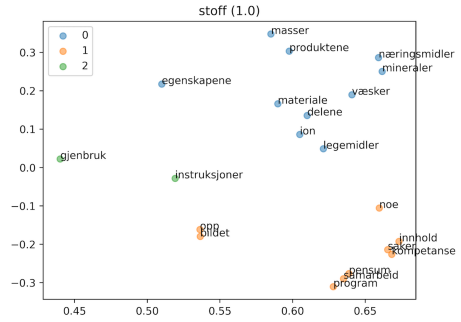


Figure 6: Stoff. Average embedding vectors for candidate words for each sample sentence. Points are annotated with the top candidate word.

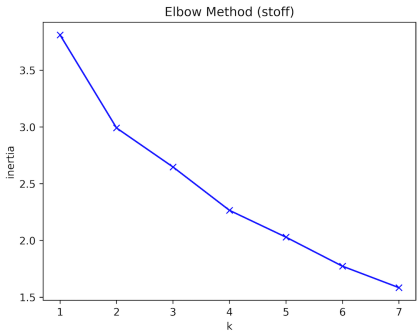


Figure 7: Stoff. Elbow graph showing inertia vs. number of clusters with clustering performed using K-means algorithm.

Unfortunately, there was significant variance across lemmas, and this pattern was not as clear cut across words. This made the application of some general heuristic fairly difficult. Based on examination of data, it was decided to try with a threshold of .83 from 1 to 2 clusters (that is any lemma for which the ratio of inertia with 2 clusters to inertia with 1 clusters exceeded .83 was deemed to have only 1 sense). This resulted in a higher overall macro-average ARI of 31.8% (see Table 1 for details by lemma).

9 Conclusion and potential further research

Overall, the approach of using BERT embeddings in WSI tasks shows some promise relative to graph-based methods, but is hampered particularly by an inability to correctly predict the appropriate number of clusters. Introducing an inertia threshold to distinguish monosemic words from polysemic

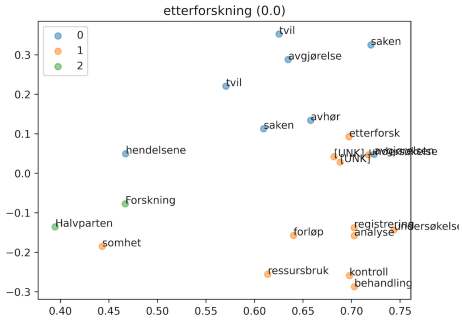


Figure 8: Etterforskning. Average embedding vectors for candidate words for each sample sentence. Points are annotated with the top candidate word.

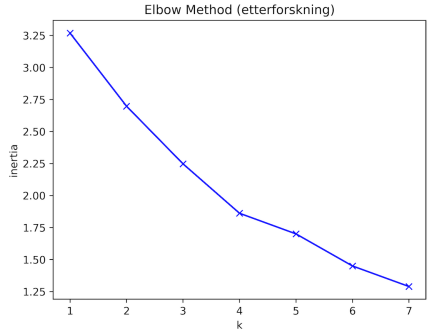


Figure 9: Etterforksning. Elbow graph showing inertia vs. number of clusters with clustering performed using K-means algorithm.

words helped improve overall macro-average ARI scores for this task, but the determination of such a threshold was not clear cut.

Further research might explore other opportunities for more accurately predicting the split in clusters. For instance, by looking at additional clustering statistics to see if some provide clearer information on the ideal number of clusters for a given word.

Another method might be to try to refine one of the methods using candidate words but initially apply some clustering of candidate words (for instance using cosine similarity based on static embeddings) to try to filter out less relevant candidate words and therefore make the candidate prediction "purer" and easier to cluster.

One might also try to determine potential word senses on a larger corpus and then make predictions based on a smaller dataset, like NorDiaChange. It is theoretically unclear, however, whether a larger corpus would help reduce or amplify the noise that makes word sense induction challenging.

Table 1: ARI with and without clustering threshold

lemma	no threshold	threshold
macro-average	21.1%	31.8%
anfektelse	14.6%	14.6%
bit	49.7%	49.7%
bølge	42.6%	42.6%
damp	33.8%	33.8%
data	0.0%	100.0%
egg	5.1%	5.1%
etterforskning	0.0%	0.0%
fag	11.9%	11.9%
fil	31.2%	31.2%
forhold	43.2%	43.2%
formiddagen	0.0%	0.0%
frakk	0.0%	0.0%
gress	0.0%	0.0%
horisont	43.4%	0.0%
husrom	0.0%	0.0%
idiot	0.0%	0.0%
kart	0.0%	100.0%
katt	39.5%	39.5%
kjemi	0.0%	100.0%
kloakk	0.0%	0.0%
landet	3.6%	3.6%
leder	0.0%	0.0%
leilighet	14.7%	14.7%
linse	29.4%	0.0%
lyng	82.4%	82.4%
mål	19.3%	19.3%
mening	45.1%	45.1%
operasjon	17.5%	17.5%
ørret	0.0%	0.0%
paere	60.1%	60.1%
plattform	17.4%	17.4%
rad	0.8%	0.8%
rev	32.8%	32.8%
ris	25.5%	25.5%
sete	45.2%	45.2%
skjerm	36.0%	36.0%
stoff	100.0%	100.0%
tro	0.2%	0.2%
tropp	0.0%	100.0%
varsel	0.0%	100.0%

Acknowledgements

The author would like to thank the teaching staff of the University of Oslo's IN5550: Neural Methods in Natural Language Processing class, and particularly Andrey Kutuzov for his guidance.

References

Asaf Amrami and Yoav Goldberg. 2018. Word sense induction with neural biLM and symmetric patterns. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4860–4867, Brussels, Belgium. Association for Computational Linguistics.

Asaf Amrami and Yoav Goldberg. 2019. Towards better substitution-based word sense induction.

Yonatan Belinkov and James Glass. 2018. Analysis methods in neural language processing: A survey.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Andrey Kutuzov, Samia Touileb, Petter Mæhlum, Tita Ranveig Enstad, and Alexandra Wittemann. 2022. Nordiachange: Diachronic semantic change dataset for norwegian.

Varvara Logacheva, Denis Teslenko, Artem Shelmanov, Steffen Remus, Dmitry Ustalov, Andrey Kutuzov, Ekaterina Artemova, Chris Biemann, Simone Paolo Ponzetto, and Alexander Panchenko. 2020. Word sense disambiguation for 158 languages using word embeddings only. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5943–5952, Marseille, France. European Language Resources Association.

Combining Transformer Based BERT-Models with Character-Level Convolutional Networks for Targeted Sentiment Analysis for Norwegian

Herman Brunborg
University of Oslo
hermabr@uio.no

Abstract

This article offers an empirical study of combining character-level convolutional neural networks (CNN) with a transformer based Bidirectional Encoder Representations from Transformers (BERT) for doing targeted sentiment analysis for the Norwegian language. Among the finding is the fact that a character-based CNN embedder together with the transformer models seems to improve the models as compared to a pure BERT embedder. The best performing model is found to obtain a proportional F_1 score of 54.41% and a binary F_1 score of 66.76%. This was obtained by combining the NorBERT 2 model with an uncased character-based CNN embedder.

1 Introduction

Unlike standard sentiment analysis, where the goal usually is to predict the polarity of a text on a higher level, for instance on document level, the goal in targeted sentiment analysis is to predict the sentiment on a per-word basis.

This article explores using different machine learning methods to predict the sentiment for the $NoReC_{fine}$ dataset from Øvrelid et al. (2020). Three different main embedders are tried out. One is a simple bidirectional Long short-term memory (LSTM) model built on top of a simple Word2Vec embedder. The two others are the newly released NorBERT 2 model from Kutuzov et al. (2021), while the other is a multilingual BERT model (mBERT) model trained on 104 different languages.

The main portion of this article focuses on the BERT models as described in Devlin et al. (2018). BERT is a pre-trained deep bidirectional representation from unlabeled texts by jointly conditioning on both left and right context in all layers. In this article, two modified BERT models were used, namely the Norwegian model NorBERT 2 and the multilingual model Multilingual BERT.

We explore if adding a character-based embedder in addition to the big transformer based embedders can improve performance even more, inspired by the promising results from Zhang et al. (2015), where they use a pure character-level CNN approach for text classification. The novelty of this paper, is that we now combine character-level embeddings with a transformers-based model, which empirically has shown to perform quite well (Devlin et al., 2018).

In the section 2, the methods used in the article are explored, followed by section 3, where the results of the investigation are presented. The articl continues with a discussion about the most interesting findings in section 4 followed by a conclusion in section 5.

2 Methods

2.1 Dataset

The dataset that was used in this project comes from the Norwegian Review Corpus (NoReC), a dataset introduced by Øvrelid et al. (2020). This is a fine-grained sentiment analysis dataset for Norwegians, annotated with polar expressions, targets and holders of opinions. It is made up of texts taken from news sources on a wide variety of domains, including literature, video games and music. The labels consist of of a BIO-tag and a polarity (positive or negative), which make a total of 5 labels (B-targ-Positive, I-targ-Positive, B-targ-Negative, -targ-Negative, O).

2.2 Evaluation

To evaluate the model, two different F_1 metrics will be used: The proportional F_1 and the Binary F_1. The Binary F_1 counts any overlaps in the predicted and true labels, while the Proportional F_1 reduces to token-level F_1.

	Sentences	Targets
Train	8634	5044
Dev	1531	877
Test	1272	735
Total	11437	6656

Table 1: The table shows the number of sentences and targets across the different data splits. It should be noted that the NoReC dataset includes more labels as described here, where only targets are used, and not holders or polar expressions.

2.3 Models

One of the most fundamental choices when doing machine learning is selecting the architecture. In this article, one simple baseline model was used, and two transformer-based BERT-based models.

2.3.1 Baseline

A simple baseline model was trained to have something to evaluate the much bigger and more computationally heavy pre-trained models against. The baseline model is a bidirectional Long short-term memory (LSTM) model, with two LSTM-layers and one linear layer. This was built on top of a frozen, pre-trained Word2Vec embedder[1].

2.4 Character-Level CNN

An alternative to doing text classification is using a purely character-level model, as suggested by (Zhang et al., 2015). The advantage of this method is that the embedder can be smaller, since only a few characters are needed as opposed to thousands of words. This means that it might be lesss prone to misspelling (Zhang et al., 2015). Empirical data suggests that it can have some advantages compared to a word-level model, for instance it might be less prone to typos.

In this article, it was used besides the BERT embedder, where the output from the word embedder and from a simple CNN being concatenated and fed into a classifier as described in 2.5.1. A small change was made however: the input size had to be increased to accept the output from both the word embedder and the character CNN.

The non-space characters used as the vocabulary for the CNN are:

abcdefghijklmnopqrstuvwxyzABCDEFGH
IJKLMNOPQRSTUVWXYZ0123456789!
æøåÆØÅ"#$%\&'()*+,- ./:;<=>?@[]‿‘| «»

additionally, all out-of-vocabulary letters were encoded to the same custom token. This token was also included to the vocabulary.

The character encoding was done by prescribing a number corresponding to the index of the letter in the alphabet, and then converting the character to its number (a "one-hot" encoding). Each word was then converted into a vector where each character encoded. Each of these vectors were then placed in a Longest word $\times$ Number of words-matrix.

This was then fed through the CNN, resulting in matrix which was then concatenated with the output from the embedder.

The CNN used was made up by a 1D convolution and a linear layer.

2.4.1 Choice of Alphabet

The alphabet used is very similar to the one used in Zhang et al. (2015), with some more characters added, because of differences between Norwegian and English, like the addition of the Norwegian letters æøå and other quote signs. There were also distinctions made between including both capitalized and lower cased letters (cased) and lowering all letters (uncased).

2.5 BERT

Two different transformer-based BERT models were utilized in this article: NorBERT 2 and mBERT.

The model NorBERT 2, a recently released improved version of the NorBERT model as described in Kutuzov et al. (2021), trained purely on Norwegian data was used as one of the models.

The second BERT model utilized in this article is mBERT. This is a model trained on 104 different languages. Empirical tests between mBERT and NorBERT have already been performed, with NorBERT generally performing quite substantially better (Kutuzov et al., 2021), but the novelty in the present study comes from the fact that we also include a character-level embedder as described in 2.4.

The optimizer used to train the big network was AdamW (Loshchilov and Hutter, 2017), a modification of the Adam optimizer (Kingma and Ba, 2014), which includes improvements regarding decoupled weight decay regularization, which is very helpful for training as big model like BERT.

[1]The embedder is available at http://vectors.nlpl.eu/repository/20/58.zip

2.5.1 Classification Layer

To include both the output from the transformer embedder and from the character-level CNN, a custom classification layer was trained with the concatenated output from the transformer-based embedder and the output from the character-level CNN. For the architecture of the classification layer, both a recurrent- and a feed-forward neural network were tested.

2.6 Ensemble Models

BERT models are known to be unstable, with different seeds resulting in potentially vastly different performance (Mosbach et al., 2020). One way to increasing the likelihood of a stable model, is training an ensemble of classifiers on different seeds. In this paper, the ensemble methods were used after the optimal hyperparameters, to find the performance for the test set. This was done by training models with the same parameters, but different seeds. Each model then predicted the character-level tokens, with the final output being the one that was predicted most often on a per-token basis.

3 Results

The table 2 shows the performance using different combinations of including the character-level CNN, changing up on the embedder and the architecture for the final classification layers. Table 3 shows the performance when the learning rate is varied and table 4 shows the importance of choosing the correct weight decay for the model.

The figures 1, 2, 3, 4, 5 and 6 show the proportional F_1 score, the binary F_1 score and the loss for different epochs for the NorBERT and mBERT embedders respectively.

The performance for the simple bi-LSTM model was 28.37% for the proportional F_1 score and 45.97% for the binary F_1 score, which means that it performs quite substantially worse than the transformer-based models. It must however be noted that these results were obtained without nearly as much time tweaking the network as was done for the BERT-models.

Char	Embedder	F. clas	Prop F_1	Bin F_1
No	NorBERT	RNN	0.5219	0.6699
No	NorBERT	FFNN	0.5203	0.6681
No	mBERT	RNN	0.4276	0.6076
No	mBERT	FFNN	0.4174	0.6060
Cased	NorBERT	RNN	0.5258	0.6632
Cased	NorBERT	FFNN	0.5249	0.6726
Cased	mBERT	RNN	0.4124	0.5879
Cased	mBERT	FFNN	0.4294	0.5939
Uncased	**NorBERT**	**RNN**	**0.5300**	0.6770
Uncased	NorBERT	FFNN	0.5248	**0.6774**
Uncased	mBERT	RNN	0.4187	0.5906
Uncased	mBERT	FFNN	0.4285	0.5968

Table 2: The table shows the best proportional and binary F_1 scores obtained for different combinations of including characters, different underlying embedders and different architectures for the final classification layer. "Char" is an abbreviation for character, with No meaning that the character-level CNN output was not included, and the cased and uncased models specifying if the character-level CNN includes both uppercase or lowercase characters or not. "F. clas" is an abbreviation for final classifier layer.

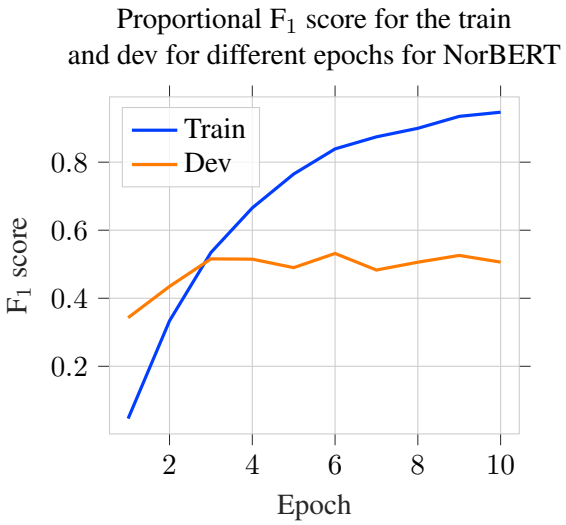


Figure 1: The plot shows the proportional F_1 score for different epochs during training and validation for the training of the NorBERT model combined with a uncased character-level CNN embedder.

4 Discussion

The language-specific NorBERT model vastly outperforms the multilingual model. The most interesting finding in this article is how much better performance we obtain from the NorBERT model as compared to the multilingual one. We know from Pires et al. (2019) that mBERT actually seem to create multilingual representations, which should mean that doing transfer learning to Norwegian should be doable. On the other hand, the study suggests that the ease of transfer is much higher for similar languages, as opposed to different ones, and since there are a lot of languages in the train-

LR Clas	LR Emb	Prop F_1	Bin F_1
0.01	1e-04	0.1057	0.2091
0.01	1e-05	0.4077	0.6327
0.01	1e-06	0.1948	0.2882
0.01	0	0.1648	0.2587
0.001	1e-04	0.4257	0.6105
0.001	1e-05	0.5258	0.6632
0.001	1e-06	0.5072	0.6313
0.001	1e-04	0.4257	0.6105
0.001	1e-05	0.5258	0.6632
0.001	1e-06	0.5072	0.6313
0.001	0	0.4886	0.6293
0.0001	1e-04	0.3829	0.5638
0.0001	**1e-05**	**0.5271**	0.6642
0.0001	1e-06	0.4986	**0.6708**
0.0001	0	0.4637	0.6387

Table 3: The table shows the best proportional and binary F_1 score obtained for different learning rates. "LR Clas" is an abbreviation for the learning rate for the classifier layer, while "LR Emb" is the learning rate for the other embedder layers.

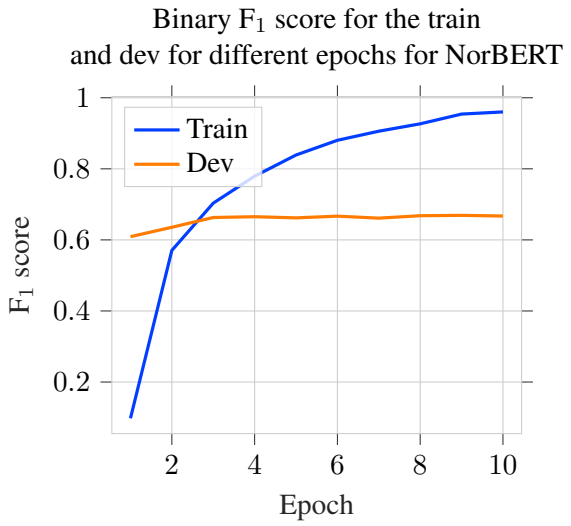


Figure 2: The plot shows the binary F_1 score for different epochs during training and validation for the training of the NorBERT model combined with a uncased character-level CNN embedder.

ing data for the mBERT model, this can be one of the reasons for the worse performance. Research also seems to suggest that language-specific models can perform significantly better than multilingual versions, as highlighted by Virtanen et al. (2019), giving yet another potential reason for the worse performance.

Making sure the mBERT model is able to adjust to Norwegian. Libovický et al. (2019) suggests that mBERT is made up of one language-neutral component, and one language-specific component. It is also know that training BERT-models is hard due to vanishing gradients and general instability (Mosbach et al., 2020). The training dataset contains 8634 sentences, which might not be enough to allow the big mBERT model to adjust to Norwegian. One could look into start by training the embedder on data without the targeted sentiment analysis, and see if that improves the performance of the model.

Weight decay	Prop F_1	Bin F_1
0.01	0.5258	0.6632
0.05	0.5306	0.6707
0.1	0.5301	0.6756
0.2	0.5371	0.6819
0.3	0.5359	0.6807
0.4	0.5344	0.6832
0.5	0.5402	0.6868
0.6	**0.5469**	**0.6892**

Table 4: The table show the best proportional and binary F_1 score obtained for different values for the weight decay in the AdamW optimizer.

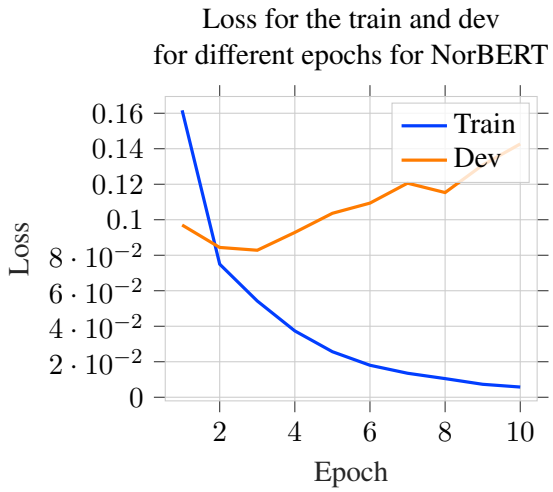


Figure 3: The plot shows the loss for different epochs during training and validation for the training of the NorBERT model combined with a uncased character-level CNN embedder.

Character-level information improves the performance of the NorBERT. From table 2 we see that the best performing models are those trained with both an embedder part and a character-based part. This might indicate that there are some patterns which the big BERT model is not able to

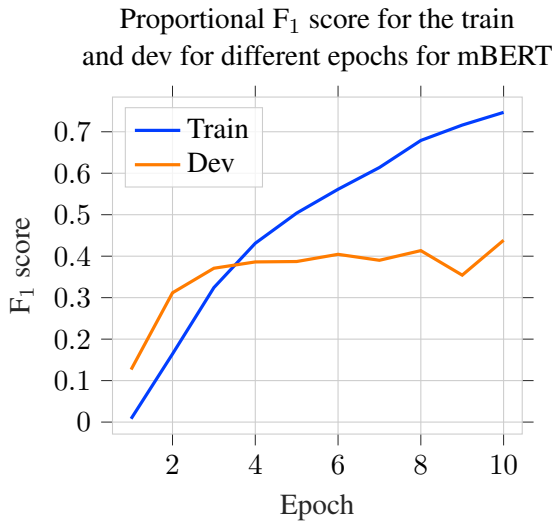


Figure 4: The plot shows the proportional F_1 score for different epochs during training and validation for the training of the mBERT model combined with a uncased character-level CNN embedder.

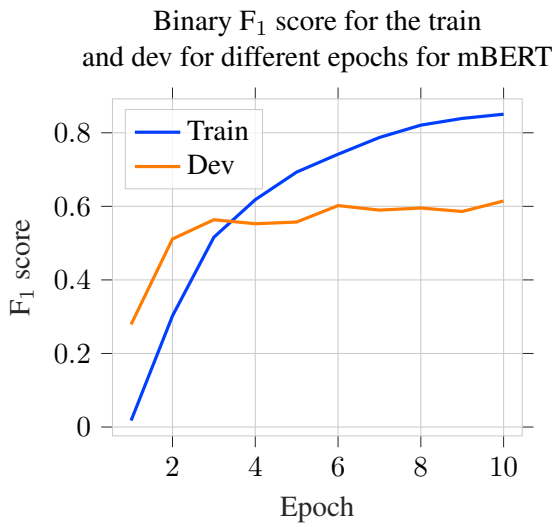


Figure 5: The plot shows the binary F_1 score for different epochs during training and validation for the training of the mBERT model combined with a uncased character-level CNN embedder.

capture. One should however note that the performance difference is not very big, and the evidence is merely empirical, so no definite conclusion that character-level information will improve BERT models should be drawn from this.

Character-level information does not seem to have a noticeable impact on the performance of the mBERT model. As opposed to the NorBERT models, it is not clear whether the character-level information improves the performance or not, with some scores being higher for the models including the character embedding and some performing worse, as seen in table 2. One possible explanation for the difference in performance, is that whereas the NorBERT model trained on Norwegian data only requires minor adjustments, the mBERT model requires much bigger changes to adapt to the Norwegian language, meaning that having even more parameters to optimize simultaneously only hurts performance. If the training set had been even bigger, the results could have been different, and we could have gotten better performance when including the character-level CNN, but more research is required in order to either verify or deny this.

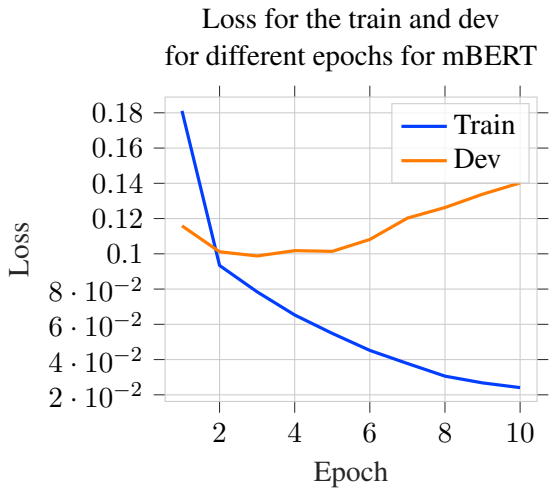


Figure 6: The plot shows the loss score for different epochs during training and validation for the training of the mBERT model combined with a uncased character-level CNN embedder.

The models might be quite prone to overfitting. If we look at the plots in figure 1, 2, 3, 4, 5 and 6, we see a clear difference between loss and F_1 performance for the train and dev data. For the train data, the binary and proportional F_1 score consistently improve. Judging from the figures, the score also does not look like it has stagnated, meaning that given enough time, they might very well be able to have a 100% F_1 score. This does however not appear to be the case on the dev data, with performance looking much flatter, and certainly not improving at nearly the same rate as for the training data. The loss tells a similar story, with the train loss decreasing, while the dev loss increases for both the NorBERT and mBERT graphs.

Fine-tuning the embedder layer in addition to the classifier improves the performance. From table 3, we clearly see that even for the NorBERT model, which is already trained on Norwegian

words, fine tuning the embedder improves the performance noticeably, with improvements around 5.5% for the proportional F_1 score and 7.8% for the binary F_1. This indicates that even for a model trained on the same language, tweaking it to fit the exact data is still worthwhile.

5 Conclusion

This study offers an empirical look at combining different concepts from machine learning to improve the performance of a standard BERT-approaches for targeted sentiment analysis in Norwegian. On one hand, the analysis indicates that using a character-level CNN in conjunction with the pre-trained embedder and training multiple models can improve performance. On the other hand, data suggests that the entire BERT model might be unstable, with much better performance on training data than on unseen data, and with this being the case also when including the character-level CNN. The final performance of the model was 54.44% for the proporional F_1 and 66.76% for the binary F_1 when combining ten ensemble models.

Suggestions for further work include doing a more thurough examination of the errors, to better understand if and how the model is unstable. Another possibility is trying to investigate how different CNN-architectures can improve the performance for the character-level embedder. Another possible direction to improve the performance, would be to train the embedder on a larger dataset, somewhat similar to how BERT models are trained, and only fine-tune it on the smaller targeted sentiment analysis-dataset.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Andrey Kutuzov, Jeremy Barnes, Erik Velldal, Lilja Øvrelid, and Stephan Oepen. 2021. Large-scale contextualised language modelling for Norwegian. In *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 30–40, Reykjavik, Iceland (Online). Linköping University Electronic Press, Sweden.

Jindrich Libovický, Rudolf Rosa, and Alexander Fraser. 2019. How language-neutral is multilingual bert? *CoRR*, abs/1911.03310.

Ilya Loshchilov and Frank Hutter. 2017. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2020. A fine-grained sentiment dataset for Norwegian. In *Proceedings of the 12th Edition of the Language Resources and Evaluation Conference*, Marseille, France, 2020.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual bert? *CoRR*, abs/1906.01502.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: BERT for finnish. *CoRR*, abs/1912.07076.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

A Code

The code is available at `https://github.uio.no/hermabr/IN5550/exam`. For a more detailed explanation of the code, check out the README-file in the GitHub repo.

Task-oriented Semantic Parsing with a sequence to sequence architecture

Matias Jentoft
University of Oslo
Faculty of Informatics
email@address.com

Abstract

This paper is about automatic semantic parsing for task-oriented, compositional, queries. The task is to extract intentions from natural language sentences, which can then be used for user interaction systems. To explore this task, we have experimented with different variants of sequence to sequence (seq2seq) architectures: varying the sentence representations and the bridging strategies between the encoder and the decoder. We have analyzed some of the challenges in this particular NLP-task when it comes to using pre-trained language models, and find the model we developed that gives the best performance on development data. In the end we test our best model on test data, which is a seq2seq model with pre-trained, static, embeddings in both encoder and decoder. It achieves 0.402 exact match accuracy on the test dataset.

1 Introduction

Semantic parsing is to extract semantic information from natural language text. This can be useful for example in dialogue systems, for a system to be able to understand user utterances (Rongali et al., 2020). A basic understanding of intentions of the user is essential for a system to be able to give reasonable answers or follow-up questions, and to possibly conduct extra-linguistic tasks like doing look ups in a navigational application. The specific sub-task we will focus on is task-oriented semantic parsing (TOSP), where the prime objective is to conduct such aforementioned extra-linguistic tasks. In this paper we use a dataset which uses the TOP-format presented in Gupta et al. (2018) for representing the parse containing this semantic information. The format is hierarchical, or tree-structured. As sequence to sequence (hereafter just seq2seq) architectures work with linear sequences, we will work with a linearized or serialized version of this tree.

The baseline we have access to, gives an exact match accuracy score of 0.38. By studying the nature of the task, and the current methodology in Natural Language Processing, we improve this baseline, and create a TOSP-system that can be a better starting point for a dialogue system further downstream.

The dataset we use is the TOP dataset. The dataset has 44 000 queries. It was presented in Gupta et al. (2018), and uses a novel hierarchical representation of queries. The queries are crowd sourced, where the contributors are instructed to create sentences or phrases that could be asked a system that assists with navigation and event queries. Some of the queries are short, like “pampering parties", and some are longer and consist of fully grammatical sentences, like “If I take off now what time should I arrive at Fenway Park?". Expert annotators have created the parses for the queries. The parses are represented as trees, with the leaves or terminals being words from the query, and the non-terminal nodes being Intents or Slots. The top node of each parse has to be an Intent. However, this representation allows for complex queries with nested intents, as can be seen from this serialized version of a complex parse from TOP in table 1. This is opposed to representations with only one top intent for the whole query, with slots inside it. Intents describe a wish for something to be done by the system, and the slots describe entities that are relevant for a system to know for executing the request further down in the pipeline. The use of the TOP format gives us the opportunity to handle a larger array of queries, which is a prerequisite for a proper, human-like, understanding of meaning of natural language.

The evaluation metric which is common in this sub-field is exact match accuracy (Gupta et al., 2018; Einolghozati et al., 2019), which is defined to be a full match between prediction and gold standard on utterance or query level. In addition,

sometimes recall and precision scores as labelled bracketing scores common in syntactic parsing (Gupta et al., 2018) (which is a related task in NLP), are reported. This metric compares the span of a given label within each utterance, which means that partly correct parses on query level will contribute to the score as well. Another informative metric available is that of tree validity. This metric does not compare the predictions to any gold standard, but instead looks at each prediction and measures weather it abides to the rules of TOP, such that the prediction could have been a valid parse.

In this paper we mostly do not use any other evaluations than exact match accuracy in this paper. Having only one metric makes it easier to directly compare different models and architectures. One additional argument for only weighting full query-level matches is that most downstream tasks, like query execution by a dialogue system, are expected to be performed exactly as expected. If a request for driving directions to a specific restaurant is parsed in any other way, the expectations of the user are not met, and the system is not useful. However, in section 3.5 we have a look at some options for post-processing to manually improve the validity of trees directly.

For the reader with access to the code used for the experimentation, it should be noted that we did not use the textual predictions themselves for the comparison of predictions with gold parses. Instead we use the numerical identifiers for each item in the vocabulary for comparison, and thereby increase the efficiency of evaluation greatly, both during the training epochs and during the final, reported, evaluation. This is the metric used in all the tables reporting results.

TOP dataset	
Input Sentence	**Parse**
stuff to do tonight	[IN:GET-EVENT stuff to do [SL:DATE-TIME tonight]]
Traffic near me	[IN:GET-INFO-TRAFFIC Traffic [SL:LOCATION [IN:GET-LOCATION [SL:SEARCH-RADIUS near] [SL:LOCATION-USER me]]]]

Table 1: Examples from TOP dataset: one simple and one complex (nested) query parse

2 Related work

There are two main directions for solving TOSP. Either one can predict the serialized version of the parse-tree directly with a seq2seq strategy, like in Rongali et al. (2020) and Chen et al. (2020), or one can predict a sequence of transformation-actions, which when applied to the query creates this parse-tree, like in Gupta et al. (2018) and Einolghozati et al. (2019). All of these papers work on the same dataset as us: the TOP dataset which includes complex queries.

We will now briefly describe two systems from the literature. Rongali et al. (2020) presents a seq2seq system, with some task-specific peculiarities. For the encoding part, they use pre-trained BERT embeddings, which is the state-of-the-art method for representing text in NLP as of today. These embeddings are not static as the ones we use in this paper. Each time a token is encoded at inference time, a unique representation is created based on the current context. The more interesting part of their system, is that they use a Pointer Generator Network (See et al., 2017) for the decoder. This network generates tokens in two ways: either by copying tokens from the input sequence with the help of pointers, or generating tokens not seen in the input from a limited special vocabulary. Earlier, this type of network has been used for text summarization (See et al., 2017). This strategy works for this particular task as well, since all the elements in the input should also be in the output, and the special token vocabulary is very small (less than 100 intent and slot types). This is similar to the task of text summarization, where parts of the input (the original text) can be used in the output (the summary). They achieve exact match accuracy of 87.67 on the TOP dataset with their unified model in 2020.

In Einolghozati et al. (2019), a more complex ensemble model is presented. The embeddings are contextualised just as in the previous paper, but in the encoder a ranking method decides which of the predictions from the ensemble of models gets chosen. This means that several models with different hyperparameters are trained on the training data, and a final prediction is chosen from several possible predictions.

Both of these mentioned models are different from the models presented in this paper, in terms of embeddings used, Pointer Generator Network added, and in that the second is utilizing an ensem-

ble of models.

3 Architecture, experiments and results

In this section we present base architecture of the system used for experimentation, and we gradually add elements to this system. The results after each experiment on the development set are presented and discussed.

3.1 Baseline with improvements

In this article we only work with the direct parse-prediction of a seq2seq model. Seq2seq is a common architecture in NLP for systems that creates a new sequence from an old sequence, and where there is not a 1-to-1 relationship in number of tokens or elements in the input and output (Cho et al., 2014; Bahdanau et al., 2014; Sutskever et al., 2014). Machine translation between languages is a typical example of this, while word-class tagging (POS-tagging) is an example of a task where the output has the same length as the input. In seq2seq implementations, the whole input has to be encoded to a common size vector first, and this is done by the module named encoder. The resulting context-vector can then be used by the decoder-part of the system to generate an arbitrary length sequence. This is a simplified description of seq2seq, and in reality additional modules, like attention, are used by the decoder to weight the different parts of the encoded input when generating the output. This is due to the limitations of what information the context vector can carry.

The baseline model is the starting point for all the rest of this task. It is a seq2seq architecture, consisting of a Recurrent Neural Network (from now on just RNN) in both encoder and decoder. The output of the encoder is autoregressively used by the decoder to create the output sequence, which consists of tokens from the input sentence, as well as structural tokens, which are needed for a serialized semantic tree (e.g. IN:[] and so on). The encoder utilizes pre-trained, static, publicly available [1] embeddings trained on English wikipedia 2021 with assistance of the Gensim Contiuous Skip-gram approach (Fares et al., 2017) as representation for each token. The decoder uses a different context vector at each time steps, regulated by an additive attention mechanism (Bahdanau et al., 2014). This mechanism consists of a one-layer neural network, with weights trained to learn the relationship or alignment between each token in the input to each token in the output. The decoder trains embeddings from scratch, based on the vocabulary of the parses in the training data, which includes special tokens like brackets and intent- and slot names, see table 1.

The results on the development data, before making any adjustments, can be seen in table 2. The hyperparameters for the baseline are 20 epochs, batch-size 248, two (encoder and decoder) unidirectional RNNs with 1 hidden layer each, and static word embeddings created from the English Wikipedia Dump of November 2021.

We created a slightly improved version of this baseline, and the results from this can be seen in table 2 as well. Here we have added a decay of the teacher forcing ratio (0.95) and a decay in the learning rate (0.95) during training. The learning rate starts at a quite high level (0.003), to have increased efficiency in the training. If the learning rate starts out too low, it will take a very long time to get the parameters to make good predictions. With a high starting learning rate the parameters in the model change comparatively much in the right direction. But when the model becomes quite good at predicting, it is good to have a lower learning rate so that the parameters do not fluctuate too much, and go "past" the golden spot. Therefore, the learning rate decays exponentially toward the end of training.

Teacher forcing is a concept and an algorithm in auto regressive generation training, like the training of an RNN (Williams and Zipser, 1989). In the beginning of training, the predictions at a given timestep in the decoder will most likely be wrong more often than not. If there is no teacher forcing, the generation at the next timestep will be based on a preceding sequence, which is not correct or likely in the dataset. Therefore, in the beginning of training, it is useful to force the sequence up to that point to be the correct one, and predict the current token based on that. But since the model is supposed to be able to generate parses on unseen data as well, we want the last part of the training to simulate that. This means, that when the model is becoming quite good, we decrease the ratio of timesteps when teacher forcing is used. This is done exponentially by multiplying the teacher forcing rate by a number between 0 and 1.

We also added a fine-tuning of the encoder pre-trained embedding parameters during training. The

[1] http://vectors.nlpl.eu/repository/

motivation for this is that the pre-trained embedding values are trained with a different objective than TOSP. With TOSP, some words in the input might be cues for a specific intent in the parse, and these connections are emphasized by allowing some fine-tuning of these parameters.

These three changes to the system were not systematically tuned in a grid-like way, but are rather collectively assumed to improve the baseline, as they do.

Baseline results	
Model	**Exact Match Accuracy**
Baseline	0.388
Improved baseline	0.423

Table 2: Baseline results on development data set

3.2 Adding directions and layers to the encoder

In modern machine learning, the number of trainable parameters in a system can improve the systems ability to create useful representations. In a neural network, this can mean either to expand the number of nodes in each layer, or by adding more layers of nodes. Both of these changes result in more trainable parameters between the nodes. Adding more layers is what gives the concept of deep learning its name. It is not a trivial addition to a seq2seq model to add more layers, and this will be discussed and experimented with in section 3.3. In a sequential neural network like RNN, where outputs at a given timestep depend on the calculations from previous timesteps (previous words), one can also add a sequential reading from the end. The result is that a representation for a word is also impacted by the upcoming words. This is called bidirectional processing (Schuster and Paliwal, 1997). Results comparing the unidirectional and bidirectional encoder versions of the system can be seen in the table 3. The hyperparameters are the same as before, but the bridging strategy used is a concatenation of the direction outputs of the encoder, and then a linear transformation to a fixed sized context vector fitting as input to the decoder. We will describe what bridging is in section 3.3.

From the results we can see that bidirectionality in the encoder does not improve the model, in fact the results are quite poor on the development data set. It seems like the information about future words in the input sequence does not contribute to the prediction of intents in the parse. One hypothesis is that information about intentions/intents in the English language follow a quite strict word order, and can thereby be extracted well from only the left context of a given token.

Bidirectional encoder	
Model	**Exact Match Accuracy**
Improved baseline	0.423
Bidirectional encoder	0.169

Table 3: Results on development data set, from experiments with bidirectional encoder compared to the unidirectional improved baseline

In the following experiments on adding more layers to the encoder, we only use the unidirectional version of the encoder. This is due to the bidirectional model yielding quite poor results in the previous experiments. In an RNN, layers are stacked on top of each other, and the output of the first layer is the input of the second layer, and so on until the last layer, which creates the actual outputs used by the attention module of the decoder.

As can be seen from the results below, adding layers does not improve the results of the model either. This may be because the bridging strategy used so far (concatenation and linear transformation) is not optimal. We will present more experiments with bridging strategies in the next section.

Layers	
Model	**Exact Match Accuracy**
1 layer	0.377
2 layers	0.254
3 layers	0.140
4 layers	0.001

Table 4: Experiments with different layer counts in the encoder. The 1 layer model has forced linear transformation in the "bridge", like the other models in this experiment. This is the reason why the result in the first column deviates from the results in table 3. All results on development data

3.3 Bridging from encoder to decoder

As mentioned earlier, in a seq2seq model there is an encoder which outputs a fixed-sized context vector. In the baseline this context-vector consists of the raw, last (and only) layer of the encoder. This only works because that vector is exactly the same

size as the hidden (input) layers of the decoder. If there is a difference in shape and size between the two vectors, we need a transformation, or a bridge, between the encoder and the decoder. In this section, we will discuss what different options we have for this bridging, and what issues one encounters if there are more layers in the encoder. Some illustrations of what bridges are and can be are seen in figure 1. One simple solution is to discard the outputs of the other layers, and only use the last layer as input to the decoder. One can also take the element-wise average of all the hidden layers. Another option is to take the highest values for each position in the feature space from the encoder last hidden layers, and create a new vector from that: a strategy called max pooling. For all of the alternatives except concatenation, it is optional to have some transformation of the data to create the final context vector. We suggest two different types of transformations: a linear (matrix multiplication) operation with trainable parameters, and a tiny multi-layer perceptron (feed-forward neural network) with a non-linear transformation. The results of experiments with these settings can be seen in table 5.

For the experiments with bridging we used the best non-1-layer model from the previous section. In our case that was the (unidirectional) model with 2 layers. The motivation for this decision is that a 1-layer encoder would not let us try out all the possible bridging strategies, like the averaging or concatenating of layer outputs. There is a caveat here however, and that is that another, and perhaps more intuitive, bridging model like using the last layer might have yielded different results when experimenting with layer sizes. To do a full grid search on all combinations is outside the scope of this paper.

None of the bridging strategies help improve the improved baseline with only 1 layer and no transformation. It is difficult to say why that is, as more layers is common in neural machine learning, and we have added more parameters in both the encoder and the bridge. This means that a deeper understanding of neural networks is more important than just throwing parameters into a system.

3.4 Embeddings

Already in the improved baseline model we added fine-tuning to the encoder embeddings, which improved the performance of the system. Let us now

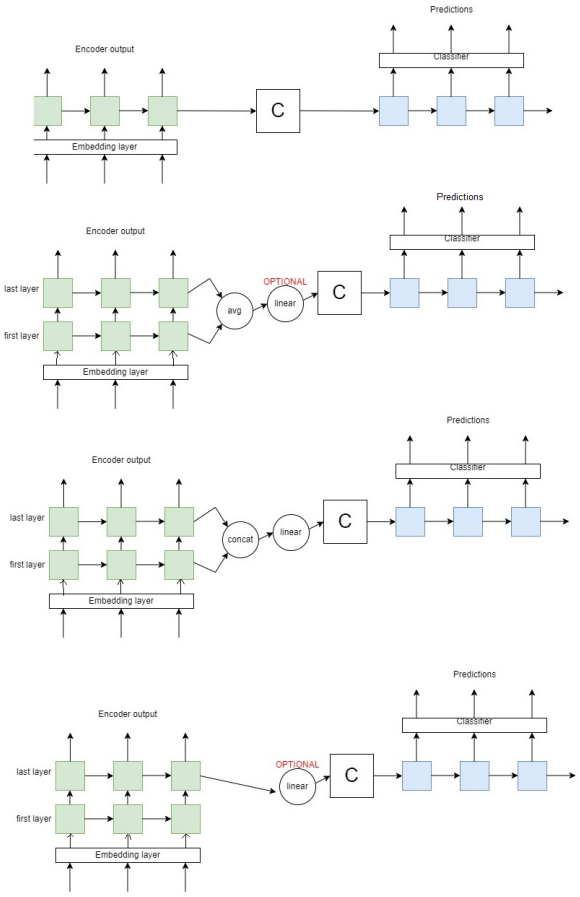


Figure 1: Some different bridge models. All models ignore the attention mechanism, and focus only on how the context vector is generated. The first illustration describes the baseline, where the context vector is the last hidden layer of the encoder. The second shows how to average over the multiple hidden layers. The third shows how one needs to have a transformation for concatenated hidden layers. The last illustration shows how to use only the last layer as input to the encoder. For number 2 and 4 a transformation is optional

turn to the decoder representations. In the baseline, the representation for the decoder generative vocabulary is trained from scratch. This means that each token present in the parses from the training data, gets an individual, random, numerical representation, which is updated and improved for the particular task during training. This is not optimal, as it takes time and resources to adjust the parameters from random to optimal for a task. Usually one would use pre-trained embeddings (which already carry semantic information, if not for this specific task) in such an NLP task. However, since the output of the system uses special tokens (intent and slot names) which are not common in natural language text, there are no fully covering pre-trained

Bridging		
Model / pooling strategy	**Transformation**	**Exact Match Accuracy**
Improved baseline	na	**0.423**
Last layer	linear	0.355
Last layer	MLP	0.284
Concat	linear	0.339
Concat	MLP	0.358
Average	linear	**0.374**
Average	MLP	0.368
Max pooling	linear	0.371
Max pooling	MLP	0.350

Table 5: Experiments with different bridging strategies between the encoder and the decoder. All experiments with 2 layers, on the development data set

embeddings available, even if it would be possible to train such embeddings on data in the format of the TOP parses. The Pointer Generator Network (Rongali et al., 2020) mentioned in an earlier section could also solve this issue quite nicely, by using only the limited special vocabulary and pointers to the input tokens. However, both training new embeddings with for example skip-gram or implementing a Pointer Generator Network was outside of the cope of this article. As mentioned above, the baseline has pre-trained static embeddings for the encoder (which are fine-tuned for the task in the improved baseline), and trainable embeddings for the decoder. In the following section we present experiments with pre-trained embeddings for the decoder as well.

The issue with using pre-trained embeddings for the decoder, is that the output vocabulary need to have some special tokens (see table 1), which are not found in any pre-trained embeddings. Another issue is that using all tokens from the pre-trained embedding vocabulary creates a huge amount of output classes for the decoder to predict from (200 000), which is too much for the memory of the systems we are using for training (SAGA supercluster from NRIS, Norway) Therefore the pre-trained embeddings are created in the following manner: first we downloaded the same pre-trained embeddings as for the encoder, then we filtered this collection to only keep the vocabulary which already exists in the parses of the training data. Then we added all the special tokens used in this parsing schema: the intent and slot types. This was done by creating an average embedding based on the parts of the intent-symbols, by averaging the embeddings for for example “get" and “directions" in “[IN:get-directions" into one vector which carries meaning from both of them. In table 6 we see the results of this, compared to the baseline. To get some more points of comparison, we did the experiments with both 1 and 3 layer encoders. The reason for using 3 layers instead of 2, which would be the best non-1-layer configuration according to the results from section 3.2, is to have as wide a range of comparison as possible with reasonable experimenting efficiency.

For these experiments we used the linear transformation in the bridge, with an average pooling of the hidden last layers of the encoder, as this was the one that yielded the best results in the previous experiments.

The results for multiple layers seem to be very unstable at best. The addition of pre-trained embeddings for the decoder actually deteriorates the results, which is the opposite effect as for the 1-layer configuration. This is quite surprising, since one would think that the larger parameter-count in the 3-layer model, given equal training epoch count, should benefit from the pre-trained embeddings already containing some predictive ability as opposed to the non-trained embeddings. Because of this, we will focus on the results from the 1 layer experiments. Using the pre-trained embeddings with additional special tokens seems to be a viable option for this configuration, and it also cuts the training time with about 20 percent.

Embeddings for decoder		
layers	**Decoder Embeddings**	**Exact Match Accuracy**
1 layer	from scratch	0.423
1 layer	pre-trained	**0.440**
3 layers	from scratch	0.300
3 layers	pre-trained	0.242

Table 6: Experiments with pre-trained embeddings for the decoder, on the development data set

3.5 Post-processing

Looking at the final results on the development data set, we can see that some of the predictions of the

system do not conform to the TOP parsing rules, see example in table 7. In a linearized/serialized tree, there needs to be a balanced number of square brackets. With the simple trick of making sure that the amount of square brackets at the end of each parse is correct, we make sure that the outputs follow the rules of the parsing system. There is naturally no guarantee that the adding of brackets at the end of the parse is correct, since the closing brackets might rather belong in an earlier position. Studying different heuristics for making sure the parses are valid can be an avenue for future work

Bracket problem example
[IN:GET-EVENT best [SL:CATEGORY-EVENT concerts] [SL:LOCATION Vegas]]]]]]]]]

Table 7: Prediction from the system, that is breaking the rules of a linearized tree

4 Testing on the test set

To make a fair comparison, we are running the baseline on the test-data as well, not on the development data. In that way we can see if our system has brought any improvement to the TOSP task. After landing on one specific model we can run it on the unseen test set. The settings/hyperparameters of the best model are repeated in table 8.

Parameters of the best model	
parameter	**value**
teacher forcing decay	0.95/100 batches
encoder layers	1 layer
bridge pooling	average
bridge type	linear transformation
decoder embeddings	pre-trained +
encoder embeddings	fine-tuned

Table 8: The parameters of our best model from tests on the development data, which will be evaluated on the test data

Results on test data	
model	**Exact match accuracy**
baseline	0.392
best model	**0.402**

Table 9: The results after testing on unseen test data

The results on the test set are showed in table 9. Compared to previous experiments on the TOP dataset, these results are not very impressive. The state-of-the-art systems are just below 0.9, like the extended SVM ranking model with ELMo-embeddings in Einolghozati et al. (2019) which yields an exact match accuracy of 87.25, or the model in Rongali et al. (2020) with a Pointer Generator Network, which yields EMA of 86.67 on the same data set. However, we manage to improve the results twice: first by doing some changes to the baseline, and then by adding pre-trained embeddings to the encoder of the seq2seq model.

Wrong predictions	
Correct parse	**System parse**
[IN:GET-EVENT Where can we take [SL:ATTRIBUTE-EVENT the kids]]	[IN:UNSUPPORTED Where can we take the the kids]]
[IN:GET-EVENT What is going on [SL:DATE-TIME right now]]	[IN:GET-INFO-TRAFFIC What is going on [SL:DATE-TIME right now]]
[IN:UNSUPPORTED outdoor dining in miami]	[IN:GET-EVENT [SL:ATTRIBUTE-EVENT outdoor] [SL:CATEGORY-EVENT in [SL:LOCATION chicago]]

Table 10: Examples of wrong parses made by our model

In table 10 are some examples of mistakes from the parser. The examples are taken from predictions made on the development data. For the first example, our system predicts the unsupported category of intents, which is an actual category in the dataset. It does not seem to recognize the phrase "where can we take" as a wish to find a place. And "the kids" is not the most obvious target of an event either. For the second example, the whole tree structure is correct, but the intent type is wrong. It could be argued that if this phrase is uttered to a navigation system in a car, the system parse would be an acceptable interpretation as well. In the third example, the annotators have categorised the phrase as unsupported, while the system "sees" a pattern with other event requests. However, the system does not carry the "dining" part from the input (which would have

been solved with a Pointer Generator Network by copying from the input). We can also see an example of how intuitively similar words like Miami and Chicago probably have quite similar embeddings, and are intermixed in the prediction.

5 Conclusions

From our experiments we have seen that it is not trivial what is fed from the encoder to the decoder, and there are many ways to go about it. Averaging over the hidden states of the encoder seems to be the best way to do the bridging. We think the detailed experimentation with different bridging strategies could be useful for other researchers working with seq2seq systems for TOP. We have also showed that it is possible to use pre-trained embeddings for non-standard vocabularies, by combining elements of the already existing vocabulary to create useful embeddings for special tokens. Lastly, using teacher forcing decay, fine-tuned embeddings in the encoder, and learning rate decay appears to improve the performance of such systems.

The context vector is not the only element used by the decoder in seq2seq system when generating an output. There is also the attention-module, which utilizes the outputs of each encoder timestep separately. We did not study different options for the attention mechanism in this paper, but there are several paradigms and parameters to experiment with in that sub-field as well.

Acknowledgements

The authors want to thank the four reviewers for their useful comments on the draft of this article. These comments improved the structure and content of the paper greatly, and also taught the authors a thing or two about formatting of articles with LaTeX.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. 2020. Low-resource domain adaptation for compositional task-oriented semantic parsing. *CoRR*, abs/2010.03546.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer. 2019. Improving semantic parsing for task oriented dialog. *CoRR*, abs/1902.06000.

Murhaf Fares, Andrey Kutuzov, Stephan Oepen, and Erik Velldal. 2017. Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NODALIDA 2017, Gothenburg, Sweden, May 22-24, 2017*, pages 271–276. Association for Computational Linguistics.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2787–2792, Brussels, Belgium. Association for Computational Linguistics.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! A sequence to sequence architecture for task-oriented semantic parsing. *CoRR*, abs/2001.11458.

Mike Schuster and Kuldip Paliwal. 1997. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. *CoRR*, abs/1704.04368.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Targeted Sentiment Analysis on $NoReC_{fine}$ dataset using pre-trained language models and RNNs

Rohullah Akbari
University of Oslo
rohullaa@uio.no

Liang Jia
University of Oslo
liangj@uio.no

Ece Cetinoglu
University of Oslo
ecec@uio.no

Abstract

In this paper, we applied different neural network architectures in order to classify the targeted sentiment in the $NoReC_{fine}$ dataset[1]. The architectures we have used are recurrent neural network such as Gated recurrent unit (GRU) and Long short-term memory (LSTM), the Norwegian ELMo ($NorELMo_{30}$), pre-trained transformer-based language models such as the NorBERT and the Multilingual BERT. We trained these models with and without including character-level information and encoding types of BIO and BIOUL. The pre-trained models significantly improve the performance compared with RNNs. The best model was found to be the NorBERT on the BIO encoded dataset with F_1 at **0.644** (binary) and **0.513** (proportional).

1 Introduction

Sentiment Analysis (SA) is a research field within Natural Language Processing (NLP) that aims to identify and categorize opinions in text and determine whether they are written in a positive, negative, or neutral tone. However, there could be more than one sentiment towards different targets in one sentence. A fine-grained sentiment analysis or targeted sentiment analysis (TSA) is used to solve this problem, which is identifying opinions in sentences and analyzing them regarding their polar expressions, targets, and holders. For example, the sentence below has only one sentiment towards one target, where the *normal* sentiment analysis model works fine.

(1) Denne disken$_{\text{POS}}$ er svært stillegående
'*This disk is very quiet-going*'

But the second sentence has two different sentiments towards two different targets, where normal sentiment analysis cannot identify its sentiments.

(2) En klassisk, men like så episk oppbygget inspo-pop som fengsler i refrenget$_{\text{POS}}$, men Perry$_{\text{NEG}}$ sliter med den dypere vokalen i versene.
'*A classic, but just as epic built-up inspo-pop that captivates with the chorus but Perry struggles with the deeper vocals in the verses.*'

The aim of this paper is to develop machine learning models that are able to classify these types of sentiments. In order to do so, we will apply a range of different neural network architectures, such as

(1) Recurrent neural network
(2) ELMo
(3) BERT models

The selected models for Recurrent neural networks (RNN) and pretrained models were also tested with included character-level information. The two best models are then applied with another more fine-grained label encoding of BIOUL.

2 The dataset

In this task we are working with the $NoReC_{fine}$ dataset (Lilja Øvrelid and Velldal, 2020) which is a dataset for fine-grained sentiment analysis in Norwegian. We have three separate data files for training, validating, and testing. The size of the dataset was shown in the table [1]. It has been annotated with respect to polar expressions, targets, and holders of opinion. The distribution of different polar labels was shown in table [2]. The dataset is encoded in **B**egin **I**nside **O**utside (BIO) type of encoding. In total, there are five classes in the data

(1) **O**: outside
(2) **B-targ-Positive**: beginning of a target with

[1]The code for this paper can be found at https://github.uio.no/ecec/TSA. Read the README.md for how to run the code.

positive sentiment
(3) **I-targ-Positive**: inside of a target with positive sentiment
(4) **B-targ-Negative**: beginning of a target with negative sentiment
(5) **I-targ-Negative**: inside of a target with negative sentiment

An example of a sample from the dataset is shown in table [3].

Train	Test	Development	Total
8633	1271	1530	11434

Table 1: Dataset size of training, test and development sets

Label	Train	Test	Dev	Total
B-neg	1558	210	256	2024
B-pos	3486	525	620	4631
I-neg	1472	163	244	1879
I-pos	3399	572	581	4552
O	134317	20355	24195	178867
Neg (total)	3030	373	500	3903
Pos (total)	6885	1097	1201	9184

Table 2: Number of each tag appearing in training, test and development sets.

We observe that the dataset mostly consists of words with neutral (O-tagged) sentiment, with very few words with positive sentiment and even fewer words with negative sentiment (table [2]). It seems that the separation of training, development, and test datasets has already considered the imbalance of different labels and they have a similar distribution for different labels.

Han	B-targ-Neg
redda	O
den	O
perfekte	O
hustrue-skuespilleren	O
Bonnie	B-targ-Pos
Bedelia	I-targ-Pos
og	O
blødde	O
overfladisk	O
.	O

Table 3: Sample 99 from training dataset. Here, the *B-targ-Neg*, *B-targ-Pos* indicate the beginning of a token which has been annotated negative and positive, *I-targ-Pos* indicate the inside of a token with positive annotation and the *O* indicate the outside (neutral) token.

3 Architectures and Methodologies

A diverse collection of models were trained and tested for the task. All models were fine-tuned at the end. When the hyperparameters which yield the best results were found for each model, we tested the models with the optimal hyperparameters for different encodings of the gold labels and/or additional embeddings which provide character-level information.

3.1 Recurrent neural networks (RNNs)

Recurrent Neural Networks (RNNs) like Gated recurrent unit (GRU) and Long-short term memory (LSTM) have been proved to be effective in TSA and bringing significant improvements (Zhang et al., 2016; Yukun Ma and Cambria, 2018, 2019). Therefore, we are interested in how these networks deal with the $NoReC_{fine}$ dataset. For all of the RNNs, we used the static embedding with an ID of 58 trained on the Norwegian-Bokmaal CoNLL17 corpus from NLPL [2].

3.1.1 Baseline

The baseline model for this task was a simple bidirectional long short-term memory (BiLSTM) neural network with a linear output layer. We have later on adjusted this model such that it could be used with different versions of encodings for the gold labels, or with embeddings that provide character-level information.

[2]The embedding is available at http://vectors.nlpl.eu/repository/20/58.zip

3.1.2 GRUs and LSTMs

We have built two different recurrent neural network models for this task, and tested them for a variety of hyperparameters. Firstly, we improved the baseline model and ensured that the model had some additional properties and parameters to be tuned. The baseline model was a simple bidirectional LSTM, whereas our improved model can be bidirectional or non-bidirectional, can use BIOUL encoded target labels, can add character-level information on top of the pre-trained embeddings, and most common properties of a neural network, such as the size of the hidden dimensions or the number of layers, can be tuned. Additionally, a GRU model can be trained with the same properties.

3.2 Embeddings from Language Model (ELMo)

The problem with static embeddings, section 3.1, is that these embeddings assign the same vectors to same words in different contexts. This leads to incorrect classification. The solution to this is to use the Embeddings from Language Model (ELMo) which assigns each word a vector which is a function of the whole input phrase (Oren Melamud, 2016; Bryan McCann, 2017). In this way, the words are context-dependent, meaning that ELMo produces different representations for words that share the same spelling but have different meanings. ELMo is based on bi-directional LSTMs to process character-level tokens and generate word-level embeddings (ELMo, 2022).

In this project, we implemented the NorELMo (ID 217) which is an ELMo model for Norwegian[3]. This model is a set of bidirectional recurrent ELMo language models trained on Norwegian Wikipedia texts. The implementation is done by using the *allennlp* package [4]. The complete architecture for this section consists of NorELMo as the first layer, where it extracts word embeddings from the given text-data, and on top of this layer we have implemented a GRU model.

3.3 Transformer models

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based model that relies purely on the self-attention mechanism, which is made possible by a deep bidirectional Transformers at its core. This is essential since a word's meaning can often alter as a phrase progresses. BERT is able to process words in relation to all the other words in a sentence by looking at a word and all the other words that come before and after it at the same time. Because of this, the BERT models have become very popular in the NLP world in recent years, due to their good performance on most NLP tasks.

Another attempt has been made to run BERT models in the $NoReC_{fine}$ dataset. One example is from a master's thesis (Rønningstad, 2020) where the best result obtained with BERT was **0.5958** (binary) and **0.5105** (proportional).

The transformer models implemented in this section are the NorBert (2022) and the MultiBERT (Multilingual Cased) (Devlin and Petrov, 2019). As the name suggests, NorBERT (ID 216) is a BERT model trained on Norwegian data[5]. MultiBERT is a large multilingual model containing 104 languages. Both BERT models were implemented with the *AutoModel* for activating the model and *AutoTokenizer* for tokenizing the sentences from the *transformers* package in PyTorch. On top of the BERT model, a linear layer in order to project the output into the right number of classes.

3.4 Character-level information

Traditionally, we built Neural Language Models using words in a sentence or paragraph or even document. And this indeed provides us with solid performance on different NLP tasks (Darwish, 2013; Hakan Demir, 2014). However, using words to train the model will generate a huge vocabulary and may hinder the improvement of models due to their capacity for handling unknown words, punctuation, and other document structures. Furthermore, this may cause the model to require more time and resources to train. Nevertheless, the character-based models overcome these drawbacks and provide a promising probability. A Hidden Markov Model (HMM) with character-level information could reduce the data sparsity problem, which is inherently present in word-level information, and get 25% error reduction compared with the same model without character information

[3] http://wiki.nlpl.eu/Vectors/norlm/norelmo

[4] https://github.com/allenai/allennlp

[5] http://wiki.nlpl.eu/Vectors/norlm/norbert

(Dan Klein, 2003). Previous studies have proved that the CharacterBERT has better performance than the BERT on different medical domain tasks (Hicham El Boukkouri, 2020).

There are mainly two ways of using character-level information for NLP tasks, which vary based on the nature of the task. For example, Yoon Kim (2015) used the characters to generate a word representation for each token in one sentence, where they employed CNN for word representation. On the other hand, Xiang Zhang (2015) used character information which was not mapped to words first, where they also used CNN but were directly mapped to sentiments and token categories.

In our implementation, we chose the first way of using character information and tried to add the character level information to our word-based deep neural networks. The word character was first encoded with the generated alphabet based on all three data sets, and the character information was extracted using random initialized embedding and applying a 1D convolution layer. After getting the character information, we just concatenate the character features with the features from other methods such as RNNs, BERTs, and ELMo and feed them together to the classification layer.

3.5 Label encoding

There is quite a lot of research effort on NER or TSA, especially generating the state-of-the-art models. However, little research has studied the effects of different annotation schemes on NER or TSA tasks. Based on the authors' knowledge, there is no adequate work that investigates the impact and implications of utilizing multi-annotation systems on Norwegian NER or TSA tasks in the literature. To fill this gap, we will look into the effects of alternative annotation systems on the Norwegian TSA problem. **B**egin **I**nside **O**utside (BIO) of entities are popularly tagging format used for token tagging in chunking tasks like named entity recognition. Unit and last tags were added in BIOUL format to further distinguish the end of tokens and mid-entity tokens, and represent the single token entity. Basically, it may be unnecessary for regular named-entity recognition to distinguish the end of entity token labels from the single entity label. However, it could make a difference when using Conditional random fields (CRFs).

In this project, we created a Python script that transfers the original dataset, which is BIO encoded, to the BIOUL encoded `conll` data file. Then the data file was used in the same way as the original data, with some minor adjustment of the code. The table 4 illustrates an example of the same sentence as table 3 yet with the BIOUL encoding scheme. The new dataset contains the following nine classes

(1) **O**: outside
(2) **B-targ-Positive**: beginning of a token with positive sentiment
(3) **I-targ-Positive**: inside of a token with positive sentiment
(4) **L-TARG-POSITIVE**: last of a token with positive sentiment
(5) **U-TARG-POSITIVE**: single token without L and I with positive sentiment
(6) Same for the negative sentiments for points 2-5.

Han	U-targ-Neg
redda	O
den	O
perfekte	O
hustrue-skuespilleren	O
Bonnie	B-targ-Pos
Bedelia	L-targ-Pos
og	O
blødde	O
overfladisk	O
.	O

Table 4: Same example as table 3 but with BIOUL encoding. Note that this sample contains U-targ-Neg and L-targ-Pos.

3.6 Evaluation

The models were evaluated by *proportional* F_1 and *binary* F_1 metrics. For the proportional, precision is defined as the ratio of overlap with the projected span, whereas recall is defined as the ratio of overlap with the gold span, which reduces to token-level F_1. Any overlapping anticipated and gold span is considered correct by binary. Based on this, we may anticipate that the model will provide better binary F_1 than proportional F_1. a

4 Training

The training of the models was compute-intensive and time consuming. We trained the models on Saga and the Google Colab with a seed number of 5550.

4.1 Tuning parameters

In order to maximize the performance of the model, we tuned the hyper-parameters. The list below shows which parameters were tuned for the different architectures:

1) **LSTM & GRU**: grid search of hidden dimension, amount of dropout between layers, amount dropout in RNN model (except for 1 layer model), whether they were bidirectional or not, number of layers in the RNN and whether character-level information is included or not.

2) **ELMo**: tuning of the RNN model on top of the embedding layer. Almost the same parameters as above; hidden dimension, number of RNN layers, and the amount of dropout.

3) **BERT**: testing BERT and whether character-level information improves the performance or not.

4) **BIOUL encoding**: The models with the best performance were retrained with the BIOUL dataset.

In total, we trained 96 RNN models, 24 ELMo models, and 8 BERT models.

5 Results

The hyper-parameters of each models are listed in the Appendix section.

5.1 Best results with BIO encoding

The best performances of the models are found in table 5. It is clear that NorBERT won among the rest of the models, with **0.644** for binary F1 and **0.513** for proportional F1. And it is noted that all pre-trained models perform better than RNNs. The ELMo and the Multilingual BERT have quite similar results.

5.2 Character-level information

The best performances of the models that were trained with added character-level features are found in table 6. The results show that the added character information makes the models even worse for GRU and LSTM and has little influence on pre-trained models.

Architecture	Binary F_1	Proportional F_1
Baseline	0.328	0.157
GRU	0.383	0.201
LSTM	0.428	0.257
ELMo	0.597	0.476
NorBERT	**0.644**	**0.513**
MultiBERT	0.597	0.440

Table 5: The best performance of the each architectures with BIO encoding. Baseline refers to the BiLSTM with default value. GRU, LSTM, ELMo, NorBERT and MultiBERT refer to the combination of different hyperparameters giving best performance.

Architecture	Binary F_1	Proportional F_1
GRU	0.221	0.134
LSTM	0.398	0.199
NorBERT	**0.642**	**0.516**
MultiBERT	0.593	0.435

Table 6: The best performance of the architectures with included character-level information with BIO encoding.

5.3 Label encoding

The dataset with BIOUL encoding was used to retrain the NorBERT and ELMo models. The result from this is in the table 7. Both scores are a little bit lower than the model on BIO encoded data.

6 Discussion

The fine tuning of hyperparameters improved the baseline model from **0.328** to **0.428** for binary and **0.157** to **0.257** for proportional F_1, as shown in table 5. And similar results were also found in GRU models, but to a smaller extent. However, the fine-tuned version of the LSTM and GRU models do not produce very promising results for TSA on the $NoReC_{fine}$ dataset, with performance still falling below 50%.

Architecture	Binary F_1	Proportional F_1
ELMo	0.582	0.469
NorBERT	**0.614**	**0.483**

Table 7: The performance of NorBERT and ELMo with BIOUL encoding.

Architecture	Time taken/epoch
ELMo	$\sim$ 100s
NorBERT	$\sim$ 110s

Table 8: The time taken for training 1 epoch of NorBERT and ELMo at GPU.

The pretrained models, on the other hand, gave quite promising results. In the table 5, we can see that the NorBERT model has a significant improvement compared with the baseline. This was actually expected since BERT models are the most dominant architectures in the NLP world for the moment. NorBERT is based primarily on Norwegian text from the ground up, which may explain why it outperforms the other pretrained model. This is in line with the research efforts of Jeremy Barnes (2021) and Rønningstad (2020) in this field, where they have also built BERT model on TSA in $NoReC_{fine}$ dataset and achieved the best scores of **0.5958** for binary F_1, **0.5105** for proportional F_1 and **0.6271** for binary F_1, **0.4970** for proportional F_1, respectively. Our score is slightly higher, probably due to the random seed. It is interesting that Rønningstad got his best performance with the Multi-lingual BERT model, while Barnes, like us, obtained the best score with NorBERT.

Another interesting finding was that the Norwegian Elmo model performed better than the MultiBERT, but it could not beat the NorBERT. However, the ELMo is less time-consuming than the NorBERT while training (table 8). With regard to the training time for RNNs, it takes only seconds for each epoch, which is not comparable to the pre-trained models.

The addition of character-level information to the models does not seem to improve the performances, and even makes the RNNs worse. The implementation of character features is a bit naive. We just used the random initialized embedding for the characters of words and did not include any further processing for character features. This could be the possible reason that the extra features from character level does not improve the performance of models. The pre-trained models seem more stable when adding character features, compared with the RNNs.

The resulting scores for NorBERT and ELMo on BIOUL encoding data are similar to those on BIO encoding data. Since we just tested NorBERT and ELMo on BIOUL encoding data, it is unknown whether the RNNs on BIOUL and BIO encoding data vary a lot. On the other hand, Alshammari and Alanazi (2021), and Aasmoe (2019) compared the effects of different encodings for NER on Arabic and Norwegian dataset, and found a better performance on the BIOUL encoding scheme than the BIO encoding, but in this case, we obtained the opposite.

6.1 Conclusion

In this project, we performed targeted sentiment analysis on the $NoReC_{fine}$ dataset. We implemented various neural architectures; GRU, LSTM, ELMo, NorBERT, and MultiBERT. These models were trained with two types of encodings; the BIO encoding from the dataset and the BIOUL encoding. We have fine-tuned the models in order to find the best performance. In our case, we found out that the NorBERT performed best and the Norwegian ELMo performed a bit behind the NorBERT. However, we also confirmed the *performance vs. time* when training the NorBERT and ELMo. ELMo is less time-consuming, but its performance is not as good as NorBERT's. Another interesting finding was that the pre-trained models perform best with BIO encoding and a bit worse with BIOUL.

7 Future work

This section outlines a number of potential directions for this project that were not included in the original scope.

7.1 Error analysis

The NorBERT model obtained almost the same result with and without including the character-level information. The performance did not differ too much when the NorBERT model was trained with the BIOUL encoding. In total, these factors did not impact the result of the model at all, which seems a bit strange. Therefore, an error analysis would be interesting in this case, which could help explore the decisions of the classifier in more detail and try to find out why the classifier is obtaining the same result for different encoding types and with/without including character-level information. More specifically, these questions would be interesting

(1) Is there any type of sentences in the original dataset that NorBERT cannot correctly classify? What kinds of sentences does it struggle with?

(2) How does the NorBERT handle such sentences with BIOUL encoding?

7.2 Character-level information

The addition of character information did not give us better results for all the architectures in our implementation. The next study could use character features in the same way that previous research has shown to improve model performance. One example is Hicham El Boukkouri (2020), where he used the character feature by firstly applying two non-linear Highway layers, which are basically relatively deep feed-forward neural networks, before projecting to embedding and processing with CNN. It is interesting to see if the addition of character features would have the same improvement for TSA.

7.3 Other types of encoding

Due to the limited time, we did not implement the performance of more different encoding schemes on all of the different architectures. In this paper, we only experimented with the encoding types of BIO and BIOUL, and obtained some interesting results, especially with BIO. Other interesting encoding types like IO, BIOU, or BIOL are also worth experimenting with and could provide a much more comprehensive understanding of the effects of different encoding schemes.

References

Tobias Langø Aasmoe. 2019. Named entity recognition for norwegian - experiments on the norne corpus.

Nasser Alshammari and Saad Alanazi. 2021. The impact of using different annotation schemes on named entity recognition. *Egyptian Informatics Journal*, 22(3):295–302.

Caiming Xiong Richard Socher Bryan McCann, James Bradbury. 2017. Learned in translation: Contextualized word vectors.

Huy Nguyen Christopher D. Manning Dan Klein, Joseph Smarr. 2003. Named entity recognition with character-level models.

Kareem Darwish. 2013. Named entity recognition using cross-lingual resources: Arabic as an example. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1558–1567, Sofia, Bulgaria. Association for Computational Linguistics.

Jacob Devlin and Slav Petrov. 2019. Multilingual bert.

ELMo. 2022. Elmo — Wikipedia, the free encyclopedia. [Online; accessed May-2022].

Arzucan Ozgur Hakan Demir. 2014. Improving named entity recognition for morphologically rich languages using word.

Thomas Lavergne Hiroshi Noji Pierre Zweigenbaum Junichi Tsujii Hicham El Boukkouri, Olivier Ferret. 2020. Characterbert: Reconciling elmo and bert for word-level open-vocabulary representations from characters.

Vinit Ravishankar Erik Velldal Lilja Øvrelid Jeremy Barnes, Andrey Kutuzov. 2021. Proceedings of the third in5550 workshop on neural language processing (wnnlp 2021). page 20.

Jeremy Barnes Lilja Øvrelid, Petter Mæhlum and Erik Velldal. 2020. A fine-grained sentiment dataset for norwegian. pages 5025–5033.

NorBert. 2022. Norbert — Nordic language processing laboratory, the free encyclopedia. [Online; accessed May-2022].

Ido Dagan Oren Melamud, Jacob Goldberger. 2016. context2vec: Learning generic context embedding with bidirectional lstm. pages 51–61.

Egil Rønningstad. 2020. Targeted sentiment analysis for norwegian text.

Yann LeCun Xiang Zhang. 2015. Text understanding from scratch.

David Sontag Alexander M. Rush Yoon Kim, Yacine Jernite. 2015. Character-aware neural language models.

Haiyun Peng Yukun Ma and Erik Cambria. 2018. Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive lstm.

Haiyun Peng Yukun Ma and Erik Cambria. 2019. Applying deep learning approach to targeted aspect-based sentiment analysis for restaurant domain. pages 206–211.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2016. Gated neural networks for targeted sentiment analysis.

A Appendix

A.1 Parameters for models in table [5]

- GRU model:
  - NUM_LAYERS=1
  - HIDDEN_DIM=500
  - WORD_DROPOUT=0.6
  - RNN_DROPOUT=0.2
  - EPOCHS=30
  - BIDIRECTIONAL=False
- LSTM model:
  - NUM_LAYERS=1
  - HIDDEN_DIM=500
  - WORD_DROPOUT=0.6
  - RNN_DROPOUT=0.2
  - EPOCHS=30
  - BIDIRECTIONAL=False
- ELMo:
  - CLASSIFIER =’gru’
  - DROPOUT=0.5
  - EPOCHS=20
  - HIDDEN_DIM=300,
  - NUM_LAYERS=3
- BERT:
  - EPOCHS=50
  - FREEZE=False,
  - NUM_LAYERS=1
  - HIDDEN_DIM=100

A.2 Parameters for models in table [6]

- GRU model:
  - NUM_LAYERS=2
  - CHAR_EMBEDDING_DIM=30
  - CHAR_HIDDEN_DIM=50
  - WORD_DROPOUT=0.5
  - RNN_DROPOUT=0.2
  - EPOCHS=30
  - BIDIRECTIONAL=True
- LSTM model:
  - NUM_LAYERS=2
  - CHAR_EMBEDDING_DIM=30
  - CHAR_HIDDEN_DIM=50
  - WORD_DROPOUT=0.5
  - RNN_DROPOUT=0.2
  - EPOCHS=30
  - BIDIRECTIONAL=True
- BERT:
  - EPOCHS=50
  - FREEZE=False,
  - NUM_LAYERS=1
  - HIDDEN_DIM=100

A.3 Parameters for models in table [7]

- ELMo:
  - CLASSIFIER =’gru’
  - DROPOUT=0.5
  - EPOCHS=20
  - HIDDEN_DIM=300,
  - NUM_LAYERS=3
- BERT:
  - EPOCHS=50
  - FREEZE=False,
  - NUM_LAYERS=1
  - HIDDEN_DIM=100

Forbidden Transitions: Investigating BiLSTM-CRF models for Targeted Sentiment Analysis

Helene Bøsei Olsen and Sigrid Riiser Kristiansen
University of Oslo
{helenbol,sigrirk}@ifi.uio.no

Abstract

Fine-grained Targeted Sentiment Analysis (TSA) is the combined task of both identifying a target of an opinion in a text and its associated polarity. This paper explores the effect of extending a Bidirectional Long Short Term (BiLSTM) model with a CRF layer for a TSA task on Norwegian text. We train LSTM-based models with and without CRF to jointly predict targets and sentiment polarity on multi-domain texts from the NoReC$_{fine}$ dataset. Additionally we investigate the effect of using different pre-trained word embeddings, applying custom word features and manually adding transitions constraints. While manually constraining the the CRF layer indeed keeps the model from predicting forbidden transitions, it does not improve performance scores. However, we confirm that adding a CRF inference layer improves the overall performance for targeted sentiment analysis. We find that there is still room for improvement and outline some ideas for further work along with our conclusion.

1 Introduction

Sentiment analysis has traditionally been approached as a document- or sentence-level classification task, determining the overall sentiment polarity. It is often the case that both negative and positive sentiment is expressed in the same text, but with respect to different targets. The task of fine-grained Targeted Sentiment Analysis(TSA) aims to solve this challenge. To extract opinions towards a target, TSA can be seen as two tasks: (1) identifying the target and (2) identifying the sentiment polarity towards the target. The complexity of the task can be illustrated with a sentence having a negative polarity towards one target while at the same time having a positive polarity towards another. An example from the NoReC$_{fine}$ dataset:

Jakob Oftebro storspiller i en svært ujevn film

B-POS	I-POS	O	O	O
Jakob	Oftebro	storspiller	i	en

O	O	B-NEG
svært	ujevn	film

Figure 1: A tagged sentence from the NoReC$_{fine}$ dataset.

«Jakob Oftebro gives a brilliant performance in a quite uneven movie ». This sentence contains both a positive and a negative target—the actor and the movie, respectively. In a a sentence like this, it is crucial to differentiate positive and negative targets. The associated BIO-tag sequence is shown in Figure 1.

This example depicts how the fine-grained analysis is a complex task and not easily solvable.

In this work we investigate how a Conditional Random Field inference layer affects a BiLSTM model in solving Targeted Sentiment Analysis for Norwegian text. We will first describe some of the relevant previous work regarding TSA. What follows is a description of the NoReC$_{fine}$ dataset and challenges it entails. Section four details the architecture for LSTM, BiLSTM and CRF, and is followed by the experiments section. We describe our experiments on parameter tuning for the specified models, pre-trained word embeddings, custom features, as well as describing the evaluation metrics. In the final part of this work we discuss the results of the experiments and conduct an error analysis.

2 Related Work

Targeted Sentiment Analysis is frequently referred to as open-domain targeted sentiment analysis (Mitchell et al., 2013; Luo et al., 2022). A related task is aspect-based sentiment analysis, which differ from TSA in that it is usually based predefined aspects within one domain (Zhang et al., 2015).

TSA is often approached as a sequence labeling task (Huang et al., 2015; Lample et al., 2016), consisting of two subtasks: extracting the target as a Named Entity Recognition (NER) task and classifying the polarity directed at the target as a Sentiment Analysis task.

Some early efforts to solve the task of TSA used statistical methods such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF) (Yang and Cardie, 2013). Mitchell et al. (2013) exploited the linguistically informed features within a CRF to outperform a strong baseline on Open Domain Targeted Analysis in Spanish and English. Following efforts demonstrated that using neural models improved the performance for the task. Zhang et al. (2015) proposed a model extending the CRF baseline from Mitchell et al. (2013) with a LSTM model leading to significantly higher results compared to the baselines. Lample et al. (2016) increased the performance of the LSTM model for Named Entity tagging by adding a CRF layer as the inference layer. This combination became dominant in sequence labeling tasks (Huang et al., 2015) and resulting in state of the art performance for Yang et al. (2018).

The task has been approached with both a pipeline method (Mitchell et al., 2013; Zhang et al., 2015) by using separate models for the two subtasks, and a joint method (Yang and Cardie, 2013), where a single model is jointly performing the target extraction and target polarity classification. (Hu et al., 2019) points out disadvantages of a sequential tagging approach and explores a span-based approach as an alternative.

Recently the use of pre-trained language models such as BERT (Devlin et al., 2019) has become the mainstream approach for many NLP tasks, including sentiment analysis.

The current state-of-the-art results for aspect-based sentiment analysis is based on a pre-traind BERT model (Luo et al., 2020). However, most recent work has been done on rather limited, domain-specific datasets. Luo et al. (2020), Hu et al. (2019) and Li et al. (2019), among others, use datasets from three separate domains: restaurant reviews, tweets, and laptop reviews. Even if a domain-specific model performs well in its respective domain, it might still achieve weak results in more generalized tasks. This is because the vocabulary might be domain specific and not transferable to other domains, e.g. an adjective in the restaurant domain can have a different polarity in the laptop domain. As the challenges vary between domains and languages, it does not make sense to directly compare scores between our work and other related work. Therefore, in this paper, we base our work on a BiLSTM baseline provided by the IN5550 course at the University of Oslo, and report the scores in comparison to this baseline.

Train	dev	test	total
8634	1531	1272	11437

Table 1: number of sentences across data splits.

3 Dataset

For our experiments, we use a modified version of NoReC$_{fine}$, a dataset for fine-grained sentiment analysis for Norwegian (Øvrelid et al., 2020). NoReC$_{fine}$ consists of sentences from The Norwegian Review corpus (Velldal et al., 2017), a corpus of reviews from different news sources and over different domains such as literature, music, sports and movies. The texts are split into sentences and annotated with BIO-tags and polarities for the targets for each entity. This results in five possible labels: B-targ-positive, I-targ-positive, B-targ-negative, I-targ-negative and O. In this paper they are occasionally referred to as B-POS, I-POS, B-NEG, I-NEG and O. The original NoReC$_{fine}$ dataset is annotated with respect to polar expressions, targets and holders of opinion, while the modified dataset used in this paper only contains information about the polarity and identification of the targets.

The modified NoReC$_{fine}$ dataset consists of 11437 sentences and 6656 targets. Figure 2 shows the unbalanced nature of these types of datasets, where the majority of the tokens are O, outside an entity. Another important aspect of the dataset is the distribution between the positive and negative targets. What can be seen in Table 2 is that positive targets occur almost twice as often as negative targets.

Although Øvrelid et al. (2020) state that «the targets should be as short as possible while preserving important information», they also note that target identification is not always a straightforward process. Furthermore, they report that whereas annotators tend to agree about central elements of the expressions, there is significantly less agreement about target spans.

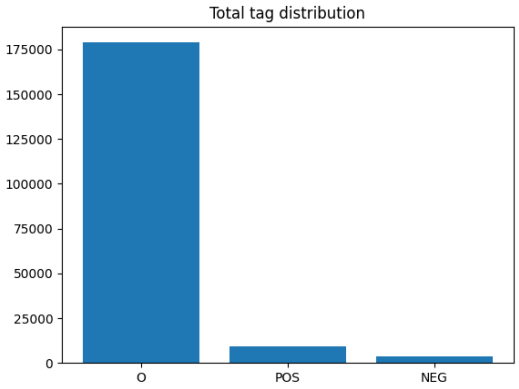


Figure 2: Tag distribution over the dataset

Label	Count
O	11434
B-targ-Positive	4631
I-targ-Positive	4552
B-targ-Negative	2024
I-targ-Negative	1879

Table 2: Class label count for the whole dataset.

The sentences in the NoReC$_{fine}$ dataset contains target spans of various length. Although the average target length is 1.9 tokens, there are occurrences of longer targets consisting of up to 35 tokens.

An example of this is

> Og **tekstlinjen ” Well I ’ ve been alone / and I ’ d take you home / but my bedroom smells like cum ” fra ” Down The Lane ”** rager lett der oppe blant årets morsomste.

(And the lyric "Well I’ve been alone / and I’d take you home / but my bedroom smells like cum" from "Down The Lane" easily stands tall among the funniest of the year.) where the (positive) target spanning 24 tokens is shown in boldface.

This sentence is a good representation of two challenging aspects of the task at hand: the length of the target span and the occurrences of tokens in other languages.

4 Architecture

In this section we provide a brief description of the models and their architecture used in this paper; LSTM, BiLSTM, BiLSTM-CRF.

4.1 BiLSTM

Recurrent Neural Networks (RNN) are neural networks with recurrent properties which makes the output directly or indirectly dependent on earlier outputs. Long Short-Term memory network (LSTM) is a type of RNN which can predict the output based on long distance features by adding a context layer and gates to control the flow of information (Hochreiter and Schmidhuber, 1997).

A bidirectional LSTM (BiLSTM) adds another LSTM layer which processes the input in the opposite direction, from the last word to the first. By combining the output from both layers, we obtain information about both previous and following states. This information is especially useful for a task such as TSA, because the words carrying sentiment information can be on either side of the target.

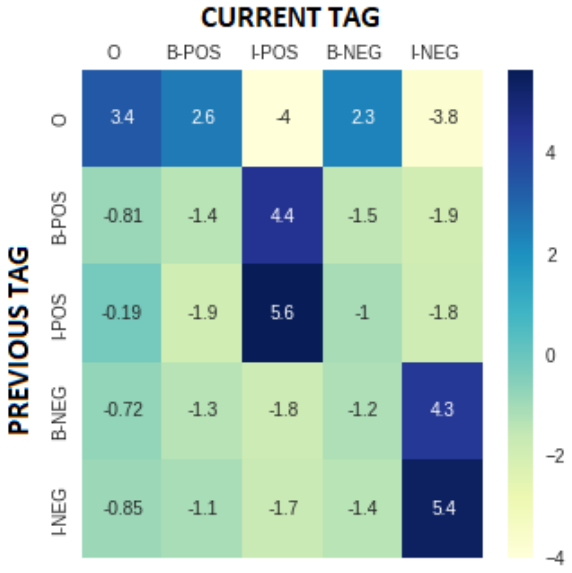


Figure 3: CRF transition matrix showing transition probabilities from one tag to another.

4.2 Conditional Random Field

While the BiLSTM output carries sequential information in both directions, the predicted output tags are conditionally independent given the LSTM output states. As a result of adding a Conditional Random Field (CRF) layer, we can infer the labels by computing the joint probability of the tag sequence. Consequently, it can take a wider context into account than e.g. Hidden Markov Models (HMMs) which rely on the assumption that the current state only depends on the previous state. CRFs were proposed by Lafferty et al. (2001) as a

means to avoid the label bias problem of Maximum Entropy Markov Models (MEMMS), described in more detail by Lafferty et al. (2001). CRF layers have repeatedly been shown to improve the results of sequence tagging tasks (Lample et al., 2016). Following the work of Huang et al. (2015), we explore the effect of adding a CRF layer to the BiLSTM baseline model and report the following results.

The score for a specific tag sequence given a sequence of words can be calculated as:

$$S(\mathbf{y}|\mathbf{x}) = \sum_{t=0}^{n} A_{y_{t-1},y_t} + \sum_{t=0}^{n} U_{y_t,x_t}$$

where A is the transition matrix with probability scores from transitions y_{t-1} to y_t and U the emission matrix, which gives the probabilities for for label y_t given token x_t. In our model, the CRF layer's learned parameters is the transition matrix A. The emission scores are given as the output from the BiLSTM, with the shape $n \times m$ where m is the number of possible tags. An example of a learned transition matrix from our models is shown in Figure 3.

The conditional probability is given as:

$$P(\mathbf{y}|\mathbf{x}) = \frac{e^{S(\mathbf{y}|\mathbf{x})}}{\sum y' e^{S(\mathbf{y}'|\mathbf{x})}}$$

where $\mathbf{y}'$ denotes all possible sequences of tags.

The most likely sequence,

$$\mathbf{y}^* = argmax_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$$

is computed using a Viterbi decoding algorithm. We then use negative log likelihood to compute the loss.

Note that with *possible tags*, we also include tag sequences that are forbidden in a BIO tagging scheme. For example, a transition from `B-targ-Positive` to `I-targ-Negative` will never be seen in the gold standard. Ideally, the model should learn through training that these transitions are close to impossible. However, the parameters are randomly initialized, and with little training data, these restrictions might not be captured. Lester et al. (2020) shows that a CRF layer can be improved by manually placing restrictions on the output. We include this as one of our experiments, where we manually initializing each forbidden transition in the transition matrix to a very negative number.

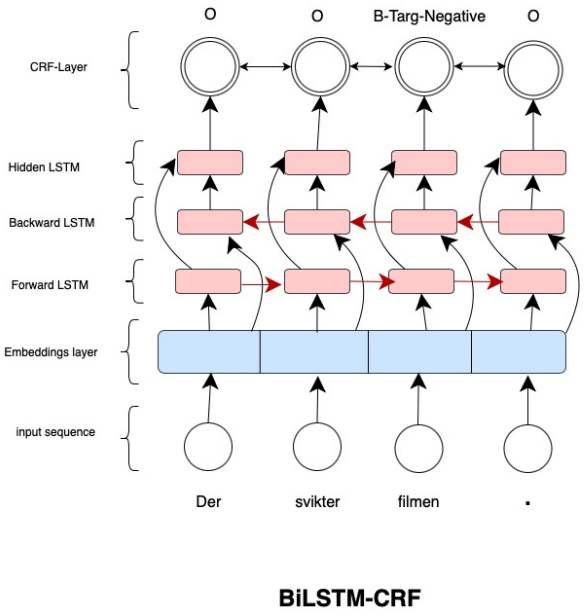


Figure 4: BiLSTM CRF model inspired by Lample et al. (2016).

5 Experiments

In this section we describe the modelling, set-up, and evaluation metrics for our experiments. The first section describes the baseline model and the BiLSTM-CRF model. Following is a description of the experimental set up and the hyper parameters we are tuning for the different models. Moveover, we describe the experiments with different pre-trained static embedding models, custom word features, and manual initialization of model parameters.

5.1 Baseline BiLSTM

We use a simple Bidirectional Long Short Term Memory (BiLSTM) tagger as a baseline model. For word representation we use a pre-trained static embedding model for Norwegian words, taken from the NLPL repository. The BiLSTM is appropriate for this task as TSA can be approached as a sequence to sequence problem. The forward and backward states can be effective for utilizing both future and past input features. This makes a BiLSTM model able to represent global sequence information.

5.2 BiLSTM-CRF

We combine a Bidirectional LSTM and a CRF to form a BiLSTM-CRF model, which is shown in Figure 4. We used the framework for CRF avail-

able on GitHub [1]. The model takes a sentence as a sequence as input, which is represented as embedding vectors with a 100 dimension. The vector representations are then used as input to the BiLSTM. In this model, the LSTM layers functions as a filter to include only the important features from the input. The hidden states from the forward pass and the backward pass are then concatenated at each word and contains global information about the whole sequence. The CRF layer deals with the sequential data by adding conditionality to the sequence so that the restrictions of the BIO-tag system is not violated. e.g. a B-targ-Positive tag is never followed by an I-targ-Negative tag.

5.3 Pretrained word embeddings

We experiment with different types of word embeddings as feature representations. We are using pre-trained embeddings from the Nordic Language Processing Laboratory (NLPL) repository [2]. Several studies have revealed that pre-trained word embeddings layer improve performance on tasks involving sequence taggers (Collobert et al., 2011; Yang et al., 2018). The word embeddings represent each word in the dataset and are used as initialization for the first layer in the BiLSTM-model.

We are experimenting on the following pre-trained embeddings:

- ID58 **Norwegian-Bokmaal CoNLL17 corpus** is based on the Word2Vec Continuous Skipgram algorithm, vocabulary size = 1182371 and lemmatization = False
- ID124 **NoWaC**, algorithm: fastText Skipgram, Vocabulary size: 1356633 and lemmatization = False.
- ID106 **NoWaC**, algorithm: Gensim Continuous Skipgram, Vocabulary size: 1356632, lemmatization = False.

The chosen pre-trained embeddings represent three different algorithms: Word2Vec Continous Skipgram, fastText Skipgram and Gensim Continuous Skipgram. We are not making use of lemmatized words as it might ignore syntactic nuances and semantic information about sentiment (Camacho-Collados and Pilehvar, 2018).

[1] https://github.com/kmkurn/pytorch-crf.git

[2] http://vectors.nlpl.eu/repository/

Spelling features
Starts with capital letter
All capital letters
All lowercase
Non-initial capital letters
Combination of letters and digits
Including non-alphanumeric characters
Alphanumeric characters only
Numeric characters only
Text preprocessing
Change all numbers to 0

Table 3: Features and preprocessing.

5.4 Features

Following Huang et al. (2015) we extract one-hot feature representations for each word and concatenating them with its embedding vector. The features are listed in Table 3.

5.5 Experimental setup

The baseline BiLSTM model is trained with the hyperparameters shown in Table 5. We train a LSTM, LSTM-CRF, BiLSTM, and BiLSTM-CRF model with [1, 2, 3] hidden layers, batch-size [5, 64, 128], learning rate [0.1, 0.001] and hidden dimension[100, 200, 300]. We are using Exponential LR scheduler with gamma = 0.9 and Adam as optimizer for all experiments. All experiments has a dropout set to 0.01 after the embeddings layer and a dropout set to 0.2 in the LSTM.

Next, we are investigating the effect of different pre-trained embedding layers on the best model from the hyper-parameter tuning. We are using the three pre-trained embeddings announced in the previous section. The embeddings with the best results from this experiment is used throughout the remaining experiments.

5.6 Evaluation

We evaluate the results of the model using proportional and binary overlap measure F1 as proposed for this specific task by Øvrelid et al. (2020).

The F1 score is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

$$= \frac{TP}{2 * TP + FP + FN}$$

Where TP = true positive, FP = False positive and FN = False negative.

Hyper parameters	LSTM	LSTM-CRF	BiLSTM	BiLSTM-CRF
No. layers	3	3	2	2
Hidden dim	200	200	100	100
Batch size	5	5	5	5
Initial learning rate	0.001	0.001	0.001	0.001
Epochs	7	7	4	6
Patience	5	5	2	4
Binary F1	0.431	0.441	0.473	0.533
Propotional F1	0.218	0.264	0.322	0.356

Table 4: Results from tuning hyperparameters for LSTM, LSTM-CRF, BiLSTM and BiLSTM-CRF.

	BiLSTM	BiLSTM-CRF
Optimizer	Adam	Adam
No. Layers	1	2
Hidden dim	100	100
LR	0.01	0.001
Batch size	50	5
Dropout	0.01	0.01
Embeddings	ID 58	Id 124

Table 5: Hyperparameters for Baseline BiLSTM model and the best BiLSTM-CRF model.

The Binary F1 score measures overlapping spans of prediction and gold labels as correct. The proportional score is a more strict measure as it measures token level F1 by assigning precision as the ratio of overlap with the predicted span and recall as the ratio of overlap with the gold span.

In this paper, we report both binary and proportional F1 scores, but the latter is more influential when comparing performance.

6 Results

6.1 Comparing models

Table 4 compares the best models results from the tuning of the hyperparameters. A closer inspection of the table shows that using a CRF increases the performance of the models and that the BiLSTM-CRF returns the best result. Interestingly, the bidirectional models require fewer parameters compared to the LSTM. Both the LSTM and the LSTM-CRF performs the best with three layers and 200 hidden dimensions, while the Bidirectional models has two layers and 100 hidden dimensions.

6.2 The impact of pre-trained word embeddings

As opposed to the findings of Huang et al. (2015), we can see from the results in Table 6 that the choice of embeddings makes a significant difference on the performance. The pre-trained embedding with ID 124 yields a result with seven percent points better proportional F1 score than the second best. This is consistent with the work of Rønningstad (2020), which used fastText for all their experiments with LSTM-models. Based on these results, we use the fastText pretrained embeddings (ID 124) in all further experiments.

BiLSTM-CRF	Binary F1	Prop. F1
ID 58	0.462	0.281
ID 106	0.465	0.286
ID 124	0.533	0.356

Table 6: Results from experimenting with different pretrained embeddings.

6.3 Adding custom features

The results from adding custom features to our best models are seen in Table 8. It appears that these features do not improve the model, which is in accordance with the work of Huang et al. (2015), concluding that LSTM-based models are less affected by engineered features due to their robustness.

6.4 Enforcing CRF restrictions

The results of manually initializing CRF parameters to reflect the restrictions in the BIO scheme is exemplified in Table 7. Clearly, this works as intended. On the other hand, this version of the model achieved a lower proportional F1 score than our best model.

7 Error analysis

In this section, we will examine our models' predictions in more detail. Although our best BiLSTM-CRF model achieves higher evaluation scores, its prediction includes a slightly higher number of

	BiLSTM	BiLSTM-CRF	BiLSTM-CRF with restrictions
I without preceding B	57	69	0
Mix-ups of NEG and POS	19	19	0

Table 7: Comparison of tag sequence errors in model predictions on the test set. Mix-ups refer to targets with mixed polarities, e.g B-POS followed by I-NEG.

	Features	No features
BiLSTM	0.299	0.301
BiLSTM-CRF	0.334	0.356

Table 8: Resulting proportional F1-scores from adding custom features to the best models.

	B	B-CRF	G
Oppløsningen	B-POS	B-POS	O
På	O	I-POS	O
218	O	I-POS	O
x	O	I-POS	O
218	O	I-POS	O
piksler	O	I-POS	O

Table 9: Excerpt from the dataset, with predictions by BiLSTM, BiLSTM-CRF with gold annotation to the right.

«illegal» tag sequences than the BiLSTM model without CRF.

As seen in Table 7, the CRF inference layer makes several target predictions using Inside tags from the BIO scheme without a preceding Beginning-tag. This is surprising, as one of our main expectations from the CRF layer was that it would learn these restrictions. It is evident from the results that both models make such errors, and they do not usually overlap.

Regardless, the BiLSTM-CRF model is significantly better than the other models at capturing targets with longer spans. The longer targets might be better captured with the BiLSTM-CRF since we have two directions and because the CRF-layer allows us to classify tags by predicting them together based on the dependence of the input tags. Despite the fact that some of the long target sequences predicted by the BiLSTM-CRF model are not present in the gold standard, this nevertheless shows the model's ability to identify targets with longer spans. These long target predictions also correctly adhere to the BIO scheme. Additionally, the BiLSTM-CRF model repeatedly exceeds the other models in capturing full targets with quotation marks, such as song titles, such as «Summer In My Hometown».

The differences between the predictions might even reflect the differences in annotation, as the example shown in Table 9. The model's prediction is not a target in the gold standard, but might resemble other target patterns. The difference between the predictions mirrors a typical variation seen in the dataset, where phrases are sometimes annotated as a single target, and other times not.

7.1 Cross-domain data

As described in 3, The NoReC$_{fine}$ dataset covers multiple domains, from restaurant reviews to game reviews. This makes the task of TSA even more challenging, since some words might have a positive polarity in one domain and negative or irrelevant polarity in another. One example is the adjective «trygt» (safe), which is a word typically associated with something positive in domains such as restaurant reviews or reviews for tools. In the training data we can find the sentence «Trygt og godt album fra The New Pornographers» (in English: «Safe and good album from The New Pornographers»), which is annotated as B-targ-Negative. This example depicts how the adjective «safe» might have negative connotations in the music domain, but positive in other.

8 Conclusion and future work

In this work, we show that extending a BiLSTM-baseline with a CRF inference layer improves the performance of a TSA task. We observe a noticeable improvement in performance for both LSTM and BiLSTM when using CRF. The CRF layer did not learn tagging scheme restrictions as well as we had expected. However, we see that these restrictions can be enforced by manually initializing the parameters.

This paper did not attempt to improve the model with state-of-the-art transformer-based technologies. While our intention was to explore the effect of other methods, it would have been useful to include a transformer-based model as a means of comparison. There are additional methods that can be explored to further improve the existing

BiLSTM-CRF model. We could follow Chiu and Nichols (2016) and Lample et al. (2016) in including character-level information, by character embeddings or Convolutional Neural Networks.

We have not devoted effort in our work looking at negation and speculation. Information about negation is highly important for sentiment analysis, and our models makes several mistakes regarding this phenomena. Including multitask learning of negation has shown to make TSA models more robust, and could be included in feature work on TSA (Moore and Barnes, 2021).

Another interesting, but not surprising, phenomenon from the results is that the models sometimes struggle with identifying targets given as pronouns. A typical challenge is when the pronoun «it» is a co-reference to a previous sentence in the document. In the NoReC$_{fine}$ dataset we can find this example of the phenomenon: «Det morsomme er at denne ikke har et eneste dødpunkt», where «denne» (*it*) is annotated B-targ-positive. It might have been useful for the model to understand what «denne» (*it*) refers to by including access to the previous sentence. One solution worth exploring is looking at the whole document instead of just one sentence as proposed by Luo et al. (2022).

References

Jose Camacho-Collados and Mohammad Taher Pilehvar. 2018. On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis.

Jason P. C. Chiu and Eric Nichols. 2016. Named Entity Recognition with Bidirectional LSTM-CNNs.

Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (almost) from Scratch.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. *Neural computation*, 9:1735–80.

Minghao Hu, Yuxing Peng, Zhen Huang, Dongsheng Li, and Yiwei Lv. 2019. Open-Domain Targeted Sentiment Analysis via Span-Based Extraction and Classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 537–546, Florence, Italy. Association for Computational Linguistics.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging.

John Lafferty, Andrew Mccallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. *arXiv:1603.01360 [cs]*.

Brian Lester, Daniel Pressel, Amy Hemmeter, Sagnik Ray Choudhury, and Srinivas Bangalore. 2020. Constrained Decoding for Computationally Efficient Named Entity Recognition Taggers.

Xin Li, Lidong Bing, Piji Li, and Wai Lam. 2019. A Unified Model for Opinion Target Extraction and Target Sentiment Prediction.

Huaishao Luo, Lei Ji, Tianrui Li, Nan Duan, and Daxin Jiang. 2020. GRACE: Gradient Harmonized and Cascaded Labeling for Aspect-based Sentiment Analysis.

Yun Luo, Hongjie Cai, Linyi Yang, Yanxia Qin, Rui Xia, and Yue Zhang. 2022. Challenges for Open-domain Targeted Sentiment Analysis. *arXiv:2204.06893 [cs]*.

Margaret Mitchell, Jacqui Aguilar, Theresa Wilson, and Benjamin Van Durme. 2013. Open Domain Targeted Sentiment. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1654, Seattle, Washington, USA. Association for Computational Linguistics.

Andrew Moore and Jeremy Barnes. 2021. Multi-task Learning of Negation and Speculation for Targeted Sentiment Classification. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2838–2869, Online. Association for Computational Linguistics.

Lilja Øvrelid, Petter Mæhlum, Jeremy Barnes, and Erik Velldal. 2020. A Fine-grained Sentiment Dataset for Norwegian. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 5025–5033, Marseille, France. European Language Resources Association.

Egil Rønningstad. 2020. Targeted Sentiment Analysis for Norwegian text. Master's thesis, University of Oslo.

Erik Velldal, Lilja Øvrelid, Eivind Alexander Bergem, Cathrine Stadsnes, Samia Touileb, and Fredrik Jørgensen. 2017. NoReC: The Norwegian Review Corpus. *arXiv:1710.05370 [cs]*.

Bishan Yang and Claire Cardie. 2013. Joint Inference for Fine-grained Opinion Extraction. page 10.

Jie Yang, Shuailong Liang, and Yue Zhang. 2018. Design Challenges and Misconceptions in Neural Sequence Labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3879–3889, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Meishan Zhang, Yue Zhang, and Duy-Tin Vo. 2015. Neural Networks for Open Domain Targeted Sentiment. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 612–621, Lisbon, Portugal. Association for Computational Linguistics.

Neural Machine Translation for Restaurant Reviews

William Ho
University of Oslo
willh@uio.no

Abstract

In this paper, we consider the problem of neural machine translation. In particular, due to the re-opening of the world and people starting to travel more, we tackle the problem of translating restaurant reviews. However, as there is no publicly available dataset for this application, we attempt to train our model by transferring knowledge from other domains. Furthermore, we show that combining datasets from different domains helps our model generalize better. To evaluate our restaurant review model, we built a test dataset from scratch with custom translated bilingual pairs, and achieve a BLEU-score of 35.60. We also demonstrate that, for our application, the gains of having a more complex model out-weights the typical downsides. Lastly, we raise some concerns related to deploying a translation system without properly assuring the quality of the generated sentences.

1 Introduction

Neural machine translation (NMT) is an important task that aims to automatically translate sentences between different languages using neural methods. Today, NMT is one of the most researched subfields of NLP, and it has proven to yield great results in practice (Wu et al., 2016), giving us easy access to translations of high quality between multiple languages.

With the world slowly re-opening, traveling and exploring new countries is becoming a higher priority for many, and trying out new dining options is a natural part of the traveling experience. With Norway's low population and generally high income, the dining options are rather limited while also being on the high-end for many tourists compared to other countries, leading to uncertainty of whether or not a restaurant is worth the visit. Thankfully, many enthusiasts leave online reviews of their dining experiences, helping to reduce this uncertainty. However, in most cases, these reviews are written in the local language of the restaurants, making it hard for tourists to make use of the available information. To this end, we experiment with ways to tackle these problems using NMT methods, which could result in practical applications.

In this paper, we experiment with NMT methods for the practical application of translating restaurant reviews from Norwegian to English to better help tourists gauge whether a particular Norwegian restaurant is worth the visit or not.

As practicality is a key part of such an application, there are several factors we have to consider when building a system. Firstly, a translated sentence should be of good quality, meaning it is understandable after being translated. Secondly, it is important that the sentiment of a translated sentence is in tact, meaning that a positive Norwegian review should also be positive when translated to English, and the same goes with negative reviews. Lastly, the system itself should run rather efficiently, as having to wait makes the application less desirable. However, there is a theoretical trade-off between the two first factors and the last factor, as a network with more parameters typically has a better ability to fit the training data giving us better performance, while a model with less parameters naturally runs faster.

We show that this trade-off is mostly irrelevant for our application by comparing several models of different complexity on the task at hand. We present the resulting models and their performances by evaluating them both quantitatively, using the BLEU evaluation metric designed for language generation, and qualitatively, looking at specific example outputs.

Furthermore, what makes this task challenging, is that there is no publicly available Norwegian-English parallel corpus that specifically contains language related to restaurant reviews. Therefore,

we train our models using several datasets from different domains and show that the models generalize quite well to our application of interest when using a varied dataset. We also build our own custom bilingual corpus by extracting online restaurant reviews written in Norwegian and manually translating them to English. This corpus is then treated as a held-out test set to evaluate our final model.

The rest of the paper is organized as follows. In section 2, we describe the task at hand in more detail and present the methods used in the paper. In section 3, we describe the experiments conducted, followed by the results in section 4. Finally, we discuss the results in section 5, and conclude our work in section 6.

2 Method

2.1 Problem description

Given an input sequence $x = (x_1, x_2, .., x_m)$ of m Norwegian words, we want to use a neural network f to predict a translated sequence of n English words $y = (y_1, y_2, ..., y_n)$, where each x and y are in a set of k pre-defined classes $C = \{c_1, c_2, ..., c_k\}$ composed of a bilingual vocabulary. Our network f consists of an encoder e and decoder d, and a final linear classifier g. Firstly, we produce a set of intermediate representations h_e, also known as memory, by feeding x into e such that $h_e = e(x)$. To generate a translated sequence $\hat{y}$, the memory is fed into the decoder, together with an additional input being the previous word of the generated sequence $\hat{y}_i$, which are then finally fed into the classifier g, $\hat{y}_{i+1} = g(d(h_e, \hat{y}_i))$ for $i \in (2, n-1)$, resulting in $\hat{y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_n)$. The first input will always be a special start-of-sentence token.

We should note that the decoding is different in the training phase and the inference phase. During training, the input to the decoder is simply the previous ground-truth label y_i, and the length of the output sequence of the decoder is only as long as the number of ground-truth labels. In inference, we do not have access to ground-truth labels. Therefore, we can for example employ a greedy-searching method that will simply feed the previously generated word of class c, $\hat{y}_{i+1} = \arg\max_c g(d(h_e, \hat{y}_i))$ as the input for the next timestep until the final special end-of-sentence token is generated. However, there is no guarantee that we will produce this token. Thus, we limit the output translation to a set length.

To train our network f, we observe that each generated word is in one of C pre-defined classes, and we can therefore optimize our network using a Cross-entropy loss-function together with back-propegation and gradient descent.

2.2 Data

As mentioned in the introduction, there is no publicly available Norwegian-English parallel corpus for our task at hand. Thus, we make use of four different Norwegian-English bilingual parallel corpora. We develop our models using a government corpus and a subtitles corpus. Then, to evaluate which model generalizes best, we test the models on a book corpus, and the best performing model is then evaluated on a held-out test DIY restaurant review corpus that we built ourselves. A summary of the datasets is found in table 1.

2.2.1 Government Corpus

The government corpus comprises two subcopora, the Bilingual English-Norwegian parallel corpus from the Office of the Auditor General website, and the Public Bokmål-English Parallel corpus .[1] [2] These corpora are expected to be of high quality, as they are professionally translated.

As this dataset is intended for the government domain, we can expect the language to be rather formal, an example being: *"Riksrevisjonen har i undersøkelsen vist at det er store variasjoner i forvaltningsenhetenes bemanning per innbygger i det enkelte fylke."* and the English *"In the audit, the OAG demonstrated that there are wide variations in the administration units' per capita staffing in the individual county."*. More interestingly, formal government text tend to contain complex and long words, as we can see from the example with *"forvaltningsenhetenes"*. We can also calculate from table 1 that the average number of characters per word for this dataset is larger than the others. Furthermore, this dataset seems to contain more statements than opinions, which may not be ideal for our application of opinion based restaurant reviews. All of this may cause our model to learn more formal, complex, and longer sentences, which are typically not used much by common people.

[1] https://data.europa.eu/data/datasets/elrc_1061

[2] https://www.nb.no/sprakbanken/en/resource-catalogue/oai-clarino-uib-no-parallel-nob/

Dataset	Train	Dev	Test	Avg. Nor (word/char)	Avg. Eng (word/char)
Government	50,000	2,500	-	13.85 / 95.25	16.81 / 106.73
Subtitles	250,000	2,500	-	6.16 / 32.61	6.76 / 35.93
Book	-	2,500	-	13.53 / 73.74	13.83 / 73.49
DIY	-	-	102	9.83 / 57.84	10.44 / 58.63

Table 1: Summary of datasets. The Avg. Nor/Eng (word/char) columns denote the average number of words / average number of characters respectively. We get the average number of characters per word by dividing the latter with the former.

2.2.2 Subtitles Corpus

The subtitle corpus consists of unofficial subtitles for movies and TV series, and unlike the professionally translated government corpus, this corpus is built by mapping translations made by internet users using movie timestamps.[3] Therefore, this dataset has no guarantee of quality, and may also bring with it some unforeseen risks.

Firstly, the translations may be wrong due to intentional and/or unintentional errors as there is no quality assurance. For instance, the training set contains a sample translating *"Don't go"* to *"**L**kke gå"* which is clearly an error. Secondly, how a sentence is translated depends on how a user interprets the original sentence, which may again lead to translations of low quality or very customized translations. Some examples of this include *"Shit, shit."* simplified to *"Faen."*, and *"That's what Esben was doing."* being expanded to *"På en måte har jeg nok inntrykk av at Esben gjorde det."*. The second example more noticeably loses its semantic meaning, going from an assertive statement to a a less certain one. Lastly, the timestamps may not match up correctly, which may lead to either completely wrong translations or as the example above with Esben, where several sentences are combined into one long sentence. All of these factors may lead to our models learning wrong translations or some unwanted bias towards certain words or phrases.

Looking more generally on this corpus, it is clearly less formal than the government corpus and from table 1 the sentences are also shorter. There are many one-liners such as *"Paris!"*, and most of the sentences are parts of dialogues, making them closer to daily speech. However, since we are in the acting domain, some of the sentences are more dramatic and less natural to our daily speech. There are also some special features in the dataset, such as sentences starting with ♩ to indicate music, which may lead to some unwanted biases in our model. Overall, when combining this dataset with the government dataset, we end up with a more diverse and balanced dataset with a wider distribution, and we can expect our model to generalize better to other domains when using both of them rather than just a single one.

2.2.3 Book Corpus

This corpus consists of a translated and aligned version of the book "Hound of the Baskervilles" by Arthur Conan Doyle. [4] As this book is not in the same domain as the two previous datasets, we use it to evaluate the generalizability of our models. Although both this corpus and the subtitles corpus are mostly based on fictional work, there are still several differences between them.

Similarly to the subtitles corpus, this book corpus is also generated by an internet user. However, the quality of the alignments are expected to be of better quality than that of the subtitles corpus, as the translator has previous professional experience within this field. Thus, we would expect less errors within the data.

Much like the subtitles corpus, this corpus also consists of a considerable amount of dialogue. However, in this case, the conversations are presented through a first-person narrator, compared to that of Movies/TV Series where the dialogue is mostly directly exchanged. An example of the first-person narration form is *""De vil vel ikke si at De vet hvor han er?" sa jeg."* with the English translation of *""You don't mean that you know where he is?" said I."*.

Furthermore, it is important to note that the language of a Doyle book is rather old compared to that of the subtitles corpus, as his books dates back a couple of centuries. Therefore, we would expect to find some unusual formulations in the book corpus compared to today's everyday speech. This

[3] https://opus.nlpl.eu/OpenSubtitles-v2018.php

[4] https://farkastranslations.com/bilingual_books.php

is exemplified by *"How did he die?"* having the Norwegian translation of *"Hvordan gikk det for seg?"*, which is a Norwegian phrase that is much less commonly used today.

2.2.4 DIY - Restaurant Review Corpus

As we aim to apply NMT to restaurant reviews, we create our own Norwegian-English parallel corpus to benchmark our final model for the task at hand. The dataset consist of a subsample of original Norwegian internet user generated restaurant reviews found on TripAdvisor, which we then manually translate to English to construct the corpus.[5] As restaurant standards may be different from place to place, the reviews are restricted to only Norway.

To generate a realistic and fair dataset, compared to the training data, for our model to be evaluated on, we cherry-picked all the reviews in our dataset ourselves to keep some reasonable features in tact.

We attempt to keep the average sentence length within the range of both the government and subtitles datasets by including some longer and shorter sentences. We try to avoid too many uncommon food related words, such as specific dishes, but also left some in to keep the dataset realistic. We also try to keep a balance between positive and negative reviews to make sure that the model has actually learned to distinguish between a good and bad restaurant, which is the main purpose of the application.

2.2.5 Tokenizer

We train a customized tokenizer for each of the datasets based on the frequency of words using a simple byte-pair encoding algorithm (Sennrich et al., 2016). We generally try to keep the vocabulary as small as possible, which we discuss further in the next section.

2.3 Architecture

We employ two different architectures for the task of NMT as described in section 2.1. Firstly, we implement the encoder and decoder using RNNs as a baseline, and then we replace the RNNs with transformers.

The RNNs are implemented with GRU cells. The encoder is bidirectional, while the decoder is uni-directional, because we do not want to train the decoder to rely on future information as we simply will not have access to it during inference. As this model is a baseline, we just want to gauge what results we can expect, thus we do not implement any additional mechanisms such as attention.

The transformer model is modified with three additional mechanisms. More specifically, our transformer uses a pre-norm variant (Nguyen and Salazar, 2019), position-infused attention (Press et al., 2021), and the GLU non-linearity (Shazeer, 2020). Again, the encoder is bi-directional while the decoder is uni-directional, which is achieved by masking out the upper triangular attention weight matrix.

In both cases, before being fed into the encoder and decoder, the inputs are projected through an embedding layer to produce embedded word encodings. The embedding layer is of the same size as the vocabulary, meaning it depends on the dataset and the tokenizer used. We note that since we defined our vocabulary to be bilingual in section 2.1, we can use shared bilingual embedding layers.

One important part of our task is to keep the system somewhat efficient, meaning that the model should be kept as simple as possible without degrading the performance by a huge margin. Thus, we try to keep the vocabulary sizes as small as possible, as the embedding layers depend on them. To reduce the amount of trainable parameters further, we use the same embedding layer for both the encoder and decoder to halve the amount of trainable embedding weights. Furthermore, since our embedding layers are of the same size of our vocabulary, we can also share these weights with the classification layer, saving even more parameters.

2.4 Evaluation

Evaluating generated translated text is not trivial, as there can be many different ways to translate a single phrase. Therefore, we will have to rely mostly on our own qualitative analysis of the sentences when evaluating a model. However, this is impractical when training a model, because we would like to automatically evaluate a model quantitatively. For this, we use the BLEU score (Papineni et al., 2002), which mostly correlate with human evaluation. Although the BLEU score ranges from 0 to 100, a sentence with a score above 60 is typically of better quality than what a human would generate, and can be expected from a perfect model, while a score above 30 is usually an understandable to good translation.

[5] https://no.tripadvisor.com/

2.5 Training and Inference

The training and inference is mostly described in section 2.1. Here we add some additional details.

As mentioned in section 2.1, during the training phase we simply feed the ground-truth labels of the previous timestep to the decoder to produce a new prediction which we compare to the current ground-truth with a cross-entropy loss. However, this is not possible during inference as we do not have any labels, meaning we have to feed something else to the model instead. The simplest method is the greedy search method, where we simply feed in the previously predicted word. Another way of doing this is by using beam-search, which searches further into the future before deciding which word to use. Although beam-search typically yields better results, it is also much more expensive. Therefore, we employ the greedy search when training our model, and only use beam-search to evaluate our final models.

3 Experiments

We conduct several experiments aimed towards the goal of generating translations for restaurant reviews of both good quality and high efficiency.

3.1 RNN vs. Transformer

We compare two different sequential models in RNNs and transformers with architectures described in section 2.3 using the combination of the government and the subtitles corpora. The RNN is mainly used as a simpler model to give us an impression of what results we can expect from our datasets. Therefore, we focus mainly on transformers for the remaining experiments.

3.2 Different training datasets

Since we have two training datasets, we train three different models using the transformer architecture on different partitions of the training datasets. Two of the models are trained only on a single dataset each, and the last model is trained on the combination of both datasets. As the evaluation data split, we use the respective development splits, meaning that the combined dataset will have 5,000 evaluation samples. We evaluate each of the models on the book dataset to evaluate which model will hopefully generalize best to our restaurant review dataset.

3.3 Vocabulary size

For each of the corpora, we experiment with different vocabulary sizes as a hyperparamter to see how the sizes affect both the performance and the run time, which is an important trade-off for our application. For the larger combined corpus, we find it natural to select a larger vocabulary size, while keeping the vocabulary size smaller for the two separate corpora.

3.4 Other Hyperparameters

Other than the vocabulary size, there are many hyperparameters for a transformer, but we focus on some that may have the biggest influence on our model in terms of translation quality and time efficiency.

Firstly, we experiment with the hidden sizes, which naturally influences how many parameters our transformers will have, meaning it will affect both the speed and performance. More importantly, we experiment with the number of encoder and decoder layers. This is more important because of the inference phase, where we perform a forward pass for each predicted word, meaning the majority of the run-time is spent on decoding while we only have to run the encoder once. Therefore, we specifically choose to have fewer or equal layers in the decoder compared to the encoder. Specifically, we perform a grid-search over the hidden size of values 100, 150, 200, number of encoder layers of 2,4,6, and decoder layers of 2 and 4.

3.5 Evaluation and final test

After finding some good hyperparameters for the three models from section 3.2, we test how each of them perform on the book dataset to evaluate how well the models may generalize, as well as their runtime. Finally, we choose the best model based on the book dataset to use on our final DIY dataset for a final evaluation of the task at hand.

3.6 Common setup

Each model is trained with a learning rate of 0.0005 using the AdamW optimizer for 20 epochs. We also employ a cosine learning rate scheduler with two hard resets.

4 Results

4.1 RNN vs. transformers

Looking at the results table 2, we can clearly see that the transformer models vastly outperformed

Dataset	Vocab	Hidden	Encoder	Decoder	Eval BLEU	Book BLEU	Book speed
Government	3,000	200	4	4	27.53	2.90	0.034
Government	6,000	200	4	4	27.94	2.92	0.069
Subtitles	3,000	150	4	4	26.45	11.04	0.027
Subtitles	6,000	150	4	4	27.44	11.81	0.026
Combined	5,000	250	6	2	34.64	15.58	0.025
Combined	5,000	250	6	4	35.94	15.85	0.024
Combined (RNN)	5,000	100	2	2	3.26	-	-

Table 2: Some selected results specifically chosen for section 5.2. Vocab and Hidden denote the vocabulary size and hidden size respectively. Encoder and Decoder denotes the amount of layers in each of the networks. Eval BLEU is computed on the respective evaluation dataset. Book speed is the average time to process a single sentence on a CPU measured in seconds per sentence. Note that the book speed includes time spent printing the results.

the RNN based model measured by the BLEU score. Although the goal of this experiment was to get an impression of the results we could expect, we are unable to extract much insight out of these results.

4.2 Different training datasets

In this section, we present the results of the experiments for each individual dataset with their hyperparameters. A selection of key results are presented in table 2.

4.2.1 Government dataset

Although the model trained on the government dataset performed quite well on the government dataset itself with a BLEU scores above 27 for several configurations, it failed to generalize to the book dataset, with a mere 2.9 BLEU.

As discussed in section 2.2.1, the sentences in this dataset are quite long, formal, and contains complicated words, and the model was able to learn these characteristics quite well. With the Norwegian example sentence of *"Undersøkelsen viser at Miljøvern- departementet og Klif i liten grad gjennomfører kontroll for å avdekke ulovlig eksport av farlig avfall."*, the government model with a vocabulary size of 6,000 generated *"The investigation shows that the Ministry of the Environment and the Norwegian Climate and Pollution Agency did not monitor compliance with uncovering illegal export of hazardous waste."*. This is very close to the ground-truth in *"The investigation shows that the Ministry of the Environment and the Norwegian Climate and Pollution Agency only to a very limited extent carry out supervisory activities in order to uncover the illegal export of hazardous waste."*. We see that the semantic meaning of the sentence is very close to perfectly kept, with the only discrepancy being that the generated sentence is more confident that the Agency did not monitor, compared to the ground-truth's limited extent. Additionally, the model also changed the sentence to the past tense with *did not* instead of *carry out*. Or else, the complicated and long nature of the sentence is preserved nicely.

On the other hand, the model's performance on the book dataset was very poor. The Norwegian sentence *"Hun brast i en heftig gråt."* was translated to *"Hungary at a crisis."* is clearly way off the ground-truth *"She broke into passionate sobbing as she spoke."*. This is somewhat expected, as we do not expect a novel to be written in the same formal way as a governmental document.

4.2.2 Subtitles dataset

The subtitles models generalized better than the government models. This model achieved a BLEU of around 27 on its own dataset, while achieving over 11 BLEU for the book dataset. The following results are computed using the subtitles model with a vocabulary size of 6,000.

This dataset comprises of custom translations of Movies/TV Series, meaning we are likely to find a lot of characteristics related to oral speech, some examples being *"I'm goin' with him."* and *"Egon, what are we doing?"*, which the model generated as *"I'm going with him."* and *"What are we doing, Egon?"*. The first example contains the pronunciation of *goin'*, which the model did not learn to catch. Otherwise, the sentence keeps the semantics perfectly. As for the latter example, the semantic meaning is clearly kept, although *Egon* is swapped around. Overall, these are great translations.

It is only natural that the this model outperformed the government model, as a novel and Movies/TV-Series are both entertainment, meaning

the language are more likely to be closer related to each other than with the government data. However, as discussed in 2.2.3, there are still noticeable differences between the book corpus and the subtitles corpus, which we now see has an effect on the performance of the model.

In many cases, the generated sentences would make sense by themselves, such as *"He kept an old boots in the vehicle."*. However, when comparing it to the ground-truth translation *"He held an old black boot in the air."* and the Norwegian sentence *"Han holdt en gammel støvel i veiret."*, the translation seems a bit absurd. We also notice some grammatical errors such as *"an old boots"* in the generated sentence. A much better translation is *"Hun lo og slo hendene sammen."* to to the English *"She laughed and hit her hands together."* compared to the ground-truth *"She laughed and clapped her hands."*. Here, we clearly see that the main message of the sentence is preserved, and that the action of clapping is also understandable from the translation. Overall, the quality of the translations are decent, in that they are actual sentences with meaning, but there are clear errors as well.

4.2.3 Combined dataset

Of the three datasets used, the models trained on the combined dataset performed best on the books dataset with a BLEU of over 15, while also achieving a BLEU of over 34 on its own dataset. The following results are based on the combined model with 4 decoder layers from table 2.

We see that the BLEU score on the evaluation set is way higher than the other two individual datasets, which also indicates that this model would perform better on the individual datasets too, which indeed is the case. Although not present in table 2, the model achieved a BLEU of 38.59 on the government dataset and 29.61 on the subtitles dataset. If we take a look at the same examples as the two previous models, the long government example is now generated as *"The investigation shows that the Ministry of the Environment and the Norwegian Climate and Pollution Agency to a limited extent carry out control to identify illegal export of hazardous waste."*, which is arguably better than the previous translation, as this one correctly indicates that some action was indeed carried out, whilst also describing the action more semantically correct using substitute words. As for the subtitles examples, this model now produced *"I'll go with him."* and *"What are you doing, Egon?*. Although the first sentence is a bit off the ground-truth in terms of substitute words, it still capture the meaning of the sentence completely, and the second sentence is exactly the same translation. Overall, we can say that the added data improved the translation for both datasets.

As for the book dataset, the BLEU score is better than the subtitles, but the overall quality may be the same. Looking at the same examples as the subtitles, *"Han holdt en gammel støvel i veiret."* is now translated to *"He kept an old boot in the weeds."*. We see that the grammatical error is corrected, but it is still unable to hit the word *"air"*, which is understandable as *"veiret"* is an uncommon and old spelling of the word. On the other hand, the other example is now translated to *"She laughed and hit your hands together."*, which is definitely a worse result, as the action is now targeted to a second person *"your"*.

4.2.4 Final test on DIY Restaurant Review

Although the few examples we looked at for the book dataset were not in the favor of the combined model compared to that of the subtitles model, the combined model still had a considerably better BLEU score, meaning that the overall performance is likely to be better. Therefore, we choose the combined model to test on the final held-out DIY restaurant review test set, to see how well it would perform on our application of interests.

The final BLEU score is a great 35.60. However, this may be an overly confident score when looking closer at the translations.

First of, we notice that in many cases it is able to translate long sentences quite well. An example is *"Det eneste negative var at vi skulle gjerne hatt en drikkemeny, men det kunne vi sikkert bare ha spurt om."* being translated to *"The only negative thing was that we would like to have a drink, but I'm sure we could just ask."*. This is a very good translation of a long sentence, although it slightly misses the menu part. Another good example is *"Et hyggelig og uformelt sted å samle 28 venner og familie en lørdagskveld i egen avdeling i koselige pub-omgivelser."* being translated to *"A nice and informal place to collect 28 friends and family a Saturday night in their own section of cub committees."*. Compared to the ground-truth in *"A nice and informal place to gather 28 friends and family on a Saturday night in our own section in cozy pub-surroundings."*, we see that the translation is mostly correct except some smaller parts, in this

case, the very end of the sentence. Smaller parts being wrong is a recurring theme for almost all cases, an example being *"Burgeren har vi spist bedre, den var ikke god."* being translated to *"The burger has eaten better, it wasn't good."*, while our custom translation was *"We have eaten better burgers, it was not good."*. For these examples, one would get a general impression of whether or not the review was good or not.

Surprisingly, the model struggles with short sentences. Some examples translations is *" ALDRI spis her!"* to *"ALDRI eat here!"*, where we see that the Norwegian word has leaked into the translation, and *"Meget sur betjening."* to *"I'm very surprised."* which have completely different meanings. However, a complete failure of the model was *"Ble godt tatt imot og plassert på bordet med en stor nesehornfigur, artig."* translated to *"Got a good caught and placed on the table with a big nose - skilled *****, fun."*, with ***** being a vulgar racial slur.

Overall, although having some minor errors, most of the sentences still keep the general sentiment of the review being positive or negative, with the exception of some extreme cases.

5 Discussion

5.1 Combining datasets

In section 4.2.3, we show that combining the government and the subtitles datasets results in better performance for both datasets, while also generelizing better to the books dataset. There are two interesting observations from this result. Firstly, this combination naturally yields more data that we can train our model with, which intuitively would mean a better trained model. Secondly, as the two datasets are somewhat on two completely different spectra in language, we can expect that combining them would neutralize some of the extreme cases, while also making the distribution of the data we train our model with more diverse. Therefore, when lacking data to train a model for an application of interest, in our case restaurant reviews, one can still produce decent results when transferring learned information from another domain.

5.2 Performance vs. runtime

As mentioned in the introduction, there is usually an inherent trade-off between performance and runtime when making a model more complex. However, from our results in table 2, we do not experience this trade-off as much as one would expect. When translating a few sentences at a time (as one would in practice) we found that the runtime for the subtitles model with a vocabulary size of 3,000, having a total of 2,271,200 trainable parameters was basically the same as the combined model with 4 decoder layers and a total of 7,288,660 trainable parameters. In general, all the models translated the sentence within a fraction of a second, and runtime should therefore not be a problem for any of them. However, the performance of the larger model is noticeably better than the smaller model. Therefore, this theoretical trade-off is not an absolute rule that makes or breaks a practical application, and in our case, we benefited greatly from having more parameters with little to no cost in runtime.

5.3 Performance of the final model

The goal of this paper is to build a NMT model for the practical application of translating restaurant reviews to reduce the uncertainty of visiting a restaurant. We introduced several properties such a system should have, and our final model mostly check all the boxes.

Firstly, the translated sentences should be of good quality, meaning that they are understandable. From a quantitative standpoint, the model's overall BLEU score on the restaurant review dataset was 35.60, which is quite good as discussed in section 2.4. For most of the examples presented in 4.2.4, although not perfect, one would most likely be able to infer the general idea of the translated sentences of what to expect from the restaurant.

Secondly, it is important that the sentiment of a translated sentence is kept in tact. For the examples in 4.2.4, this is mostly true. The only questionable examples may be *"ALDRI eat here!"* and *"I'm very surprised"*. However, the first of these sentences would most likely be seen as negative from the fact that *"ALDRI"* is in all capital letters followed by an exclamation mark. The second sentence has a neutral sentiment when taken out of context as it is in this case, because we simply cannot know what caused the surprise. In this case, it does lose its sentiment, but it probably will not do any harm as it does not turn positive either.

Lastly, the system itself should run efficiently for a better user-experience. As discussed in section 5.2, the performance of our model is very good, and should generally not be a hurdle.

One concern that has to be raised for our model is related to the output of *"Got a good caught and*

*placed on the table with a big nose - skilled *****, fun.".* As this model is intended for practical use, having the risk of producing such vulgar language is all but acceptable, especially when the application is mostly used in foreign countries, meaning that there may exist cultural barriers. For this reason, the model cannot be deployed to production without further development and sanity checking.

6 Conclusion

In this paper, we tackle the problem of translating restaurant reviews using neural machine translation methods. The main challenge of this task was the lack of a proper dataset. Therefore, we trained a model with data from other domains and directly transferred the model to our domain of interest by evaluating the model on a custom built restaurant review test set, which the model performed adequately on considering not being trained on any related data. We also show that training a model with data from several domains help the model generalize better. Although having many parameters, the resulting model still ran efficiently for our application. However, a major concern of our model is related to the lack of quality assurance, which is especially important considering the intended use-cases of the system. In the future, one should highly consider at least fine-tuning a model with some examples from the target domain when performing transfer learning from a completely different domain, principally when considering a text generation task.

References

Toan Q. Nguyen and Julian Salazar. 2019. Transformers without tears: Improving the normalization of self-attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better language modeling using shorter inputs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5493–5505, Online. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Noam Shazeer. 2020. Glu variants improve transformer.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation.

Contextual Lexical Substitution and Ego-Graph Vector Induction for Norwegian Word Sense Induction

Aksel Tjønn
akseltj@ifi.uio.no

Abstract

We have compared the two current basic approaches to Word Sense Disambiguation/Induction: the graphical and lexical substitution based techniques. The graphical approach being represented by the ego-graph vector induction (egvi) algorithm (Logacheva et al., 2020) using both the original fastText approach as well as novel alteration of egi using BERT embeddings, while the lexical substitution approach is represented by BertWSI (Amrami and Goldberg, 2019). We find that the WSI task is still an on-going NLP challenge, but key insights for future research highlight the importance of how many word senses/how large clusters are allowed, max sentence length as well as choice of masking patterns on WSI results. Our experiments show egvi outperforming BertWSI on Norwegian WSI, but also that more data is necessary to draw authoritative conclusions due to the wide confidence intervals involved.

1 Introduction

In most languages a word can have a different meaning based on its context, like "split" in yoga, cooking or programming, an infinite source of inside jokes in any field. The fact that different meanings of words were mapped to the same vector was one of the obvious and fundamental flaws of static word embeddings which lead to the development of contextual word embeddings in the first place.

However, every occurrence of a word in a text has a different contextual embedding, and so the problem of knowing which specific meaning is intended by the word used, trivial for humans, is still a major problem in NLP known as Word Sense Disambiguation (WSD) when solved in a supervised manner and Word Sense Induction (WSI) in the unsupervised case. (Amrami and Goldberg, 2019) Some researchers have even questioned whether the concept of discrete "word senses" makes ontological sense at all. (Kilgarriff, 1997)

2 Theory

WSI was previously done with static word embeddings. Each word in the context sentence had their static word embeddings averaged to produce a context sentence vector representation. The resulting context vectors are mapped out in the embedding space and will form clusters, which are assumed to represent the sense of the word given the context. Given a new context sentence the word embeddings can be averaged and the resulting vector will be closest to one of the clusters, which is the predicted word sense. K-means and agglomerative clustering are two examples of algorithms used for determining the geometrical limits of the word sense area in the embedding space.

One major problem with this approach is that the solution will vary widely depending on the amount of clusters induced from the data. An overly restrictive approach will force different meanings into the same sense cluster, and vice versa. Intuitively it is better to have too many sense clusters than too few, as each context can give slightly different nuances in implicit associations or implication, but it really should not be a hyper parameter choice but given by the natural clustering of the data by some objective algorithm. Affinity Propagation algorithm (Frey and Duech, 2007) is widly used for this purpose in WSI, as it provides the most likely number of clusters based on the clustering of the context vector representations in the embedding space. (Logacheva et al., 2020) improved upon the Affinity Propagation algorithm for use in the WSI problem by generating ego graphs for each word made from its closest vector neighbours in a static word embedding space like word2vec. However, Affinity Propagation algorithm also requires a hyperparameter choice to decide the sensitivity of the clustering, if we are stingy or generous in our definition of "sense neighbourhood". This is however even a problem for human annotators in lexicons

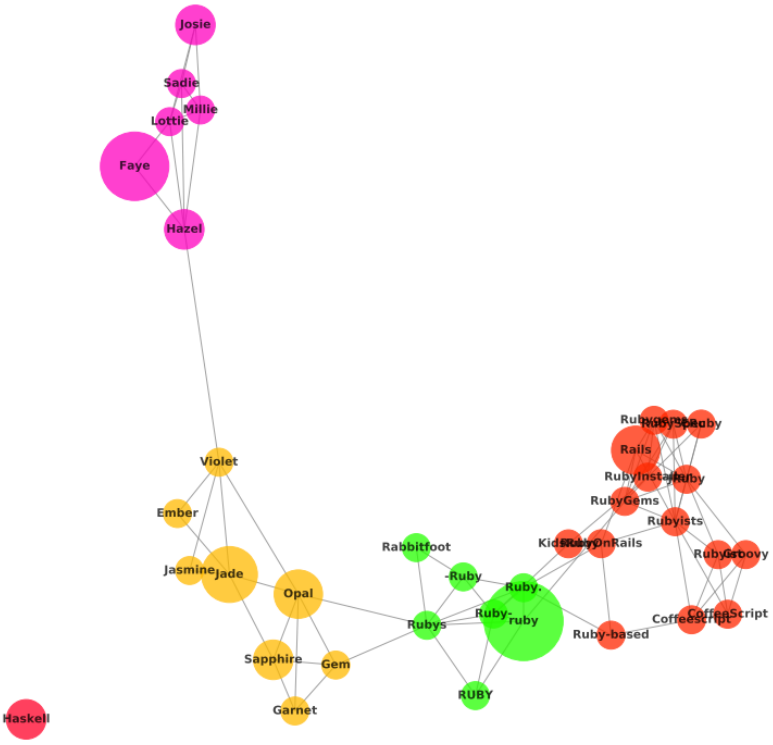


Figure 1: Figure from (Logacheva et al., 2020) demonstrates the ability of nearest neighbour graphs to distinguish different senses of the word 'ruby' in the context of a gem, the programming language and women's names.

and so should come as no surprise.

Semantic graphs can be constructed from the cosine-similarity of word embeddings (Pelevina et al., 2016) and give alternative form of context representation (Logacheva et al., 2020). For example, the semantic graph of "labrador" and "golden retriever" should be very similar, and this can also produce word sense representations by way of graph clustering algoritms. MaxMax is an example of such a graph clustering algorithm applied for word sense induction by (Hope and Keller, 2013).

SenseGram constructs ego-graph for each word, and then breaks those graphs into sub-graphs to distinguish word senses. For example, the ego-graph for the word "mouse" will have some nodes referring to rodents, pests and poison which will cluster together and others referring to buttons, scrolling wheels and pads which will be clustered away from the rodent cluster (Pelevina et al., 2017). The sense embedding vector is then formed by averaging the word embeddings of the words in each sense cluster.

Ego-Graph Vector Induction (egvi) was introduced in (Logacheva et al., 2020) and applies vector subtraction to identify the graph node pairs that are the most dissimilar (the anti-edges). This is done first in order to filter out some the plentiful noise produced between false edges formed between different word senses when including the closest neighbours of a word in the ego-graph. Additionally, this is done to combat the major SenseGram problem resulting from the fact that most of the words in the ego-graph will be related to the most commonly used sense of the word. As an example, words relating to the sense of the word "mouse" as a metaphor for a coward will be quite fringe in the mouse SenseGram and therefore hard to identify as a separate word sense if they have many edges to different word senses. We will also be using the egvi sense embeddings for WSI with BERT contextual embeddings for a contextual-to-contextual comparison with BertWSI.

2.1 Lexical substitution for WSI

The general task of replacing a word in a corpus with another based on the context is called lexical substitution, and is applied for NLP tasks like textual data augmentation, lexical relation extraction, paraphrase generation and text simplification. (Arefyev et al., 2020)

Originally lexical substitution was solved in a

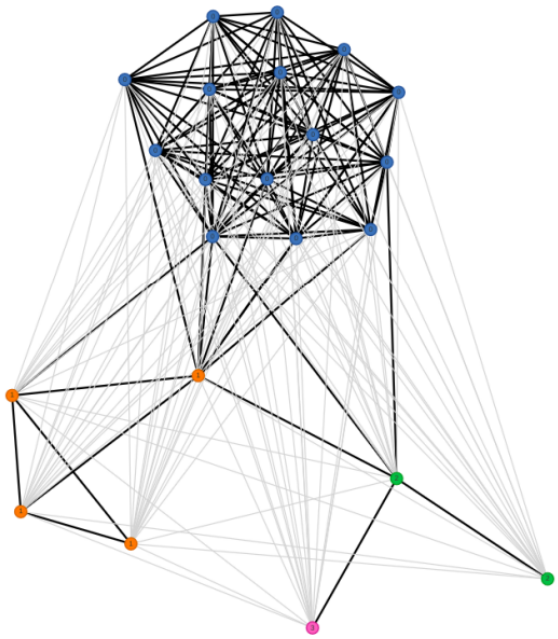


Figure 2: Figure from (Kutuzov et al., 2022) shows the word usage graph for the word 'innstilling'.

supervised manner relying on human-curated often external resources like WordNet to solve the problem as a classification task by way of sense embeddings, which are word embeddings that are trained specifically for separating word senses. SupWSD (Papandrea et al., 2017) is an example of a state-of-the-art supervised WSD, a pipeline which makes use of carefully chosen feature engineering, word embeddings as well as local collocations.

(Başkaya et al., 2013) instead attempted to identify potential substitute words using n-gram language models, and their vectors were clustered to identify differnet word senses.

(Amrami and Goldberg, 2018) first used dynamic pattern querying of recurrent language models for lexical substitution. Aside from the RNN, the dynamic symmetric patterns incorporated information about the target word itself for the word substitution task, improving results.

Later (Amrami and Goldberg, 2019) improved upon their own work by using contextual language models in the form of BERT instead of an RNN in their BertWSI, which further improved upon the lexical substitution approach. We will be reproducing BertWSI for the Norwegian NorDiaChange and compare it to egvi to see how these two approaches measure up in Norwegian WSI.

One of the latest state-of-the-art method for unsupervised lexical substitution is called ALaSca (Lacerra et al., 2021), which incorporates external knowledge from Wikipedia by ensuring their context heterogeneity with the target word via clustering to extract candidate lexical substitutes.

3 Dataset

We will be working on the NorDiaChange dataset subset 2, which was annotated to track how Norwegian words have changed meaning over time, i.e diachronic semantic change. (Kutuzov et al., 2022)

Word senses in two different sentences are classified by human annotators as either:

1. Unrelated.
2. Distantly related
2. Closely related.
4. Identical

The NorDiaChange dataset also includes word senses inventory, which is the part we will be using as the gold standard to evaluate our WSI performance.

4 Implementation details

4.1 Sense inventories and embeddings

We download sense inventories produced by (Logacheva et al., 2020) and calculate the word embeddings for each of the context word tokens connected to each word sense, and average these to calculate our sense embeddings for each of the word senses. These are compared to the fastText embeddings based on Norsk

Aviskorpus, NoWaC and NBDigital (ID 110 and 109 from http://vectors.nlpl.eu/repository/#) of the ambigious word we want to classify. http://vectors.nlpl.eu/repository/, where the closest matching embedding is chosen for the word sense classification.

We also use BERT for a contextual version of the egvi WSI algorithm. For each new sentence with an ambiguous word the contextual BERT token embeddings for the ambiguous word are averaged to produce a contextualized embedding for the ambiguous word. To determine the closest sense embedding to the ambiguous word we use the cosine-similarity measure between each of the possible sense embeddings and the word embedding, where the shortest cosine-distance is the predicted word sense solution for our WSI task.

The simplest way to understand the underlying goal we are trying to achieve here is that the sense embeddings contain information about the words usually associated with each word sense, like "river" for one sense of the word "bank" or teller" for another, and we pick the word sense with most of those associated words in the words surrounding the word. The contextualized embeddings help generalise the words used so if words like "current" are used instead of "river" this would still register.

Some sentences are too long and must be either truncated or discarded. Truncating made the results degrade considerably as sometimes the part of the sentence containing the word and its immediate neighbourhood are truncated, which ends up producing misleading context embeddings, so we chose to discard sentences that were too long. The max sentence length turned out to be one of the more important hyper parameters for improving the WSI results both with egvi and BertWSI.

4.2 Dataset preparation

The indices for the offsets of the words are often wrong. This causes the BERT tokenization to be different for the words in the sentence and the word we are looking for, to the point where the tokens are not found in the sentence and so we cannot find the BERT embeddings for the word. This leads to a data loss of around 25 percent, from around 800 to around 600 sentences. As a result, care has to be taken to make sure all the data is matched with the correct gold labels.

5 Evaluation

$$RA = \frac{TP + TN}{TP + FP + FN + TN} \quad (1)$$

The rand index (equation 1) checks how many class labels are coherent, that is, not identical as the class names for each word sense is different in the NorDiaChange dataset and the pre-trained sense inventories. TP is true positive, FP false positive, TP true positive, TN true negative and FN fake negative.

The adjusted rand index is adjusted for random chance so the results are easier to interpret without comparing to a baseline. We will be using the adjusted rand index to evaluate our WSI performance for all models to be able to compare their results.

5.1 Error estimation

As amount of training data does not make a difference when the model is frozen bootstrapping will likely introduce more error than cross-validation. We share the data in 5 portions and compute the standard deviation and resulting confidence interval.

6 Models

We test lemmatized and unlemmatized fastText, norBERT1, norBERT2, nb-bert-base and bert-base-multilingual-cased to see how the choice of pre-trained language model influences the WSI results.

Additionally, we test the three different sense inventories released by (Logacheva et al., 2020) for building the sense embeddings. One produced from a graph of top 50 nearest neighbours, another of the top 100 and a third the top 200 nearest neighbouring words and compare their WSI task results.

7 Results and discussion

7.1 Model choice effect

The fastText embeddings resylted in a performance for this task at a mean ARI of 0.056. The multilingual BERT-model turned out to produce the best results, closely followed by the nb-bert-base model. norBERT1 and norBERT2 did not quite match the other options. Overall the choice of model was one of the most important hyper parameter choices.

mean_rand_score	ci_std	confidence_interval	text_preprocessing	sense_choice	bert_choice	max_length	hidden_layers_early	hidden_layers_late
0.0124	0.0191	-0.0044-0.0291	none	sense50	nb-bert-base	60	1	6
0.0226	0.0356	-0.0086-0.0538	none	sense200	nb-bert-base	60	1	6
0.037	0.0252	0.0149-0.0591	none	sense100	norbert2	60	0	4
0.0376	0.0251	0.0155-0.0596	none	sense100	norbert2	60	1	6
0.0473	0.0494	0.004-0.0905	none	sense100	norbert1	60	1	6
0.0483	0.0463	0.0076-0.0889	none	sense100	nb-bert-base	70	0	4
0.0528	0.0513	0.0078-0.0977	none	sense100	nb-bert-base	60	0	1
0.0534	0.0557	0.0046-0.1022	none	sense100	nb-bert-base	60	0	3
0.0534	0.0548	0.0054-0.1014	none	sense100	nb-bert-base	60	1	4
0.0537	0.0568	0.0039-0.1035	lower	sense100	nb-bert-base	60	0	4
0.0543	0.0553	0.0058-0.1027	punctuation	sense100	nb-bert-base	60	0	4
0.0544	0.052	0.0088-0.1	none	sense100	multilingual-cased	60	1	6
0.0547	0.0508	0.0101-0.0992	none	sense100	nb-bert-base	50	0	4
0.0549	0.0571	0.0049-0.105	none	sense100	nb-bert-base	60	0	7
0.0556	0.0563	0.0063-0.105	none	sense100	nb-bert-base	60	0	4
0.0559	0.0573	0.0056-0.1061	none	sense100	nb-bert-base	60	0	8
0.0559	0.0534	0.0091-0.1028	none	sense100	nb-bert-base	60	0	2
0.0561	0.0578	0.0055-0.1067	none	sense100	nb-bert-base	60	0	6
0.0565	0.0584	0.0053-0.1078	none	sense100	nb-bert-base	60	1	6
0.0617	0.0531	0.0151-0.1082	none	sense100	multilingual-cased	60	0	4
0.0637	0.0503	0.0196-0.1078	none	sense100	multilingual-cased	50	0	4
0.0678	0.0528	0.0215-0.1141	none	sense100	multilingual-cased	45	0	4
0.0682	0.0686	0.0081-0.1284	none	sense100	nb-bert-base	40	0	4
0.0701	0.0368	0.0378-0.1023	none	sense100	multilingual-cased	30	0	4
0.0733	0.0536	0.0263-0.1204	none	sense100	multilingual-cased	35	0	4
0.0734	0.0608	0.0201-0.1266	punctuation	sense100	multilingual-cased	40	0	4
0.0744	0.06	0.0219-0.127	p+l	sense100	multilingual-cased	40	0	4
0.0755	0.0648	0.0187-0.1323	lower	sense100	multilingual-cased	40	0	4
0.0802	0.0632	0.0248-0.1355	none	sense100	multilingual-cased	40	0	4

Table 1: WSI results from the various hyper parameter choices. Hidden layers early is how many early layers of the pre-trained language model are averaged starting from layer 3, and hidden layers late how many late layers starting from layer 17.

7.2 Sense inventory choice effect

Surprisingly, the top 200 nearest neighbour word sense inventory did not perform as well as the top 100 inventory. We originally assumed more nearest neighbours would imply more information, but apparantly this is more irrelevant information which is not conducive to this particular dataset.

7.3 Text pre-processing effect

Overall using lower case or removing punctuation had a negative effect on the predictive power of our approach. It might be that this information helps contextualize the word to distinguish word senses.

7.4 Max sentence length effect

The choice of max sentence length turned out to have a major effect on the WSI results when using this approach. This might be because words further away from our target word has an undue influence on the word sense induction. Two options for future research based on this insight is to weight the BERT embeddings so words closer to the target word has a bigger influence on the final sense embedding. Another less drastic option could be to truncate strings that are too long, this was attempted straight forward by us, but in many cases the target word itself was truncated in these cases, resulting in poor performance. To try this solution properly care has to be taken to truncate the sentence around the target word and preferably not in the middle of a sentence.

7.5 Hidden layer choice effect

We tried summing a variety of hidden layer choices from the pretrained BERT model and observe the effect on the WSI results. Using the final four layers seemed to give the best results overall, and addings one or more of the earlier layers degraded the performance as well. The earlier layers encode more basic linguistic knowledge, so it makes sense that adding their values to the final embeddings reduces the impact from the more important later layers in determining the ambiguous word sense.

8 BertWSI reproduction comparison

8.1 Masking pattern choice effect

Model 1 from table 2 applied three masking patterns, one where the target word starts the sentence and is masked, one where the sentence ends with the target word which is masked, and one where it is masked in its original position. The results from these three masking patters were weighted and added. Model 2 from table 2 tried only using the original masking without additional patters and saw a substantial degradation from a mean adjusted rand score of 0.069 ± 0.048 compared to a mean of 0.032, although both are within the margin of error so additional data is required to confirm this.

Parameter	Parameter choice
n-represents	30
n-samples-per-rep	30
disable-tfidf	False
disable-lemmatization	False
min-sense-instances	2
bert-model	multilingual-cased
max-batch-size	10
prediction-cutoff	200
max-number-senses	5
Result BertWSI model 1	0.069 +- 0.048
Result BertWSI model 2	0.032 +- 0.057

Table 2: Results from BertWSI model 1 and 2. Difference between the two models is the type of patterns used for predicting a masked word in the lexical substitution phase, model 2 only uses the most obvious pattern leading to more repetitive synonyms.

Parameter	Parameter choice
n-represents	30
n-samples-per-rep	30
disable-tfidf	False
disable-lemmatization	True
min-sense-instances	2-4
bert-model	multilingual-cased
max-batch-size	10
prediction-cutoff	200
max-number-senses	5
Result BertWSI model 3	0.0171 +- 0.068
Result BertWSI model 4	0.0151 +- 0.059

Table 3: Results from BertWSI model 3, which is not using the additional patterns to improve results. Model 4 additionally reduces the maximum number of senses allowed to 4 from 5, which consistently reduced performance.

8.2 Max sense numbers effect

Limiting the maximum numbers of senses each word type can have is necessary as each added class makes it more likely to misclassify the word sense, even more "correctly" than the human annotators. As each sentence will have a slightly different context the choice of granularity really is an arbitrary one, and therefore it is necessary to match the choice to the choice of the human annotators of the specific dataset. Having too few choices makes it impossible to classify some word senses correctly. Having a maximum of 5 senses for each word type turned out to give the best results in our experiments, but the variance in our results are too high to be certain, especially for generalisation.

8.3 Lemmatization effect on performance

Adding pre-processing in the form of lemmatization degraded the performance of all models tested. This information appears to be helpful in performing the WSI.

9 Conclusion

We have compared a variety of hyper parameter choices to compare the performance of two state-of-the-art word sense induction methods on norwegian data: egvi and BertWSI. The best fastText egvi results were 0.056, the best bertWSI results 0.069 and the best egvi with BERT-embedding results at 0.08.

The models turned out to have a large variance in performance and we had only 800 sentences to classify. To determine the most influential parameters with confidence more data is required, a task for future research.

We conclude that the amount of word senses each word type can have had a major effect on the WSI, and further more that since the choice of word sense granularity inherently is arbitrary even for humans, for example a children's dictionary will not be as detailed as one for adults, this parameter will have to be adapted to the dataset and the level of detail provided by the human annotators to score highly.

Finally we conclude that the state-of-the-art NLP for word sense induction still has a way to go as our results overall were humbling.

9.1 Code resources

The BertWSI reproduction was based on this github repo: https://github.com/asafamr/bertwsi

Some BERT model loader code copied from: https://github.com/arushiprakash/MachineLearning/blob/main/BERT%20Word%20Embeddings.ipynb

References

Asaf Amrami and Yoav Goldberg. 2018. Word sense induction with neural bilm and symmetric patterns.

CoRR, abs/1808.08518.

Asaf Amrami and Yoav Goldberg. 2019. Towards better substitution-based word sense induction. *CoRR*, abs/1905.12598.

Nikolay Arefyev, Boris Sheludko, Alexander Podolskiy, and Alexander Panchenko. 2020. Always keep your target in mind: Studying semantics and improving performance of neural lexical substitution. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1242–1255, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Osman Başkaya, Enis Sert, Volkan Cirik, and Deniz Yuret. 2013. Ai-ku: Using substitute vectors and co-occurrence modeling for word sense induction and disambiguation. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 300–306.

David Hope and Bill Keller. 2013. Maxmax: a graph-based soft clustering algorithm applied to word sense induction. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 368–381. Springer.

Adam Kilgarriff. 1997. I don't believe in word senses. *Computers and the Humanities*, 31(2):91–113.

Andrey Kutuzov, Samia Touileb, Petter Mæhlum, Tita Ranveig Enstad, and Alexandra Wittemann. 2022. Nordiachange: Diachronic semantic change dataset for norwegian. *CoRR*, abs/2201.05123.

Caterina Lacerra, Tommaso Pasini, Rocco Tripodi, and Roberto Navigli. 2021. Alasca: an automated approach for large-scale lexical substitution. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 3836–3842.

Varvara Logacheva, Denis Teslenko, Artem Shelmanov, Steffen Remus, Dmitry Ustalov, Andrey Kutuzov, Ekaterina Artemova, Chris Biemann, Simone Paolo Ponzetto, and Alexander Panchenko. 2020. Word sense disambiguation for 158 languages using word embeddings only. *CoRR*, abs/2003.06651.

Simone Papandrea, Alessandro Raganato, and Claudio Delli Bovi. 2017. SupWSD: A flexible toolkit for supervised word sense disambiguation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 103–108, Copenhagen, Denmark. Association for Computational Linguistics.

Maria Pelevina, Nikolay Arefyev, Chris Biemann, and Alexander Panchenko. 2017. Making sense of word embeddings. *CoRR*, abs/1708.03390.

Towards a Lightweight Transformer-Based Model for Translation of Tourists Inquiries

Anastasiia Grishina
Simula Research Laboratory
Kristian Augusts Gate 23, 0164, Oslo, Norway
anastasiia@simula.no

Abstract

A number of large foundation models for language understanding and automatic translation have been proposed by the research community over the last two decades. However, these models appear too large for deployment in production. In real word scenarios, practitioners are limited by the availability of resources for training and have business-driven requirements for the speed of translation. This paper presents a series of experiments on scaling down a classical transformer model to translate sentences from English to Russian in a chatbot for tourists. We investigate how the scaled model generalizes from the domains of diplomacy and popular science in the limited-size training data to the target domain of tourist inquiries on a Trans-Siberian train. The code and materials are available at https://github.uio.no/anastg/NLP-IN9550-Simula-exam-NMT.

1 Introduction

In the pursuit of natural language understanding by artificial intelligence, the research community has proposed a number of large foundation models over the past decade (Bommasani et. al, 2021). While these models perform well on downstream tasks, practitioners often search for smaller models that can be deployed for real-world applications. Moreover, companies may have limited resources and choose training custom models from scratch, due to high costs of using a cloud infrastructure or of purchase of a GPU. Therefore, creation of lightweight models is a direction of ongoing research (Han et al., 2016; Schick and Schütze, 2021).

In addition to size-driven limitations of foundation models, training data within a target domain is often difficult to gather. In this situation, model creators must ensure the model performs well on the domain outside of the training data. This problem is known as domain shift and tackled with domain adaptation techniques (Ramponi and Plank, 2020).

This paper presents a series of experiments on scaling down a classical transformer model by Vaswani et al. (2017) to be deployed in a real-world scenario that requires English→Russian translation. The scenario details are provided in the Problem Description Section. We train our tokenizers and models from scratch on a limited-size dataset constructed from two domains and test the model on the third domain. By exploring model configurations, we focus on balancing between a vocabulary size and a model size. The key contributions of this study are as follows:

- we scale a transformer-based model down to 5 million parameters and test it on a dataset of user inquiries in the tourism domain;
- we examine the model's generalizability depending on the training datasets distribution between two different domains;
- we thoroughly analyze translation errors in the target domain and propose directions for model improvement.

The remainder of the paper is organized as follows. We present the real-world use case at the core of this study in Section 2 and the methodology and data in Sections 3 and 4, correspondingly. Experimental design details are covered in Section 5. Results, including the error analysis, are discussed in Section 6. We provide an overview of related work in Section 7 and conclude with Section 8.

2 Problem Description

In the use case of this study, a company supports a chatbot with well-configured conversational paths for user prompts in Russian. The purpose of the chatbot is to help tourists during a journey on a Trans-Siberian train. However, the chatbot lacks support for English.

To introduce the English language, one can choose from at least two options. While one option is to create decision nodes and mock-up conversations fully in English, it would require copying

and translating the existing paths in Russian. This option can be suboptimal due to a possible lack of the English language diversity. On the contrary, we aim to train a neural machine translation (NMT) model that translates user input from English to Russian automatically. We will further map the translated input to existing conversation paths in Russian. Backward translation from Russian to English is out of scope of this work.

The scenario and available data for training a model pose several constraints. Given the real-time usage of the chatbot, the model should be lightweight, i.e., contain less than 10 million (M) trainable parameters. Moreover, the model must generalize well from the domains of available training data to the domain of tourist inquiries. Finally, the training dataset must comprise a maximum of 300 thousand (K) sentence pairs.

3 Methodology

For the tokenization step, we train two byte-pair encoding (BPE) tokenizers that cover both Russian and English words (Sennrich et al., 2016). In this way, the embedding layer weights of the model are shared with its final classification layer.

For translation, we use the transformer-based encoder-decoder architecture with several variations (Vaswani et al., 2017). The first set of variations concerns the architecture tweaks that are reported to improve performance metrics in the studies that propose them, such as substituting the fully connected feed-forward network (FFN) layers with the FFN_{GEGLU} non-linear variant from Shazeer (2020). The implementation also contains the "pre-norm" variant from Nguyen and Salazar (2019), where the layer normalization is applied before the feed-forward layer transformation and residual connection. Moreover, the model has the "position-infused attention" from Press et al. (2021) that ensures the positional encodings are added to key and query vectors at each transformer layer instead of using the positional encodings prior to the encoder or decoder layers.

The second variation concerns the size of the transformer. Given that the overall goal is to create a light-weight model, we experiment with the number of layers in the encoder and decoder as well as the hidden size and number of heads. Detailed variations are explained in Section 5.1. Experiments with variations of the transformer-based NMT model have the following objectives: (i) to evaluate the generalizability of the model, i.e., how the model trained on small data from two domains performs on the target domain of the chatbot for tourists; (ii) to explore the trade-off between a larger vocabulary—that infers a larger embedding and classification layer—versus a larger number of hidden layers or more heads; (iii) to investigate how additional small variations in the training process, such as learning rate schedulers and BPE-dropout at the tokenizer level, affect the model performance (Provilkov et al., 2020). We use the BLEU score for evaluation (Papineni et al., 2001). BLEU is a precision-oriented metric based on the count of words from a generated translation that also appear in the golden translation.

The transformer with described variations outputs a distribution over the tokenizer vocabulary. Therefore, to produce a translated sentence word-by-word at inference time, we use greedy search for validation within the training domain and beam search for testing on the target domain with the beam size of 4.

4 Data and Preprocessing

Available data for training spans two domains: parallel corpora of the United Nations (UN) documents[1] and TED transcripts.[2] The UN corpus is small and contains formal diplomatic language. By contrast, TED is a larger source of semi-formal language with generic or popular science content.

We form three main training sets: "UN" that contains 50K training examples, "TED" that is comprised of 250K training samples, and the "combined" dataset that contains both training sets and counts for 300K training samples. In addition, we use the set "TED-300" that contains as much of the TED data as possible under the experiment requirements. Note that "300" stands for 300K, the result of rounding up the total number of sentence pairs in the TED data. For more details regarding the dataset statistics after preprocessing, see Table 1.

Given the data origins, we hypothesise that the UN corpus will contribute to the training only with the structure of the sentences, but not with the semantic value. If a model overfits the corpus, it will tend to repeat report numbers, country names, and specific words of the corpus. By contrast, we expect that models trained on the TED or combined corpora will perform better, because of the variety

[1] https://opus.nlpl.eu/MultiUN.php
[2] https://opus.nlpl.eu/TED2020.php

Dataset	Purpose	# sentences	Avg. len. EN	Avg. len. RU
UN	train	50 000	108	117
TED	train	250 000	80	79
TED-300	train	283 129	80	79
combined	train	300 000	85	85
UN	dev	2 500	109	120
TED	dev	2 500	80	79
TED-300	dev	2 500	80	79
combined	dev	5 000	95	100
book	test	2 500	79	72
TS	test	100	51	52

Table 1: Sizes of train/dev/test datasets, including average length of sentences in words separated by spaces, prior to tokenization.

of represented topics.

Development (dev) sets within training domains consist of 2500 pairs of held-out UN and TED sentences, and their combination for the "combined" training set. We keep 5000 pairs for the "combined" dev set, because it corresponds to 98.4/1.6% split of the combined version of the data. Given the small dev set, every sentence is useful for the validation stage and we do not reduce the number of UN dev sentences to the same proportion as the one of the UN/TED data in the combined training set.

The test set for the target domain contains 100 hand-crafted sentences in English translated to Russian by the chatbot creators. We call this custom dataset "TS". According to the chatbot team, the most common inquiries concern route and hotel suggestions, ticket and insurance details, practicalities on the train regarding towels, bed linen, showers, and food. Moreover, TS contains the shortest sentences on average over all dev and test sets.

To enlarge the test set of the target domain, we use the "book" corpus. This additional corpus is comprised of professional translations of 2500 random sentences from "Anna Karenina" by Leo Tolstoy from Russian to English. Given that the book corpus should contain general Russian language, evaluation on this set can approximate the performance of the model on some questions outside of the TS set. However, one challenge of the book corpus is that it contains sentences with direct speech and dialogues. Therefore, the model will be required to recreate the punctuation patterns that are more complex in Russian than in English and are not present in any of the training corpora.

The preprocessing step consists of adding the opening brackets and semicolons to the Russian part of the UN corpora, so that these punctuation signs match in Russian and English in the sentences that represent numbered lists, because in this corpus, the numbered lists open with "1)" in Russian and "(1)" in English. We also remove the indication of background noises in square brackets, such as "[Applause]" or "[Music]", from the TED corpus.

5 Experimental Design

5.1 Model and Tokenizer Configurations

The trained tokenizers vary by vocabulary length: 2000 tokens (small) and 5000 tokens (large). To choose the size of tokenizers, we increase the vocabulary size starting from 1000 with a step of 500 and manually inspect the results of tokenization on several sentence pairs. The size of 2000 is the smallest one that yields more subwords than single letters as tokens. The idea is increase the vocabulary size and shrink different parts of the model. The tokenizer size of 5000 is chosen based on the resulting number of model parameters.

Following the notation of Vaswani et al. (2017), the baseline model contains $N = 6$ encoder and the same number of decoder layers, the hidden layer size is $d_{model} = 256$. The model has $h = 4$ heads to keep the dimensions of keys, values and queries as $d_k = d_v = d_q = d_{model}/h = 64$ in Vaswani et al. (2017). Table 2 presents model names for all the configurations, including the baseline, and an overview of model sizes. Models are named as $model_{N,h}$, where parameters N and h are replaced with their values, so the baseline model is $model_{6,4}$ coupled with the 2000-token vocabulary.

We also fix the large vocabulary with 5000 tokens and decrease the number of encoder and decoder layers to $N = 3$ while keeping the hidden dimension and number of heads constant ($model_{3,4}$ in Table 2). Alternatively, we decrease the number of heads and the hidden layer dimension $d_{model} = 128,\ h = 2$, but keep the proportion $d_{model}/h = 64$ and other parameters as in the baseline ($model_{6,2}$ in Table 2).

5.2 Minor Training Enhancements

We have noticed that scaling down randomly initialized embedding weights positively affects the training process. Therefore, we divide the embedding weights by $\sqrt{d_{model}}$ by default.

Model	N	d_{model}	h	Vocab. size	Number of parameters
$model_{6,4}$ (baseline)	6	256	4	2 000	8.42 M
$model_{3,4}$	3	256	4	5 000	5.24 M
$model_{6,2}$	6	128	2	5 000	2.63 M

Table 2: Variations of the transformer model in experiments, where N is the number of encoder or decoder layers, h stands for the number of heads, and d_{model} is the hidden size of the model layers.

Train corpus	Dev/test corpus			
	UN	TED	book	TS
UN	9.7	0.6	0.1	0.0
TED	4.3	10.0	5.9	13.1
TED-300	4.7	12.6	**6.4**	**17.0**
combined	17.0	7.6	3.3	6.1

Table 3: BLEU score obtained using the baseline $model_{6,4}$ with 2000-token vocabulary depending on the training corpus.

Moreover, we perform four experiments with learning rate scheduling and BPE dropout: (a) linearly increasing learning rate from ca. 0.0003 to 0.001 over the first 5 optimizer steps (default parameters for LinearLR in PyTorch); (b) linearly increasing the learning rate from 0.001 to 0.003, using a higher final learning rate in an attempt to speed up learning; (c) linearly increasing learning rate over the first 4000 steps and decreasing thereafter as in Vaswani et al. (2017); and (d) setting BPE-dropout to 0.1.

5.3 Implementation Details

The model is implemented in Python 3.7 using PyTorch v1.7.1 and SacreBLEU from TorchMetrics v0.7.3. The training is done with the compute resources of Saga[3], namely 2 NVIDIA P100 GPU nodes for 2-10 hours depending on the model and corpus sizes, as well as the number of epochs.

We use the batch size of 1024 tokens, 0.1 dropout rate for encoder and decoder parts, Adam optimizer with hyperparameters set as in Vaswani et al. (2017). Generated sentences have the maximum length of 128 tokens.

By default, experiments are run for 10 training epochs. In the end, the best performing model with chosen minor training enhancements is trained on the training set that yields the best BLEU score for the TS and books test sets for 50 epochs.

6 Results and Discussion

Overall, the performance of our model is rather modest: the best performing model achieves 19.9 BLEU score on the TS dataset. We associate poor performance on held-out datasets with several factors. In the trade-off between performance and a light-weight model, the performance does not win, and in this paper we had the lightness described in Section 2 as a primary requirement.

In addition, we train the model only on one task, while large contextualized models are trained on multiple tasks and/or are fine-tuned on the target task. In our experiments, the models overfit on the train datasets with the content outside from the target domain. We discuss model performance variation depending on the training data and model size variation in detail below.

6.1 Domain Transfer

Results of training the baseline model configuration ($model_{6,4}$) for 10 epochs on different datasets are presented in Table 3. There, we report BLEU scores on average over dev or test batches on the dev sets, which are the same as training sets, and on test sets that are named differently from the train/dev sets. For example, the baseline model trained on TED-300 and validated on the TED set is tested on the UN, book and TS datasets.

The model trained on UN shows the worst generalization capability, while the model trained on TED generalizes better on all the test sets. Notably, the model trained on the combined data yields the highest BLEU scores on UN and TED, but not on the target book and TS sets. Because the model trained on partial TED corpus performs well, we enlarge the TED corpus as much as possible under 300K training samples requirement and form the TED-300 dataset (see Table 1). On this dataset, we obtain the best BLEU scores on the book (6.4) and TS (17.0) corpora with the baseline model.

At this stage, we also experiment with training process enhancements described in Section 5.2, starting from the baseline model trained on the combined corpus with linearly increasing learning rate *(a)*. Neither the larger learning rate *(b)* nor the learning rate following Vaswani et al. (2017) schedule *(c)* improves the BLEU scores. With the BPE-dropout scheme *(d)*, the performance stays

[3] https://documentation.sigma2.no/hpc_machines/saga.html#saga

Model	Epoch	Dev/test corpus				Decode
		UN	TED	book	TS	time, sec
$model_{6,4}$	10	4.7	12.6	6.4	17.0	1.5
$model_{3,4}$	10	6.2	14.3	6.5	**19.9**	0.8
$model_{6,2}$	10	5.0	12.6	6.0	13.4	0.5
$model_{3,4}$	50	**8.4**	**17.4**	**8.1**	19.5	0.8

Table 4: BLEU score and average translation (decode) time for one sentence on the TS dataset obtained using the models of variable size.

the same on TS and deteriorates for other corpora. Therefore, we stick to TED-300 corpus and option *(a)* in other experiments.

6.2 Model Size Variations

The goal of this experiment with variable model sizes is two-fold. In particular, we aim at choosing the best performing model to train it for more epochs. In addition, we try to reduce the model size and investigate the effect on the average speed of sentence decoding, i.e., translation from English to Russian. Table 4 presents the results.

The best performance—19.9 BLEU on TS—is achieved on the model with 3 encoder and decoder layers and 4 heads, $d_{model} = 256$, when it is trained for 10 epochs ($model_{3,4}$). Training this model for more epochs leads to overfitting the train set, which is TED-300. Noteworthy, this overfitting leads to the best BLEU scores for all datasets, except for TS. The smallest model with 2 heads, 6 encoder and decoder layers and $d_{model} = 128$ performs worse than both the baseline (the largest, $model_{6,4}$) and the medium-size model ($model_{3,4}$).

The speed of decoding changes from 1.5 seconds per sentence for the baseline 8.42 M-parameter model by 47% to 0.8 seconds/sentence for the best performing model with 5.24 M parameters. We to evaluate the speed of translation on the CPU of MacBook Pro 2021 edition.

6.3 Error Analysis

Certain errors in the translation of TS sentences are consistent throughout the dataset. We analyze the translation capability of the best performing model trained for 50 epochs ($model_{3,4}$). Although the BLEU score is higher when the model is trained for 10 epochs, we notice that after 10 epochs the model has poorer translation quality on TS than the model trained for 50 epochs. It can happen if more subtokens are decoded correctly after 10 epochs but the results are evaluated with full words by humans. Examples of translation are listed in Table 1 in Appendix A and referenced by example IDs in the text.

One example of an error is that the model confuses English words with other English words that start with the same syllable. For example, "towel" is translated as "tower" (Ex. 1), "luggage" is converted to "lung" (Ex. 2), "visa" is confused with "visual" (Ex. 3), and "wifi" with "wife" (Ex. 4). Being trained on a rather small corpus of less than 300K examples in TED-300, the model confuses gender forms of the possessive pronouns, singular forms with plural ones for different parts of speech (Ex. 14), chooses the wrong verb form (perfect/imperfect, Ex. 5), or mixes prepositions (Ex. 15). In addition, the model has translated a verb as a homophone noun in some sentences ("a taste" vs. "to taste", Ex. 5). One obvious mistake type is that cities are either misspelled or spelled with English letters. This error possibly originates from the domain transfer, because Russian cities' names do not appear in the training corpora at large.

Certain words are more difficult to translate than others, because they depend on the context. For example, in the phrase "on board the train", the part "on board" means "while being on the train". However, the model confuses the "on board" part with a board of a table (Ex. 6) or with a board of organization members (Ex. 7). In other cases, the "on board" part is simply omitted. Out of eight sentences with the phrase "on board", only two are translated correctly and, in two others, the phrase is omitted without the loss of the sense (Ex. 11).

One eye-catching mistake the model has made is the invention of new words (Ex. 8). The root is taken from one word, the suffix and ending are customary for reflexive verbs. These exact parts—the root, suffix and ending—are present as subwords in the BPE vocabulary, which may be the reason for such a translation. As a result, the verb resembles a real word that a child would invent to convey the meaning. In a similar direction, one difficult word for translation, "a cabin", is decoded as a transcription of an English word in cyrillic letters (Ex. 9), possibly because the word "cabinet" is well-represented in the TED-300 text and the start of its translation in Russian coincides with the cyrillic transcription.

Overall, the model has yielded around 20% translations of all sentences with no mistakes or minor mistakes that do not hinder understanding of a

tourist inquiry (evaluated by authors). It is noteworthy that even when a wrong preposition is chosen, the subsequent noun has the right ending. In Russian, the noun ending depends on the preposition, or more specifically, on the noun case that is determined by surrounding words (Ex. 10, 15).

7 Related Work

This study is closely related to three research areas: natural language understanding and translation; reducing the amount of resources for training language models; and cross-domain performance of natural language models.

Natural language understanding and NMT have been rising as research fields since the emergence of contextualized language models, such as BERT, Multilingual BERT (Devlin et al., 2019), and mT5 (Xue et al., 2021). Furthermore, architecture solutions from these models have been reused by large companies, such as Google (Wu et al., 2016). Among all, large models for Russian exist as both standalone models (Kuratov and Arkhipov, 2019) and parts of multi-lingual ones (Devlin et al., 2019; Xue et al., 2021; Reimers and Gurevych, 2019).

The trend to create larger and larger language models has faced critique from the field, due to sustainability aspects of training and using the models, as well as other negative factors, for example, reproducing social biases from the Internet corpora (Bommasani et. al, 2021; Bender et al., 2021). As a response, several studies focus on scaling down the models and speeding up training via knowledge distillation (Sanh et al., 2020), pruning (Han et al., 2016), early exit strategies for inference. For example, Cañete et al. (2022) focus on creating a light-weight model in Spanish.

In the same line of research, Schick and Schütze (2021) study how the large (GPT-3) and small (ALBERT) models perform on the SuperGLUE benchmark with variable number of parameters and propose a variant of training that improves ALBERT's performance (Wang et al., 2020). In addition, Chen et al. (2021) use prompt-guided attention to train the model for a low-resource NER task.

Finally, generalizability of models is a common topic for AI as a whole. For natural language, the generalizability is mainly achieved through fine-tuning of a general model on a target domain (Raffel et al., 2020). In this manner, several models are created to cover bio-medical and other specific domains (Lee et al., 2019).

8 Conclusion

In this study, we have experimented with transformer-based models on top of the BPE tokenizer and investigated the problem of domain shift. We have trained from scratch and tested models and tokenizers with different sizes, starting with a small vocabulary and a model of 8M parameters and gradually increasing the vocabulary size while reducing the number of layers or heads to reach 5M model parameters and 47% translation speed-up.

The experiments have been guided by a technical use case of translating tourists inquiries on a Trans-Siberian train from English to Russian. Technical requirements have imposed upper limits on the model and training data size. Under these requirements, we have achieved the BLEU score of 19.5 on the target domain corpus after 50 epochs, and 20 out of 100 inquiries have been translated in the manner understandable for a native speaker.

This work can be extended in various directions. One option is to replace attention-based encoder and decoder with RNNs. To speed up training, we may use larger batch sizes—an option we overlooked in this paper. Finally, more Russian cities can be present in the training corpora, so that the inquiries with geographical named entities are covered well.

Acknowledgements

The authors thank David Samuel for providing a code basis for this study and the NLPL group at the University of Oslo for the IN5550 course.

References

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In *Conference on Fairness, Accountability, and Transparency*, pages 610–623, Virtual Event Canada. ACM.

Rishi Bommasani et. al. 2021. On the Opportunities and Risks of Foundation Models. Technical Report arXiv: 2108.07258, CoRR e-print.

José Cañete, Sebastián Donoso, Felipe Bravo-Marquez, Andrés Carvallo, and Vladimir Araujo. 2022. ALBETO and DistilBETO: Lightweight Spanish Language Models.

Xiang Chen, Ningyu Zhang, Lei Li, Xin Xie, Shumin Deng, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2021. LightNER: A Lightweight Generative

Framework with Prompt-guided Attention for Low-resource NER.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL, Vol 1)*, pages 4171–4186. Association for Computational Linguistics.

Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.

Yuri Kuratov and Mikhail Arkhipov. 2019. Adaptation of Deep Bidirectional Multilingual Transformers for Russian Language.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2019. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, page btz682.

Toan Q Nguyen and Julian Salazar. 2019. Transformers without Tears: Improving the Normalization of Self-Attention. In *Proceedings of the 16th International Conference on Spoken Language Translation*, page 9, Hong Kong. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2001. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02*, page 311, Philadelphia, Pennsylvania. Association for Computational Linguistics.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better Language Modeling using Shorter Inputs. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5493–5505, Online. Association for Computational Linguistics.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-Dropout: Simple and Effective Subword Regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. Technical Report arXiv: 1910.10683, CoRR e-print.

Alan Ramponi and Barbara Plank. 2020. Neural Unsupervised Domain Adaptation in NLP—A Survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6838–6855, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter.

Timo Schick and Hinrich Schütze. 2021. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Meeting of the Association for Computational Linguistics (ACL, Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Noam Shazeer. 2020. GLU Variants Improve Transformer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *International Conference on Neural Information Processing Systems (NeurIPS)*, pages 5998–6008. Curran Associates, Inc.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2020. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.

Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2021. mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498, Online. Association for Computational Linguistics.

A Appendix

ID	Source sentence, En	Tranlsation produced by the model, En -> Ru	Manual backward translation with explanation of model errors, Ru -> En
		Translations with errors	
1	Where can I change **the towels and bed linen**?	Где же я могу изменить **башни и спать**?	Where can I change **the towers and sleep**?
2	What is the size limit of the **luggage**?	Каковы размеры **лёгких**?	What are the sizes of **lungs**?
3	Do I need a visa for a trip?	Мне нужна **визуальная** поездка?	Do I need a **visual** trip?
4	**What is** the **wifi** password?	**Что такое** пароль **жены**?	What **constitutes** the **wife** password?
5	I want **to taste** something Russian. Where **can I find** some food?	Я хочу **вкус** что - то русское. Где я **нахожу** еду?	I want **a taste** something Russian. Where **do I find** (imperfect verb form) some food?
6	Can I take an electric **scooter on board** the train?	Могу ли я взять электрический **аккумулятор на доске**?	Can I take an electric **accumulator** (meaning: battery) **on a board** (meaning: board of a table/whiteboard/any flat board)?
7	Does the police **on board** *speak* English?	*Считает* ли полиция **на совете** на английском языке?	Does the police *consider* English (meaning: language) **during the board meeting** (noun case corresponds to the preposition)?
8	What should I do in case I **fall ill** on board?	Что мне следует сделать, если я **болюсь** на борту?	What should I do in case I **[fall ill]** (impossible to translate: correct root, suffix and ending, but taken from different words, the word does not exist) on board?
9	How can I lock **my cabin**?	Как я могу за**х**локировать **кабин**?	How can I lock (spelling mistake) a **[cabin]** (spelled in Russian as pronounced in English)?
		Translations with minor or no mistakes	
10	Do you have a play room for children?	У вас есть игров**ую** комнат**у** для детей?	(wrong noun case and adjective ending)
11	Can I take a bike **on board** the train?	Могу ли я взять велосипед?	Can I take a bike? ("on board" is omitted without the loss of the meaning)
12	Can I go out to the city during a stop?	Могу ли я поехать в город во время остановки?	(correct)
13	Why are the toilets closed during the stops?	Почему туалет закрывается во время остановки?	(correct)
14	**What** is the average speed of the train?	**Каковы** средняя скорость поезда?	**What** [plural form of "what" instead of the correct singular one] is the average speed of the train?
15	Is there passport control on the border with Mongolia and China?	Есть ли паспортный контроль **над границей** Монголии и Китаем?	Is there passport control **over the border** (noun case corresponds to the preposition) **with** (omitted preposition) Mongolia and China?

Table 1: Examples of translation of tourists inquiries from the TS corpus by $model_{3,4}$ trained for 50 epochs. Legend: translation mistakes; correct grammatical features; **an incorrectly translated word in English**; *a slight change of a meaning that does not hinder the general understanding of the sentence*.