IN5550: Neural Methods in Natural Language Processing
Sub-lecture 3.3
Practicalities and hyper-parameters

Andrey Kutuzov

University of Oslo

7 February 2023

Contents

Practicalities

How deep should our networks be?

Practicalities

It depends...

- ...on what your task is?

Practicalities

It depends...

- ...on what your task is?
  - Computer vision often uses models that are hundreds of layers deep

Practicalities

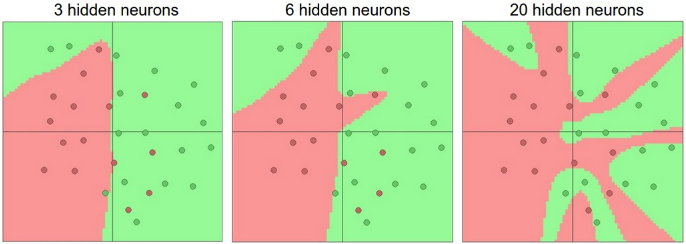
It depends...

- ...on what your task is?
  - Computer vision often uses models that are hundreds of layers deep
  - The number of layers in NLP varies between 1-2 up to 24 (*BERT*, [Devlin et al., 2019]) or even 78 (*Turing-NLG*).
- The strongest NLP models are still growing in depth, but it's not entirely clear how much extreme depth benefits.

Practicalities

The deeper and larger the model, the more capacity (parameters), the more complex functions it can approximate.

The deeper and larger the model, the more capacity (parameters), the more complex functions it can approximate.



Vanishing Gradients

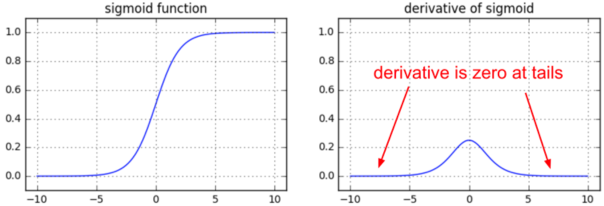
Practicalities

Vanishing Gradients

- As a gradient flows through deep neural networks, it can tend towards zero (vanishing gradient)

Vanishing Gradients

- As a gradient flows through deep neural networks, it can tend towards zero (vanishing gradient)
- Recall the sigmoid function and its derivative...

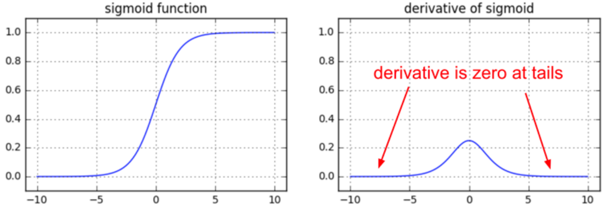


- Back-propagation computes the gradients by the chain rule ...

Practicalities

Vanishing Gradients

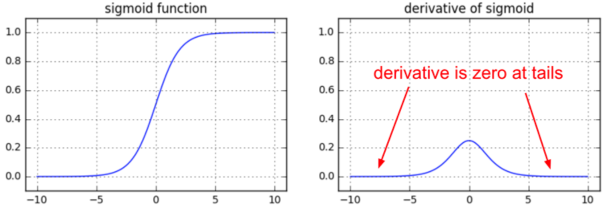
- As a gradient flows through deep neural networks, it can tend towards zero (vanishing gradient)
- Recall the sigmoid function and its derivative...



- Back-propagation computes the gradients by the chain rule ...
- You are effectively multiplying n of these small numbers to compute gradients for the early layers of an n-layer network

Vanishing Gradients

- As a gradient flows through deep neural networks, it can tend towards zero (vanishing gradient)
- Recall the sigmoid function and its derivative...

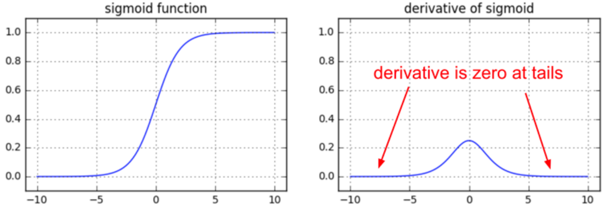


- Back-propagation computes the gradients by the chain rule ...
- You are effectively multiplying n of these small numbers to compute gradients for the early layers of an n-layer network
- The size of the gradient decreases exponentially with n

Practicalities

Vanishing Gradients

- As a gradient flows through deep neural networks, it can tend towards zero (vanishing gradient)
- Recall the sigmoid function and its derivative...



- Back-propagation computes the gradients by the chain rule ...
- You are effectively multiplying n of these small numbers to compute gradients for the early layers of an n-layer network
- The size of the gradient decreases exponentially with n
- The model learns very slow, or stops learning completely.

Practicalities

Exploding Gradients

Exploding Gradients

- If instead you have a non-linearity whose derivatives can take larger values...

Exploding Gradients

- If instead you have a non-linearity whose derivatives can take larger values...
- the gradients can become overly large (explode) and training updates will make overly large changes to parameters.

Exploding Gradients

- If instead you have a non-linearity whose derivatives can take larger values...
- the gradients can become overly large (explode) and training updates will make overly large changes to parameters.
- Learning becomes highly unstable and in practice it is impossible to optimize well

Practicalities

How can we solve these problems?

Practicalities

1. Choose the non-linearity of your hidden layers wisely

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* $>$ *tanh* $>$ *sigmoid*

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* $>$ *tanh* $>$ *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* $>$ *tanh* $>$ *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* > *tanh* > *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters
3. Make the network shallower

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* > *tanh* > *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters
3. Make the network shallower
4. Step-wise training (first train lower then further layers)

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* > *tanh* > *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters
3. Make the network shallower
4. Step-wise training (first train lower then further layers)
5. Batch normalization

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* > *tanh* > *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters
3. Make the network shallower
4. Step-wise training (first train lower then further layers)
5. Batch normalization
6. Scheduled learning rate

Practicalities

1. Choose the non-linearity of your hidden layers wisely
   - *relu* > *tanh* > *sigmoid*
   - many new options: *maxout*, *gelu*, *elu*, *geglu* [Shazeer, 2020]
2. Be careful with initialization parameters
3. Make the network shallower
4. Step-wise training (first train lower then further layers)
5. Batch normalization
6. Scheduled learning rate
7. Use special gradient-preserving architectures (*LSTM*, *GRU*, coming soon)

Practicalities

Optimization

- Which optimization algorithm to use?

Practicalities

Optimization

- Which optimization algorithm to use?
- *SGD* works well but may be slow to converge

Practicalities

Optimization

- Which optimization algorithm to use?
- *SGD* works well but may be slow to converge
- *AdaGrad*, *AdamW*, etc are good alternatives

Practicalities

Optimization

- Which optimization algorithm to use?
- *SGD* works well but may be slow to converge
- *AdaGrad*, *AdamW*, etc are good alternatives
- Depending on the task, *SGD* may still perform better

Practicalities

Optimization

- Which optimization algorithm to use?
- *SGD* works well but may be slow to converge
- *AdaGrad*, *AdamW*, etc are good alternatives
- Depending on the task, *SGD* may still perform better

```
from torch import optim

optimizer = optim.SGD(model.params(), lr=0.01)
optimizer = optim.AdamW(model.params(), lr=0.01)
optimizer = optim.Adagrad(model.params(), lr=0.01)
optimizer = optim.LBFGS(model.params(), lr=1)
```

Practicalities

Initialization

- We normally randomly initialize the parameters θ

Practicalities

Initialization

- We normally randomly initialize the parameters θ
- … but the magnitude of the random values still has a large impact on the learning process

Initialization

- We normally randomly initialize the parameters θ
- ... but the magnitude of the random values still has a large impact on the learning process
- Xavier initialization:

$$W \backsim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right] \tag{1}$$

Practicalities

Initialization

- We normally randomly initialize the parameters θ
- ... but the magnitude of the random values still has a large impact on the learning process
- Xavier initialization:

$$W \backsim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right] \tag{1}$$

- He initialization:

$$W \backsim \mathcal{N}(\mu, \sigma), \ \sigma = \sqrt{\frac{2}{d_{in}}} \tag{2}$$

Practicalities

Initialization

- We normally randomly initialize the parameters θ
- ... but the magnitude of the random values still has a large impact on the learning process
- Xavier initialization:

$$W \backsim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, +\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right] \tag{1}$$

- He initialization:

$$W \backsim \mathcal{N}(\mu, \sigma), \ \sigma = \sqrt{\frac{2}{d_{in}}} \tag{2}$$

- In PyTorch, the initialization depends on the kind of layer that you are instantiating. Have a look at the documentation: `https://pytorch.org/docs/stable/nn.init.html`

Practicalities

Restarts and Ensembles

- since we are using random initializations, we may want to check various initializations

Practicalities

Restarts and Ensembles

- since we are using random initializations, we may want to check various initializations
- you can then report the average model accuracy across all restarts

Practicalities

Restarts and Ensembles

- since we are using random initializations, we may want to check various initializations
- you can then report the average model accuracy across all restarts
- gives an idea of model stability

Practicalities

Restarts and Ensembles

- since we are using random initializations, we may want to check various initializations
- you can then report the average model accuracy across all restarts
- gives an idea of model stability
- Ensembles often increase prediction accuracy

Practicalities

Restarts and Ensembles

- since we are using random initializations, we may want to check various initializations
- you can then report the average model accuracy across all restarts
- gives an idea of model stability
- Ensembles often increase prediction accuracy
- Note: use a fixed random seed for reproducibility when comparing different data or hyper-parameters.

Practicalities

Shuffling

- The order in which the training examples are presented is important
- It is recommended to shuffle the training data before each training epoch

Practicalities

Learning rate

- Large learning rates will prevent convergence
- Small learning rates will take too long to converge

Practicalities

Minibatch size

- When using batch SGD, you have to decide on batch size
- Large batches are sometimes helpful (depends on task)
- Also computationally efficient

Practicalities

Regularization

- Like in linear models, we need regularization to avoid over-fitting;

Practicalities

Regularization

- Like in linear models, we need regularization to avoid over-fitting;
- most popular options for deep neural networks:
    1. L_2 norm (*weight decay*);

Practicalities

Regularization

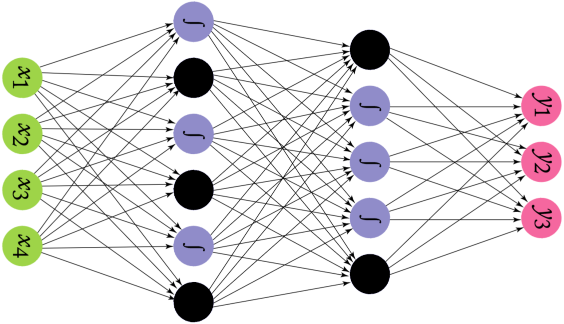
- Like in linear models, we need regularization to avoid over-fitting;
- most popular options for deep neural networks:
  1. L_2 norm (*weight decay*);
  2. Dropout [Srivastava et al., 2014] (very effective).

Practicalities

Regularization

- Like in linear models, we need regularization to avoid over-fitting;
- most popular options for deep neural networks:
  1. L_2 norm (*weight decay*);
  2. Dropout [Srivastava et al., 2014] (very effective).

Dropout simply zeroes out neurons in the layers (e.g., 50%) in each forward pass randomly:



So how many different hyper-parameters can we possibly have for deep feed-forward neural networks?

Practicalities

1. Depth (number of hidden layers)

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity
4. Initialization strategy

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity
4. Initialization strategy
5. Normalization (dropout, l_2, batch norm)

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity
4. Initialization strategy
5. Normalization (dropout, l_2, batch norm)
6. Optimization algorithm (SGD, Adam, AdaGrad)

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity
4. Initialization strategy
5. Normalization (dropout, l_2, batch norm)
6. Optimization algorithm (SGD, Adam, AdaGrad)
7. Learning rate (initial, annealing, warmup, decay etc.)

Practicalities

1. Depth (number of hidden layers)
2. Width (number of neurons per hidden layer)
3. Non-linearity
4. Initialization strategy
5. Normalization (dropout, l_2, batch norm)
6. Optimization algorithm (SGD, Adam, AdaGrad)
7. Learning rate (initial, annealing, warmup, decay etc.)
8. Batch size
9. etc...

How can we possibly choose the best values for all of these?

Practicalities

Most common strategies:

- Grid search...
- Random search...
- Bayesian search...

Practicalities

Most common strategies:

- Grid search...
- Random search...
- Bayesian search...
- ...black magic
  (a.k.a. 'trials and errors').

Most common strategies:

- Grid search...
- Random search...
- Bayesian search...
- ...black magic
  (a.k.a. 'trials and errors').

Practicalities

How is that all implemented in code?
Computation graph (sub-lecture 3.4).

References I

- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (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.

- Shazeer, N. (2020).
  GLU variants improve transformer.
  CoRR, abs/2002.05202.

- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).
  Dropout: a simple way to prevent neural networks from overfitting.
  The Journal of Machine Learning Research, 15(1):1929–1958.