Text Processing with Deep Learning

Basic concepts

Narada Warakagoda

Forsvarets Forskningsinstitutt (FFI)

Table of contents

Word Representations

Why Word Representations?

- Words are symbols
- Neural networks operate on numerical values

Naive way of Word Representation

One hot encoding

Use the word index in vector form

Example

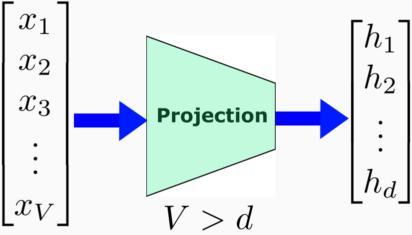
- Consider a vocabulary of 5 words:

1	Man	[1,0,0,0,0]
2	Woman	[0,1,0,0,0]
3	Boy	[0,0,1,0,0]
4	Girl	[0,0,0,1,0]
5	House	[0,0,0,0,1]

Disadvantages

- Dimension of the representation vector would be very high for natural vocabularies.
- All vectors are equally spread (vector similarity does not represent semantic similarity)

Issue 1: High Dimension

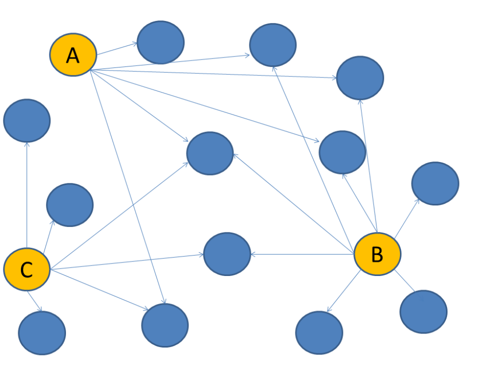


- Project one-hot encoded vectors to a lower dimensional space (Reduce the dimension of the representation)
- Also known as *embedding*
- Linear projection = Multiplication by a matrix $\mathbf{h}_{1\times d} = \mathbf{x}_{1\times V}\mathbf{W}_{V\times d}$

Issue 2 :Similar Words

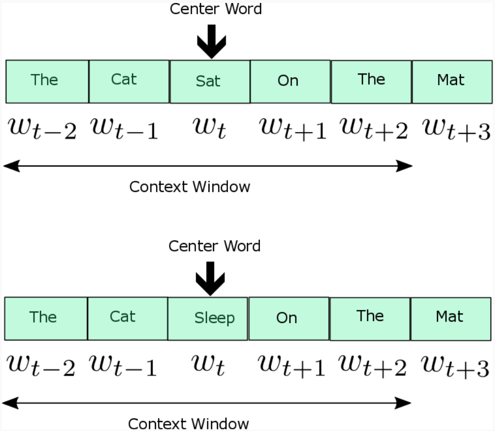
- Force vector distance between similar words to be low
- How to quantify word similarity?

Quantifying Similarity



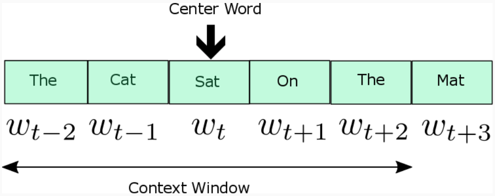
1. A is ”more similar” to B than C
2. A is ”more similar” to C than B

Quantifying Word Similarity



- Context of a word = Words occurring before and after within a predefined window
- Words that have similar contexts, should be represented by word vectors close to each other

Capturing Word Contexts



- Consider a word w_t (call it the center word)
- Consider another word w_{t+j} that lies within the context window of size C. Then $-C \leq j \leq C$ and $j \neq 0$
- We want to use the probability of context words given the center word $P(w_{t+j}|w_t)$ for $-C \leq j \leq C$ and $j \neq 0$
- If the total number of words in training database is T, then, try to maximize the overall probability

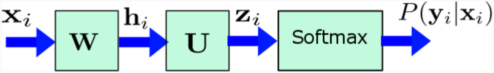
$$\prod_{t=1}^{T} \prod_{-C \leq j \leq C, j \neq 0} P(w_{t+j}|w_t)$$

Putting All Pieces Together

- Scan training data and prepare training data pairs
  - Eg: if data are $(w_1, w_2, w_3, w_4, \cdots w_T)$, then assuming a context window of 2, the training word pairs will be $\{(w_1, w_2), (w_1, w_3), (w_2, w_1), (w_2, w_3), (w_2, w_4), \cdots\}$
  - In each word pair replace the first word with the corresponding one-hot encoded vector and the second word with its index $\{(\mathbf{x}_1, y_2), (\mathbf{x}_1, y_3), (\mathbf{x}_2, y_1), (\mathbf{x}_2, y_3), (\mathbf{x}_2, y_4), \cdots\}$, where y_i is the index of word w_i.
  - For clarity denote the i^{th} pair by $(\mathbf{x}_i, y_i)$ where $\mathbf{x}_i$ is the input and y_i is the target. Let M be number of such pairs.
- Consider a neural network whose
  - First layer performs a projection to the word vectors $\mathbf{h}$ from the one-hot encoded vectors $\mathbf{x}$.
  - Second layer maps the word vectors to target one-hot vectors
- Train the network to maximize

$$L = \prod_{t=1}^{T} \prod_{-C \leq j \leq C, j \neq 0} P(w_{t+j} | w_t) = \prod_{i=1}^{M} P(\mathbf{y}_i | \mathbf{x}_i)$$

Word2vec- System Architecture



$\mathbf{x}_i \in \mathbb{R}^{V\times 1}$, $\mathbf{h}_i \in \mathbb{R}^{d\times 1}$, $\mathbf{W} \in \mathbb{R}^{V\times d}$, $\mathbf{U} \in \mathbb{R}^{V\times d}$

- Projection:
$$\mathbf{h}_i = \mathbf{W}^T\mathbf{x}_i$$
- Second layer:
$$\mathbf{z}_i = \mathbf{U}\mathbf{h}_i$$
- Softmax:
$$P(y_i = j|\mathbf{x}_i) = \frac{\exp(z_i(j))}{\sum_k \exp(z_i(k))}$$

Projection

$$\mathbf{h}_i = \mathbf{W}^T \mathbf{x}_i \tag{1}$$

where $$\mathbf{W} = \begin{bmatrix} — & \mathbf{v}_1^T & — \\ — & \mathbf{v}_2^T & — \\ & \vdots & \\ — & \mathbf{v}_V^T & — \end{bmatrix}_{V \times d}$$

Example:

$$\mathbf{h}_i = \begin{bmatrix} | & | & & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \dots & \mathbf{v}_V \\ | & | & & | \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{v}_2 \\ | \end{bmatrix}$$

- Word vector is the same as the corresponding row of the Weight matrix

Second Layer

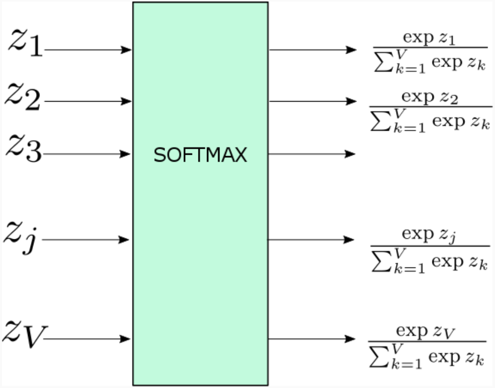
Assume $$\mathbf{U} = \begin{bmatrix} — & \mathbf{u}_1^T & — \\ — & \mathbf{u}_2^T & — \\ & \vdots & \\ — & \mathbf{u}_V^T & — \end{bmatrix}_{V \times d}$$

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_j \\ \vdots \\ z_V \end{bmatrix} = \mathbf{U}\mathbf{h} = \begin{bmatrix} — & \mathbf{u}_1^T & — \\ & \vdots & \\ — & \mathbf{u}_j^T & — \\ & \vdots & \\ — & \mathbf{u}_V^T & — \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_d \end{bmatrix}$$

- j^{th} component of $\mathbf{z}$ is given by

$$z_i(j) = \mathbf{u}_j^T \mathbf{h}_i \qquad (2)$$

Softmax



$$P(y_i = j|\mathbf{x}_i) = \frac{\exp(z_i(j))}{\sum_{k=1}^{V} \exp(z_i(k))} \tag{3}$$

Loss Function

- Loss:

$$L = \prod_{i=1}^{M} P(y_i|\mathbf{x}_i) = \prod_{i=1}^{M} \frac{\exp(z_i(y_i))}{\sum_{k=1}^{V} \exp(z_i(k))}$$

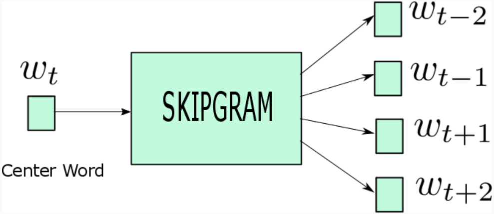
- Log loss:

$$E = -\log L = \sum_{i=1}^{M} \left[-z_i(y_i) + \log \sum_{k=1}^{V} \exp(z_i(k)) \right] \tag{4}$$

Gradients and Back-Propagation

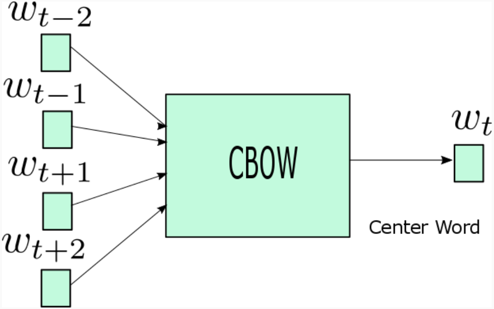
- Differentiate equation 4 wrt. $z_i(j) \Rightarrow \frac{\partial E}{\partial z_i(j)}$
- Differentiate equation 2
  - wrt $\mathbf{u}_j \Rightarrow \dfrac{\partial z_i\,(j)}{\partial \mathbf{u}_j}$
  - wrt. $\mathbf{h}_i \Rightarrow \dfrac{\partial z_i\,(j)}{\partial h_i}$
- Differentiate equation 1 wrt $\mathsf{W} \Rightarrow \dfrac{\partial h_i}{\partial \mathsf{W}}$
- By using the chain rule (i.e.) Back-propagation, we can find $\dfrac{\partial E}{\partial \mathbf{u}_j}$ and $\dfrac{\partial E}{\partial W}$

Skip-gram



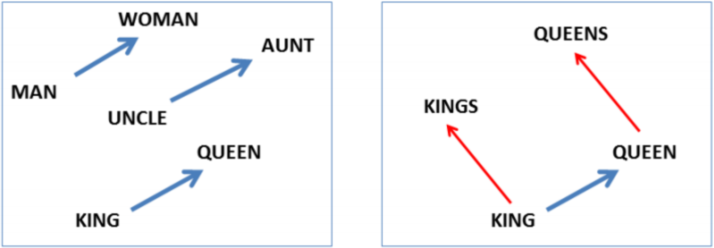
- if data are $(w_1, w_2, w_3, w_4, \cdots w_T)$, then assuming a context window of 2, the training word pairs will be $\{(w_1, w_2), (w_1, w_3), (w_2, w_1), (w_2, w_3), (w_2, w_4), \cdots\}$

Continuous Bag of Words (CBOW)



- if data are $(w_1, w_2, w_3, w_4, \cdots w_T)$, then assuming a context window of 2, the training word pairs will be $\{(w_2, w_1), (w_3, w_1), (w_1, w_2), (w_3, w_2), (w_4, w_2), \cdots\}$

Word Vector Visualisation



(Mikolov et al., NAACL HLT, 2013)

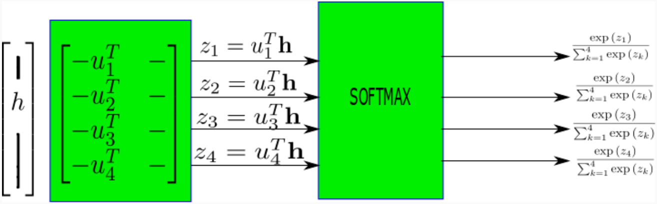
Problem of Efficient Training

- Typical vocabularies are very large (couple of 100k)
- Word pairs make it even larger (millions)
- The cost of calculating Softmax its derivatives is high

$$P(y_i = j|\mathbf{x}_i) = \frac{\exp(z_i(j))}{\sum_{k=1}^{V} \exp(z_i(k))}$$

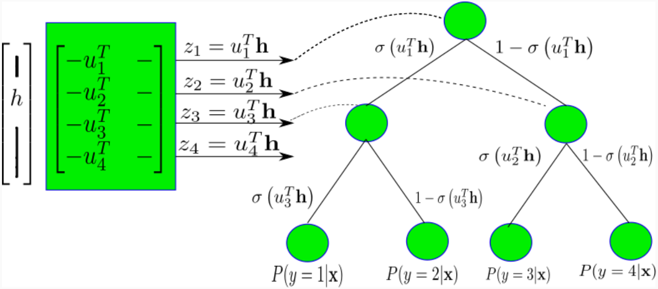
- Solutions
  - Hierarchical Softmax
  - Noise Contrastive Estimation
  - Negative sampling

Another view of Softmax



- Each output depends on all z

Hierarchical Softmax



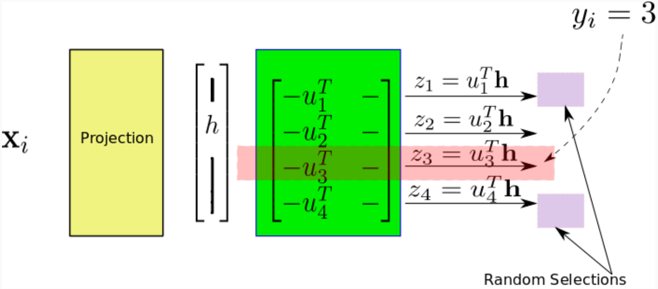
$$\boxed{\sigma(x) = \frac{1}{1+\exp(-x)}}$$

- Each output depends on only z in its path
- Exampl: $P(y=3|x) = \left[1-\sigma\left(u_1^T h\right)\right]\sigma\left(u_2^T h\right)$
- This works because sum of probabilities corresponding to all paths is 1

Hierarchical Softmax-Challenges

- Designing a suitable tree is not trivial

Noise Contrastive Estimation (NCE)



- A sampling based approach (i.e. random approximation)
- Instead of using ALL alternative words, choose some random words
- Then cast the estimation problem as a classification problem

Noise Contrastive Estimation

- Consider a garbage data set in addition to the genuine dataset.
- Consider a given input (one-hot encoded) vector $\mathbf{x}$ and draw:
  - One genuine data sample $\{(\mathbf{x}, y^d)\}$, y^d is the correct output class drawn from the data distribution $P_d(y|\mathbf{x})$
  - k garbage data samples $\{(\mathbf{x}, y_i^n)\}$, y_i^n is randomly chosen output class from a noise distribution $P_n(y)$.
- Now we have $\{(\mathbf{x}, y^d), (\mathbf{x}, y_1^n), (\mathbf{x}, y_2^n), \cdots, (\mathbf{x}, y_k^n)\}$
- Now we consider classification of each sample to either noise or data

Noise Contrastive Estimation

-

$$P\left(\text{data}|\mathbf{x}, y\right) = \frac{P(y|\text{data}, \mathbf{x})P(\text{data})}{P(y|\mathbf{x})} \tag{5}$$

$$= \frac{P(y|\text{data}, \mathbf{x})P(\text{data})}{P(y|\text{data}, \mathbf{x})P(\text{data}) + P(y|\text{noise}, \mathbf{x})P(\text{noise})} \tag{6}$$

$$= \frac{P(y|\text{data}, \mathbf{x})\frac{1}{1+k}}{P(y|\text{data}, \mathbf{x})\frac{1}{1+k} + P(y|\text{noise}, \mathbf{x})\frac{k}{1+k}} \tag{7}$$

$$= \frac{P_d(y|\mathbf{x})}{P_d(y|\mathbf{x}) + kP_n(y)} \tag{8}$$

- And

$$P\left(\text{noise}|\mathbf{x}, y\right) = 1 - \frac{P_d(y|\mathbf{x})}{P_d(y|\mathbf{x}) + kP_n(y)} = \frac{kP_n(y)}{P_d(y|\mathbf{x}) + kP_n(y)} \tag{9}$$

Noise Contrastive Estimation Loss Function

- Loss

$$L = P\left(\text{data}|\mathbf{x}, y^d\right) \prod_{j=1}^{k} P\left(\text{noise}|\mathbf{x}, y_j^n\right)$$

- Log loss

$$E = \log\left[\frac{P_d(y^d|\mathbf{x})}{P_d(y^d|\mathbf{x}) + kP_n(y^d)}\right] + \sum_{j=1}^{k} \log\left[\frac{kP_n(y_j^n)}{P_d(y_j^n|\mathbf{x}) + kP_n(y_j^n)}\right]$$

- We choose a noise distribution, so $P_n(y)$ terms can be calculated.
- How to compute $P_d(y|\mathbf{x})$?

Computation of Data Distribution

- Assume data distribution is computed by your network:

$$P_d\left(y|\mathbf{x}\right) = \frac{\exp\left(z\left(y\right)\right)}{\sum_{j=1}^{V} \exp\left(z\left(j\right)\right)} = \frac{\exp\left(z\left(y\right)\right)}{Z\left(\mathbf{x}\right)}$$

- But now we are back to the original problem, how to calculate $Z(\mathbf{x})$
- Solution: Consider it to be a parameter and try to learn it on data. In practice, the learned $Z(\mathbf{x})$ is close to 1.
- Therefore:

$$P_d\left(y|\mathbf{x}\right) = \exp\left(z\left(y\right)\right)$$

Final NCE Loss function

- NCE loss function

$$E_{NCE} = \log\left[\frac{\exp\left(z\left(y^{d}\right)\right)}{\exp\left(z\left(y^{d}\right)\right) + kP_n(y^{d})}\right] + \sum_{j=1}^{k} \log\left[\frac{kP_n(y_j^{n})}{\exp\left(z\left(y_j^{n}\right)\right) + kP_n(y_j^{n})}\right] \quad (10)$$

- To learn the parameters, find $\frac{\partial E_{NCE}}{\partial z(l)}$ and back-propagate through the network.

What we achieve through NCE?

- Faster than softmax.
- It can be shown that

$$\frac{\partial E_{NCE}}{\partial \theta} \rightarrow \frac{\partial E_{SOFTMAX}}{\partial \theta}$$

when $k \rightarrow \infty$ where θ is a parameter of the network.

Negative Sample Loss

- Yet another approximation
- Assume $kP_n(y) = 1$, for any y. That means a uniform noise distribution and $k = \frac{1}{V}$
- Substitute this in NCE loss function (equation 10)

$$E_{NEG} = \log\left[\frac{\exp\left(z\left(y^d\right)\right)}{\exp\left(z\left(y^d\right)\right) + 1}\right] + \sum_{j=1}^{k} \log\left[\frac{1}{\exp\left(z\left(y_j^n\right)\right) + 1}\right] \quad (11)$$

- Using sigmoid $\sigma(x) = \dfrac{1}{1 + \exp(-x)}$ we can write this as

$$E_{NEG} = \log\left[\sigma\left(z\left(y^d\right)\right)\right] + \sum_{j=1}^{k} \log\left[\sigma\left(-z\left(y_j\right)^n\right)\right] \quad (12)$$

Global Vectors for Word Prediction (GloVe) Algorithm

- Two types of word embedding algorithms:
  - Word counting based
  - Prediction based (Skip-gram, CBOW)
- GloVe tries to combine best of both worlds

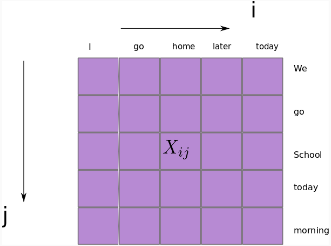
GLOVE Algorithm

- It tries to optimize

$$J = \sum_{i,j=1}^{V} f\left(x_{ij}\right)\left(\mathbf{w}_i^T \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

where

  - $\mathbf{w}_i^T$ and $\tilde{\mathbf{w}}_j$ are word vectors of i^{th} and j^{th} words
  - X_{ij} is word co- occurrence count of i^{th} and j^{th} words
  - $f(X_{ij})$ is a weighting function.
  - b_i and $\tilde{b}_j$ are biases.



Beyond Word Vectors

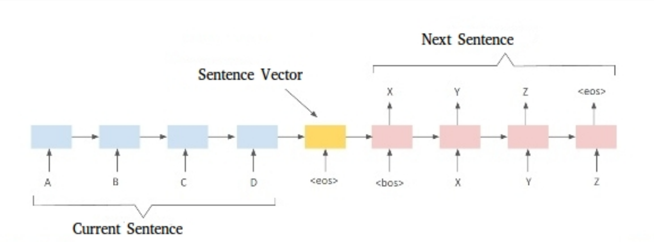
- Word2vec (Skip-gram, CBOW) and GloVe algorithms:
    - are based on shallow models.
    - do not result in universal embeddings (i.e. do not learn higher level abstractions)
    - operate on word level
    - Unsupervised learning
- Newer directions
    - Character level embedding
    - Sentence level embedding
    - Universal embedding (incorporate higher level information)
    - Supervised learning with syntactic/semantic supervision
- Examples: Fasttext, Skip-thoughts, ELMo, CoVe

FastText

- Character level embedding system
- Represent words as character N-grams
    - Example: 3-gram of word <where>
    - <wh, whe, her, ere re>
- Generate embedding vectors for N-grams and represent word with weighted sum of N-gram vectors

Skip-thought vectors

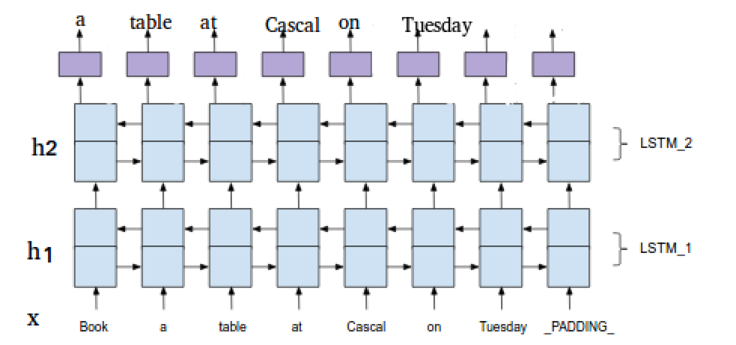
- Sentence level embedding (i.e. Each sentence is represented by a fixed length vector
- Word order is taken into account. eg: ’Rosenborg **beat** Brann’ vs ’Brann **beat** Rosenborg’
- Need semantically related sentences.
- Tries to predict the next and previous sentences from the current sentence



ELmo-Embedding from Language Models

- Embedding at word level, but the order is taken into account
- Better handling *Polysemy* (i.e. same word having different meanings in different contexts)
- Tries to predict the next word given the previous words, in both forward and backward directions
  - Sentence: *I like deep learning very much*
  - Forward: Given *I like deep* predict *learning*
  - Backward: Given *much very* predict *learning*
- Uses a stacked bidirectional LSTM

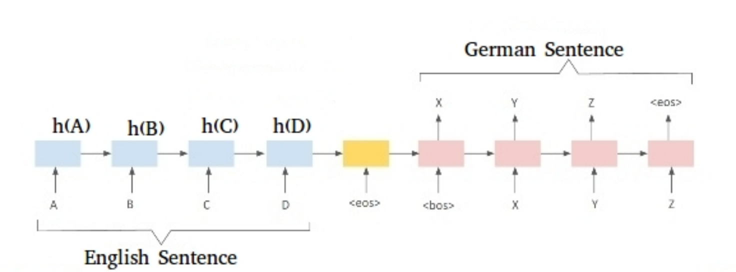
ELmo Architecture



- For each word **x**, the embedded vector is a weighted sum of all the corresponding LSTM outputs, $\sum_{j=0}^{L} s_j \mathbf{h}_j$. Here $\mathbf{h}_j$ is a concatenation of the forward and backward LSTMs, $\mathbf{h}_j = \left[\mathbf{h}_j^f, \mathbf{h}_j^b\right]$

CoVe- Contextualized Word vectors

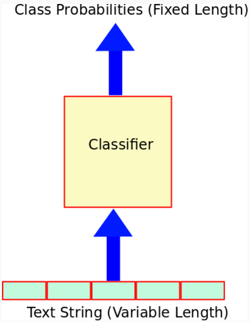
- CoVe uses a encoder-decoder architecture for language translation
- CoVe is supervised (i.e. need labeled database
- Embeded vectors are simply the hidden states of the decoder



Text Classification

Text Classification Big Picture

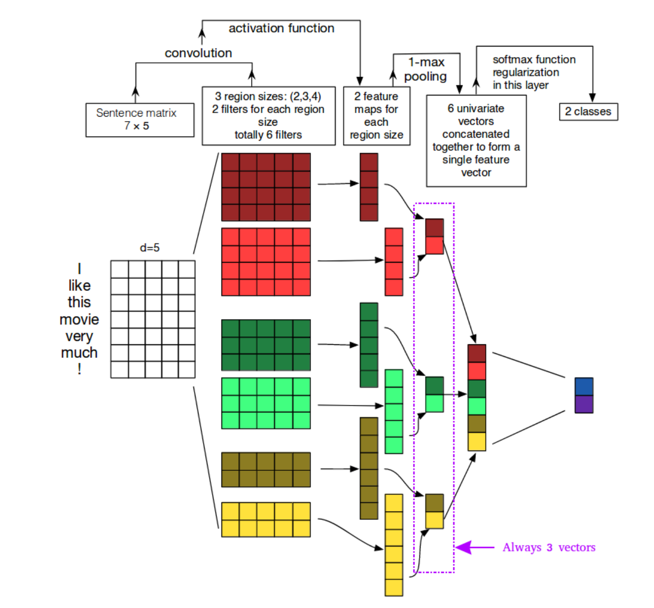
- Main challenge: Map a variable length input to a fixed length output
- Different applications (eg: classification of E-mail, SMS, Web contents in tagging, CRM, marketing, sentiment analysis.
    - Sentence classification
    - Document classification



Basic Tools

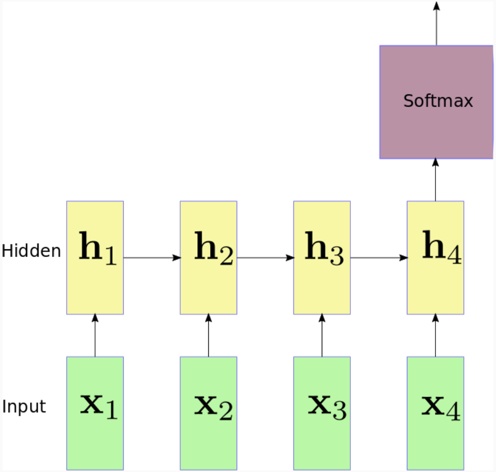
- Convolutional Neural Networks (CNN):
  - Seem less natural
  - But possible with a trick to have a fixed length output irrespective of the input size
- Recurrent Neural Networks (RNN):
  - Naturally suitable for variable length inputs
  - Often used with attention

CNN based Sentence Classification

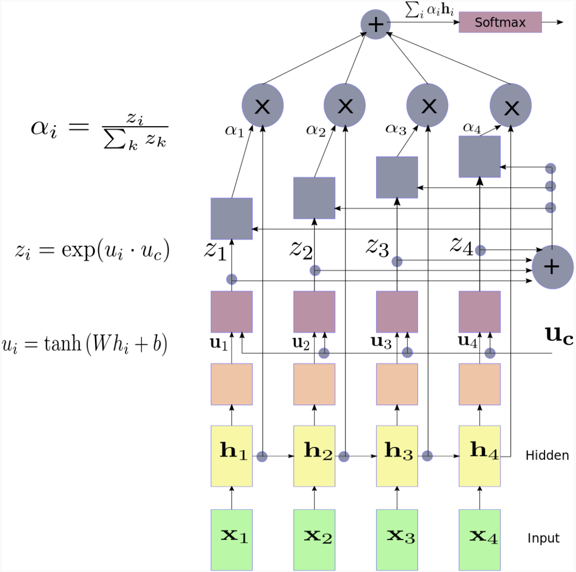


RNN for sentence classification

- Use many-to-one configuration



RNN with Attention



Hierarchical Attention Network