Experimental methodology for training of (deep) machine learning models

Deep learning course for industry

Mitko Veta

Eindhoven University of Technology
Department of Biomedical Engineering

2020

Learning goals

- Introduce additional regularisation techniques.
- Discuss the proper experimental setup for model selection and training.
- Discuss some practical considerations about training deep neural networks.
- (bonus) The backpropagation algorithm.

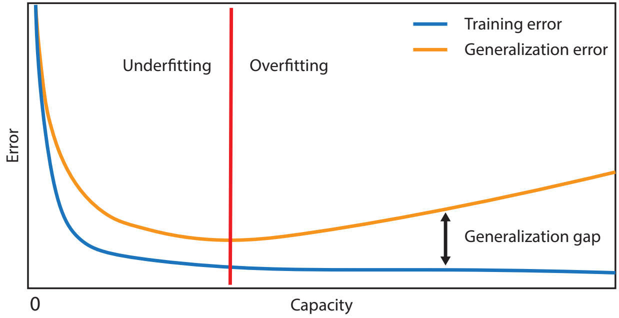
Prof. Lena Fraunhofer, King's College London
"On the application of generative deep learning methods in medical image analysis"

Today 17.30, Metaforum zaal 13

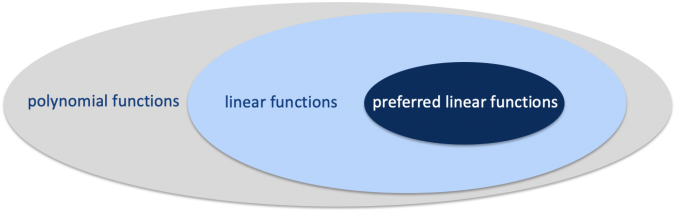
Regularisation

Regularisation

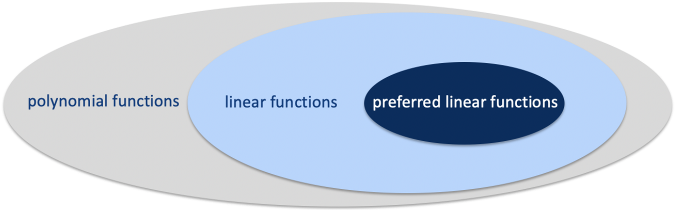
Regularisation is any modification to a learning algorithm intended to reduce its generalisation error but not its training error.



Regularisation



Regularisation



Prefer smaller weights $\mathbf{w}$:

$$L(\mathbf{w}) = RSS(\mathbf{w}) + \lambda \mathbf{w}^{\intercal}\mathbf{w}$$

Training set size

- It is possible for the model to have optimal capacity and still have a large gap between training and generalisation errors.
- In that case the gap usually can be reduced with increasing the number of training examples.

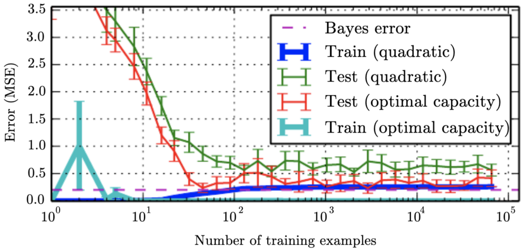
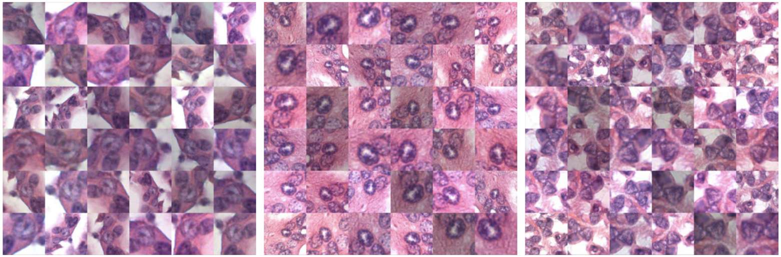


Figure source: deeplearningbook.org

Data augmentation

Data augmentation: create new, plausible examples by transforming existing examples.



Data augmentation

Which augmentations to use?

This is very problem dependent. Affine geometric transformations and intensity transformations such as contrast, brightness and colour are very standard.

Note that some transformations can change the class of the example (e.g. rotating an image of the number 6 by 180 degrees).

Model averaging

Rationale: different models will make different errors. Averaging different models can improve generalisation.

What is a "different model"? $\rightarrow$ different training set, different neural network architecture, different initialisation...

Model averaging

Rationale: different models will make different errors. Averaging different models can improve generalisation.

What is a “different model”? $\rightarrow$ different training set, different neural network architecture, different initialisation...

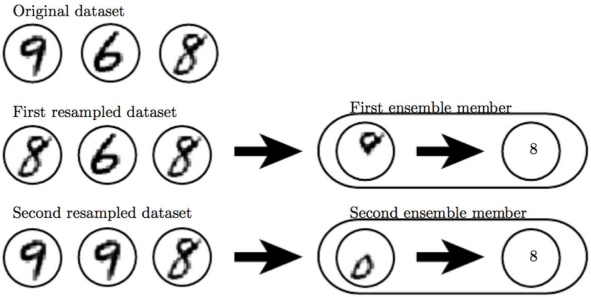


Figure source: deeplearningbook.org

Dropout

Dropout performs implicit model averaging by randomly turning off connections during the training of the neural network.

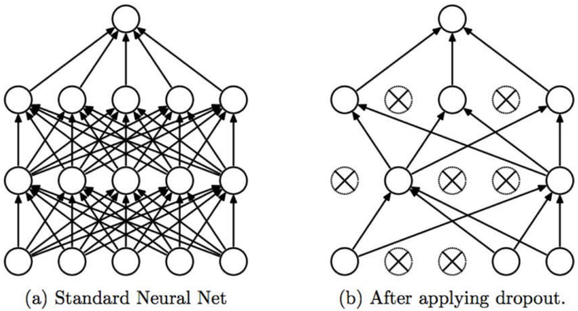


Figure source: deeplearningbook.org

Parameter initialisation

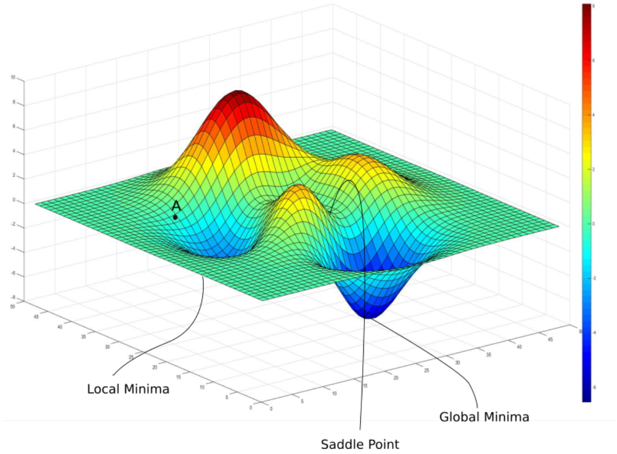
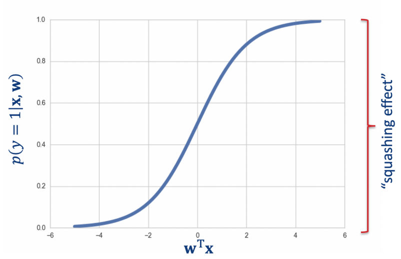


Figure source: blog.paperspace.com

Parameter initialisation

Parameters are usually initialised with small random numbers, e.g. drawn from a uniform distribution $[-s, s]$.

The scale s of the initialisation is very important for the optimisation procedure. Poorly chosen s can "stuck" the training process (some or all of the parameters will not be updated).



Batch normalisation

This technique makes the training less sensitive to the parameter initialisation and as a side-effect acts as a regulariser.

During each training update, the input of each layer is normalised as follows:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\text{batch}}}{\sqrt{\sigma^2_{\text{batch}} + \epsilon}}$$

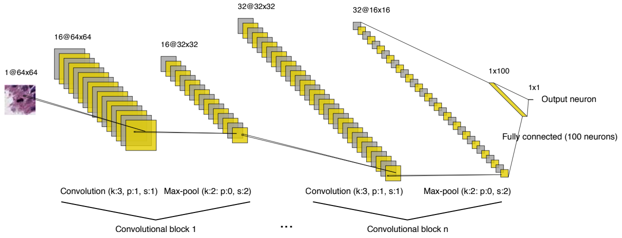
$$\text{BN}_{\gamma,\beta}(x_i) = \gamma \hat{x}_i + \beta$$

where x_i is the output of the layer and μ_{batch} and σ_{batch} are the mean and standard deviation of the current batch.

Transfer learning

Take the weights of an existing network trained for one problem and either:

- Use the network weights as initialisation for a model for a different problem.
- Use the network weights for feature extraction in combination with another classifier.



The “holy trinity” of datasets

Hyperparameters

Hyperparameters are settings that can be used to control the behaviour of the algorithm.

In general, **the hyperparameters are not modified by the learning algorithm itself.**

A setting can be chosen to be a hyperparameter when it is difficult to optimise or, more often, when its derivation from the training set can lead to overfitting.

Hyperparameters

Hyperparameters are settings that can be used to control the behaviour of the algorithm.

In general, **the hyperparameters are not modified by the learning algorithm itself.**

A setting can be chosen to be a hyperparameter when it is difficult to optimise or, more often, when its derivation from the training set can lead to overfitting.

Example: In polynomial regression the degree of the polynomial is a capacity hyperparameter.

Hyperparameters

Question: should the hyperparameters be chosen based on the performance on the testing set?

Validation set

The **validation set** is used during training to predict the behaviour (generalisation error) of the algorithm on new data, i.e., on the test set and to chose the hyperparameters.

Ideally these two sets are disjoint.

The training data is split in two disjoint subsets.

One subset is used to learn the parameters of the algorithm and the other is the validation set.

The subset used to learn the parameters is still typically called a **training set**.

The “holy trinity” of datasets

Training	Validation	Test
Used to find the optimal **parameters** of the model.	Used to find the optimal **model** (hyper-parameters).	Used to estimate the **performance** of the optimal model.
w	$f(\cdot)$	$\|\|\hat{y} - y\|\|$

Validation set

Question: How large should the validation set be?

Validation set

Question: How large should the validation set be?

Since the validation set is used to determine the hyperparameters it will typically underestimate the generalisation error.

However, it will usually better predict the generalisation error than the training set.

After the completion of the hyperparameters optimisation we can estimate the generalisation error using the test data.

In practice the testing should be done also on different test data to avoid the test data becoming “stale”.

“Babysitting” the training process
and other practical considerations

Monitor the training and validation loss curves

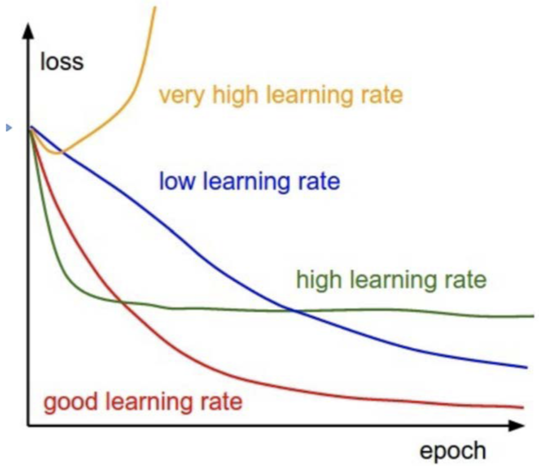
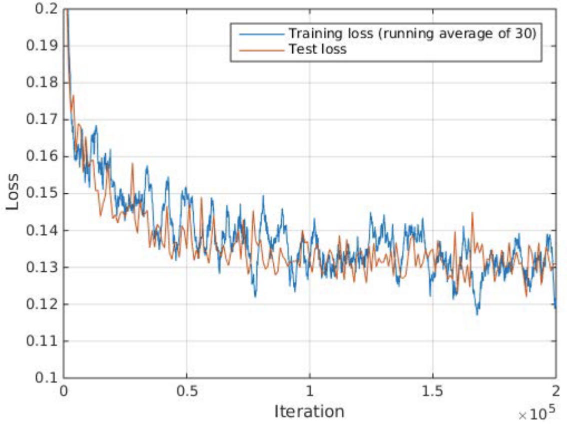
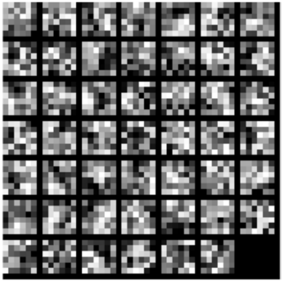
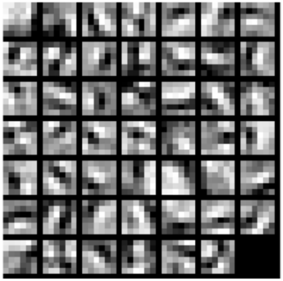


Figure from: deeplearning.net

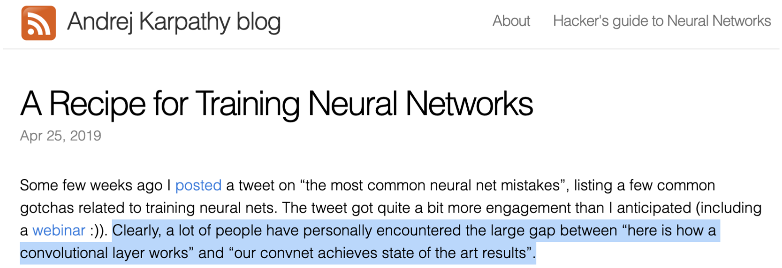
Real world training and validation curves



Inspect the kernels

Left: noisy kernels (something is/went wrong?), right: regular kernels.



Andrej Karpathy blog

About Hacker's guide to Neural Networks

A Recipe for Training Neural Networks

Apr 25, 2019

Some few weeks ago I posted a tweet on “the most common neural net mistakes”, listing a few common gotchas related to training neural nets. The tweet got quite a bit more engagement than I anticipated (including a webinar :)). Clearly, a lot of people have personally encountered the large gap between “here is how a convolutional layer works” and “our convnet achieves state of the art results”.

http://karpathy.github.io/2019/04/25/recipe/

Premise

- "Neural net training is a leaky abstraction'.'
  - Do not consider it plug-and-play.
- "Neural net training fails silently."
  - You will not get an error message if you do something wrong.

http://karpathy.github.io/2019/04/25/recipe/

Workflow

1. Pay attention to your data
2. Make a simple baseline

Workflow

1. Pay attention to your data
2. Make a simple baseline
3. Overfit
4. Regularize

Workflow

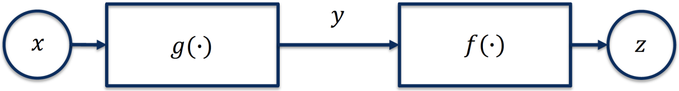
1. Pay attention to your data
2. Make a simple baseline
3. Overfit
4. Regularize
5. Tune the hyperparameters
6. Try all the remaining “tricks”

http://karpathy.github.io/2019/04/25/recipe/

Backpropagation

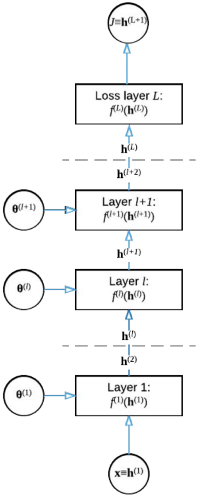
Chain rule of differentiation

$$z = f(y)$$
$$y = g(x)$$
$$z = f\big(g(x)\big)$$



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Backpropagation



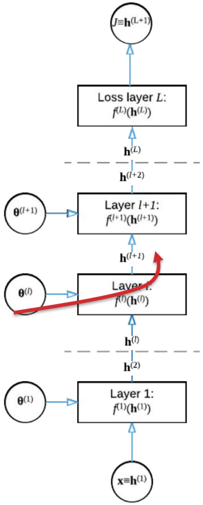
$J\equiv\mathbf{h}^{(L+1)}$
Loss layer L: $f^{(L)}(\mathbf{h}^{(L)})$
$\mathbf{h}^{(L)}$
$\mathbf{h}^{(l+2)}$
$\theta^{(l+1)}$
Layer $l+1$: $f^{(l+1)}(\mathbf{h}^{(l+1)})$
$\mathbf{h}^{(l+1)}$
$\theta^{(l)}$
Layer l: $f^{(l)}(\mathbf{h}^{(l)})$
$\mathbf{h}^{(l)}$
$\mathbf{h}^{(2)}$
$\theta^{(1)}$
Layer 1: $f^{(1)}(\mathbf{h}^{(1)})$
$\mathbf{x}\equiv\mathbf{h}^{(1)}$

$$J(\boldsymbol{\theta}) = f^{(L)}\left(\mathbf{h}^{(L)}\right)$$

$$\mathbf{h}^{(L)} = f^{(L-l)}\left(\mathbf{h}^{(L-1)}\right)$$

$$J(\boldsymbol{\theta}) = f^{(L)}\left(f^{(L-1)}(\ldots f^{(1)}(\mathbf{x}))\right)$$

Backpropagation

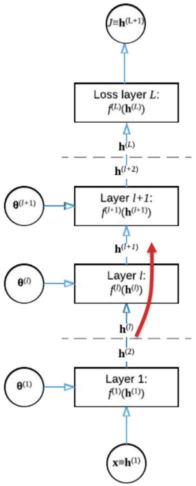


$J=\mathbf{h}^{(L+1)}$
Loss layer L:
$f^{(L)}(\mathbf{h}^{(L)})$
$\mathbf{h}^{(L)}$
$\mathbf{h}^{(l+2)}$
$\boldsymbol{\theta}^{(l+1)}$
Layer $l+1$:
$f^{(l+1)}(\mathbf{h}^{(l+1)})$
$\mathbf{h}^{(l+1)}$
$\boldsymbol{\theta}^{(l)}$
Layer l:
$f^{(l)}(\mathbf{h}^{(l)})$
$\mathbf{h}^{(l)}$
$\mathbf{h}^{(2)}$
$\boldsymbol{\theta}^{(1)}$
Layer 1:
$f^{(1)}(\mathbf{h}^{(1)})$
$\mathbf{x}=\mathbf{h}^{(1)}$

$$\frac{\partial J}{\partial \boldsymbol{\theta}^{(l)}} = \frac{\partial J}{\partial \mathbf{h}^{(l+1)}} \frac{\partial \mathbf{h}^{(l+1)}}{\partial \boldsymbol{\theta}^{(l)}}$$

$$\frac{\partial J}{\partial \boldsymbol{\theta}^{(l)}} = \underbrace{\frac{\partial J}{\partial \mathbf{h}^{(l+1)}}}_{?} \underbrace{\frac{\partial f^{(l)}(\mathbf{h}^{(l)})}{\partial \boldsymbol{\theta}^{(l)}}}_{\checkmark}$$

Backpropagation



$J\equiv\mathbf{h}^{(L+1)}$
Loss layer L:
$f^{(L)}(\mathbf{h}^{(L)})$
$\mathbf{h}^{(L)}$
$\mathbf{h}^{(l+2)}$
$\boldsymbol{\theta}^{(l+1)}$
Layer $l+1$:
$f^{(l+1)}(\mathbf{h}^{(l+1)})$
$\mathbf{h}^{(l+1)}$
$\boldsymbol{\theta}^{(l)}$
Layer l:
$f^{(l)}(\mathbf{h}^{(l)})$
$\mathbf{h}^{(l)}$
$\mathbf{h}^{(2)}$
$\boldsymbol{\theta}^{(1)}$
Layer 1:
$f^{(1)}(\mathbf{h}^{(1)})$
$\mathbf{x}\equiv\mathbf{h}^{(1)}$

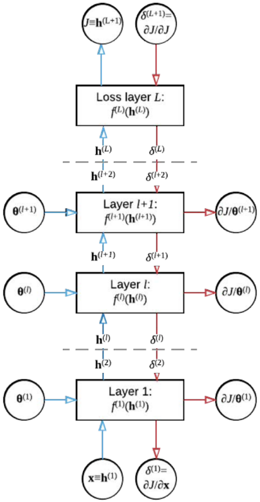
$$\frac{\partial J}{\partial \mathbf{h}^{(l)}} = \boldsymbol{\delta}^{(l)} = \frac{\partial J}{\partial \mathbf{h}^{(l+1)}} \frac{\partial \mathbf{h}^{(l+1)}}{\partial \mathbf{h}^{(l)}}$$

$$\boldsymbol{\delta}^{(l)} = \boldsymbol{\delta}^{(l+1)} \frac{\partial \mathbf{h}^{(l+1)}}{\partial \mathbf{h}^{(l)}}$$

$$\boldsymbol{\delta}^{(l)} = \boldsymbol{\delta}^{(l+1)} \frac{\partial f^{(l)}(\mathbf{h}^{(l)})}{\partial \mathbf{h}^{(l)}}$$

✓

Backpropagation



$J = \mathbf{h}^{(L+1)}$
$\delta^{(L+1)} = \partial J / \partial J$
Loss layer L: $f^{(L)}(\mathbf{h}^{(L)})$
$\mathbf{h}^{(L)}$
$\delta^{(L)}$
$\mathbf{h}^{(l+2)}$
$\delta^{(l+2)}$
$\boldsymbol{\theta}^{(l+1)}$
Layer $l+1$: $f^{(l+1)}(\mathbf{h}^{(l+1)})$
$\partial J / \boldsymbol{\theta}^{(l+1)}$
$\mathbf{h}^{(l+1)}$
$\delta^{(l+1)}$
$\boldsymbol{\theta}^{(l)}$
Layer l: $f^{(l)}(\mathbf{h}^{(l)})$
$\partial J / \boldsymbol{\theta}^{(l)}$
$\mathbf{h}^{(l)}$
$\delta^{(l)}$
$\mathbf{h}^{(2)}$
$\delta^{(2)}$
$\boldsymbol{\theta}^{(1)}$
Layer 1: $f^{(1)}(\mathbf{h}^{(1)})$
$\partial J / \boldsymbol{\theta}^{(1)}$
$\mathbf{x} = \mathbf{h}^{(1)}$
$\delta^{(1)} = \partial J / \partial \mathbf{x}$

Forward computation:

$$\mathbf{h}^{(l+1)} = f^{(l)}(\mathbf{h}^{(l)})$$

Backward computation:

$$\frac{\partial J}{\partial \boldsymbol{\theta}^{(l)}} = \boldsymbol{\delta}^{(l+1)} \frac{\partial f^{(l)}(\mathbf{h}^{(l)})}{\partial \boldsymbol{\theta}^{(l)}}$$

$$\boldsymbol{\delta}^{(l)} = \boldsymbol{\delta}^{(l+1)} \frac{\partial f^{(l)}(\mathbf{h}^{(l)})}{\partial \mathbf{h}^{(l)}}$$

Summary

- There is a variety of regularisation techniques, they are often used in combination.
- Always use the “holy trinity” of training, validation and test subsets.
- Training of neural networks is rarely plug-and-play. There is a large gap between “here is how a convolutional layer works” and “our convnet achieves state of the art results”.