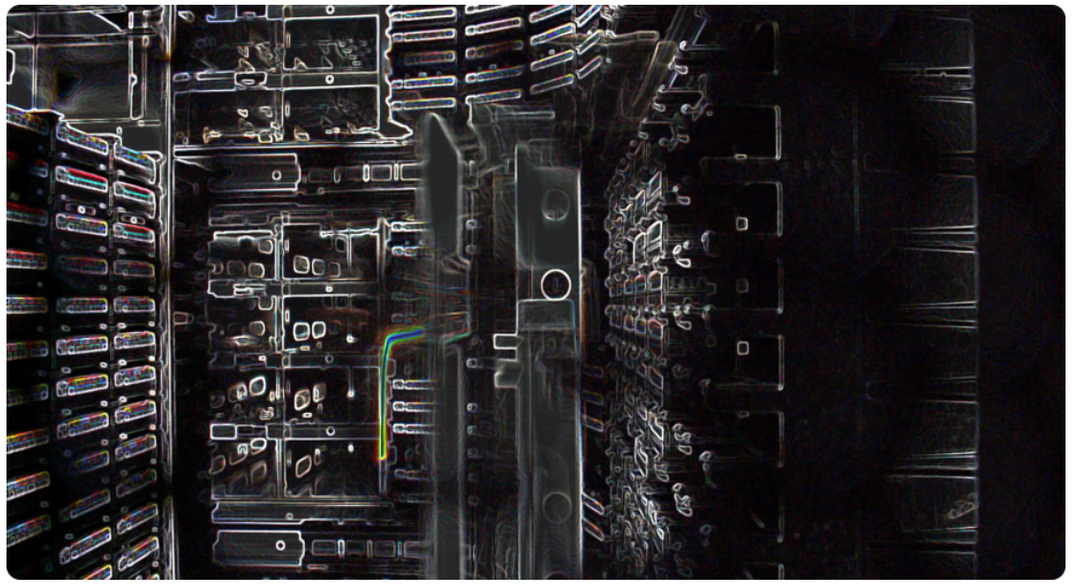
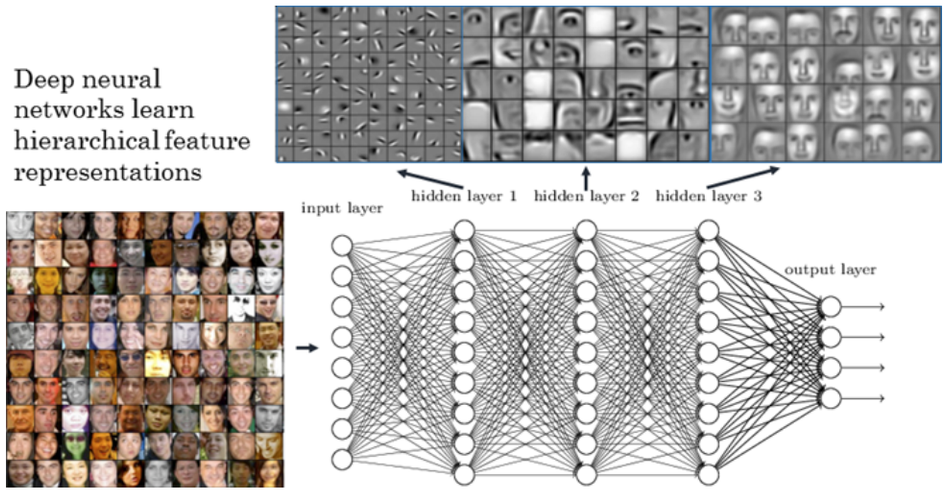


Deep learning

Deep learning with Keras

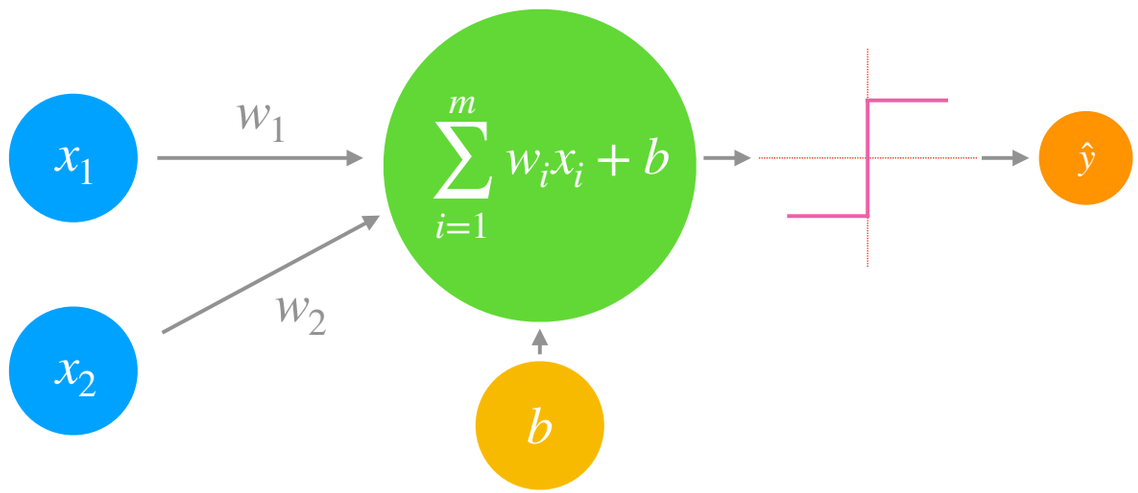
Recap / Questions?



- The environment @ dba.projects.sda.surfsara.nl
- If you have any questions, post them @ Canvas

- Deep learning is a subfield of machine learning in which the model learns to represent the data using many levels of concept hierarchies.
- Neural networks are made from nodes and edges which combine into layers. The network maps the input x to an output y.

Recap / Questions?



- The environment @ dba.projects.sda.surfsara.nl
- If you have any questions, post them @ Canvas

- Neurons are the basic building blocks of neural networks
- Neurons multiply their input by their weights (one per input), add a bias term and then send the result to an activation function
- This is only one type of neuron. In the upcoming sessions we will discuss other types.

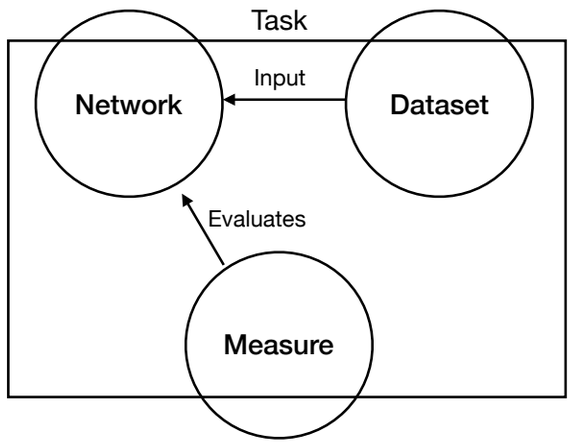
Overview

Today we will cover

- How to train a neural network.
  - Loss function.
  - Updating weights systematically.
- Solve a more complex XOR problem using Keras.
- How to evaluate the performance of a network.
  - Over- and underfitting.
  - Hyperparameter tuning.
- Practice a methodology of fitting a network to Fashion MNIST data

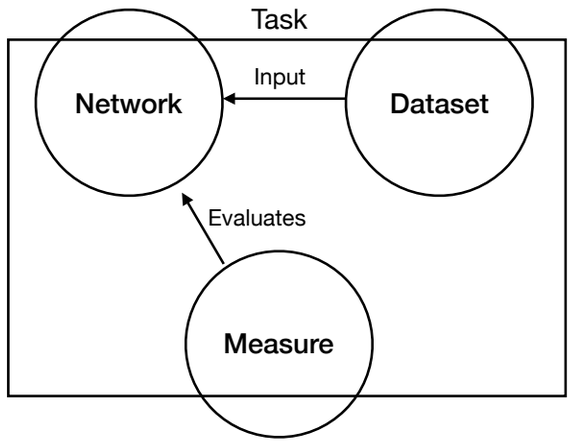
A step back

- What do we want to achieve with our neural networks?
- We want to **input data** and get out some **meaningful result.**
- In machine learning this problem is formulated so that we have a **task** which we want to perform.
  - Regression
  - Classification
  - Clustering



A step back

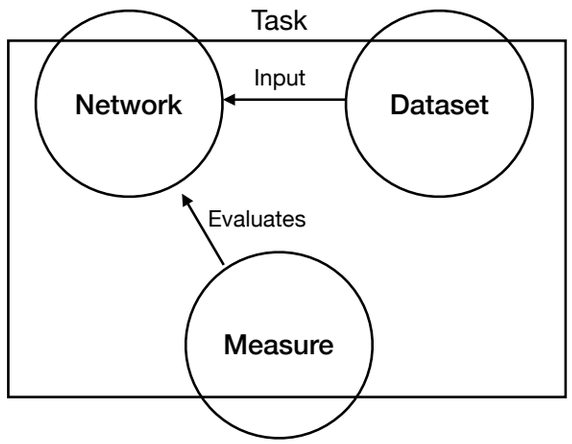
- Alongside the task we have some input, usually in the form of a **dataset**.
- When the have some **metric** / **measure** which can evaluate how well we are performing the task.
  - Accuracy
  - F1 score
  - Area-under-curve (AUC)



A step back

- Our network should then learn to **map** our **input to** some **output**.
- We thus treat the network as some function f which maps some x to $\hat{y}$ some prediction:

$$f(x) = \hat{y}$$

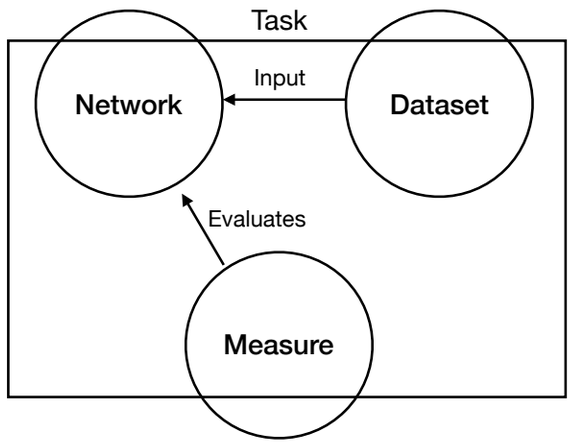


Supervised learning

- We then provide some feedback to our network using **loss**.
- The loss is a function which takes as input the output of the network $\hat{y}$, and for **supervised learning**, the correct output y

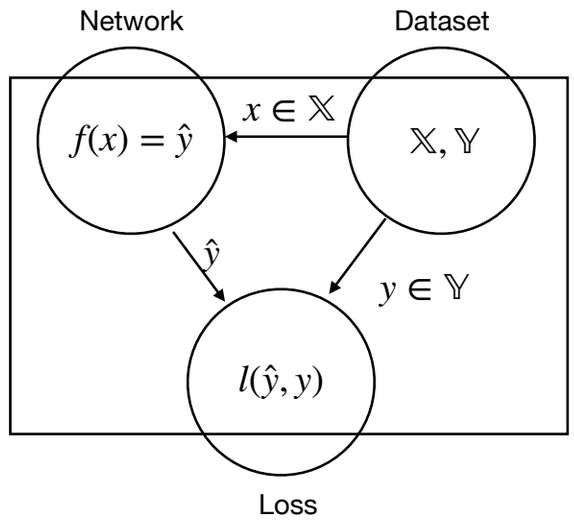
$$l(\hat{y}, y)$$

- The loss outputs a single number which tells us how well we are doing for that example.



Supervised learning

- The loss should be large when we are performing poorly and low (or 0) when we are performing well.
- The loss function is dependent on the task at hand.
- We **do not use the metric** to provide feedback to the network for technical reasons.



- Metrics are typically not differentiable, making them unsuitable for optimisation purposes
- Metrics give you some indication of model performance for your domain

Example: Linear Regression

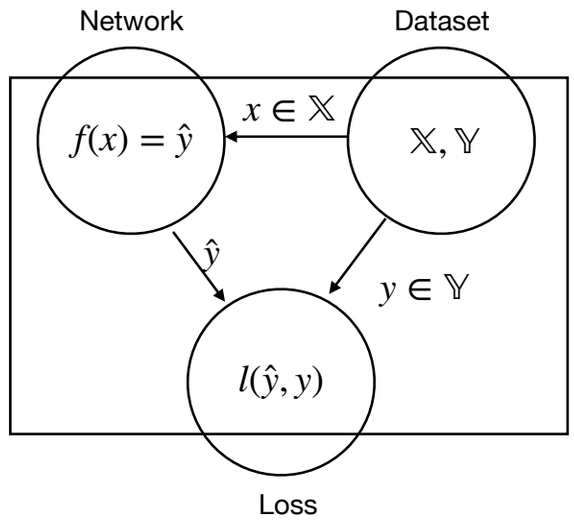
- For example, if we map our inputs directly to an output using:

$$f(x) = w^T x + b$$

- And use this loss:

$$l(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

- we get linear regression.



Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$ $l(\hat{y}, y) = |\hat{y} - y|$ loss

Total loss

Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$

Total loss

Understanding loss

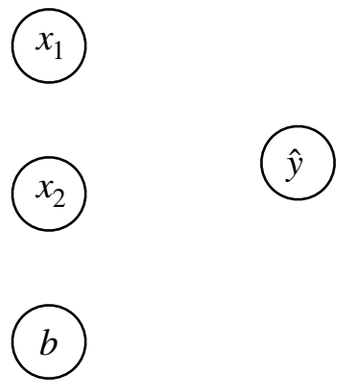
3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$



Total loss

Understanding loss

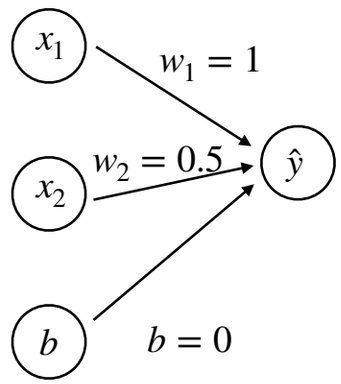
3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$



Total loss

Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

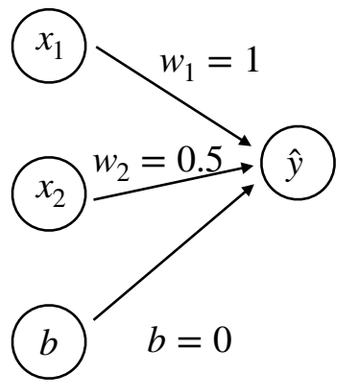
$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$f(x) = w_1 x_1 + w_2 x_2 + b$



Total loss

Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

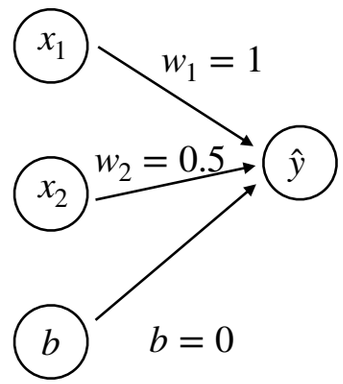
$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$f(x) = w_1 x_1 + w_2 x_2 + b$



$x^1 = (1,1) \quad f(x^1) = \hat{y}^1 = 1.5 \quad l(\hat{y}^1, y^1) = 0.5$

Total loss

Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

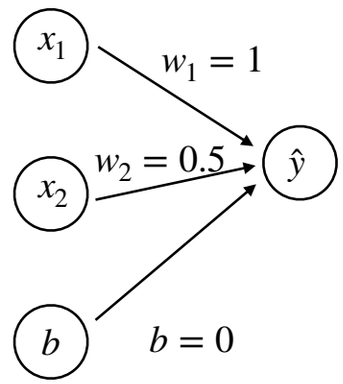
$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$f(x) = w_1 x_1 + w_2 x_2 + b$



$x^1 = (1,1) \quad f(x^1) = \hat{y}^1 = 1.5 \quad l(\hat{y}^1, y^1) = 0.5$

$x^2 = (1,0) \quad f(x^2) = \hat{y}^2 = 1 \quad l(\hat{y}^2, y^2) = 0$

Total loss

Understanding loss

3 data points $= \{(x^1, y^1), (x^2, y^2), (x^3, y^3)\}$

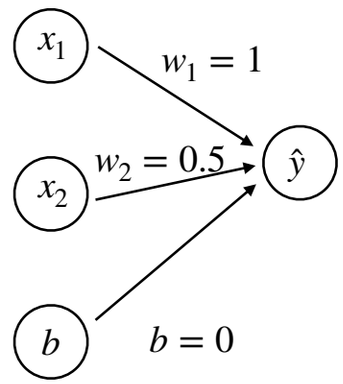
$x^1 = (1,1) \quad y^1 = 2$

$x^2 = (1,0) \quad y^2 = 1$

$x^3 = (0,0) \quad y^3 = 0$

$l(\hat{y}, y) = |\hat{y} - y|$ loss

$f(x) = w_1 x_1 + w_2 x_2 + b$



$x^1 = (1,1) \quad f(x^1) = \hat{y}^1 = 1.5 \quad l(\hat{y}^1, y^1) = 0.5$

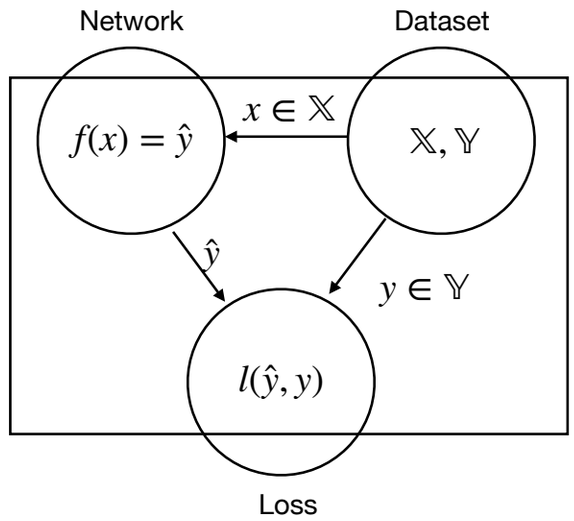
$x^2 = (1,0) \quad f(x^2) = \hat{y}^2 = 1 \quad l(\hat{y}^2, y^2) = 0$

$x^3 = (0,0) \quad f(x^3) = \hat{y}^3 = 0 \quad l(\hat{y}^3, y^3) = 0$

Total loss $\sum_{i=1}^{3} l(\hat{y}^i, y^i) = 0.5$

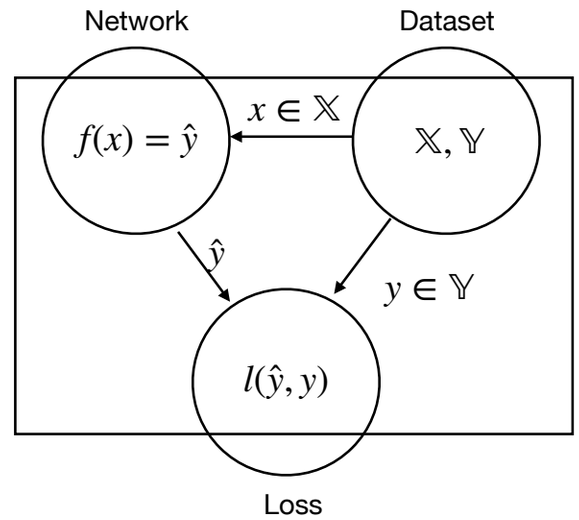
Minimising loss

- We then seek to minimise the loss over the whole dataset.
- To minimise the loss we adjust the **weights** of the network so that the loss decreases.
- We then seek to find the weights of the network which minimises the loss.



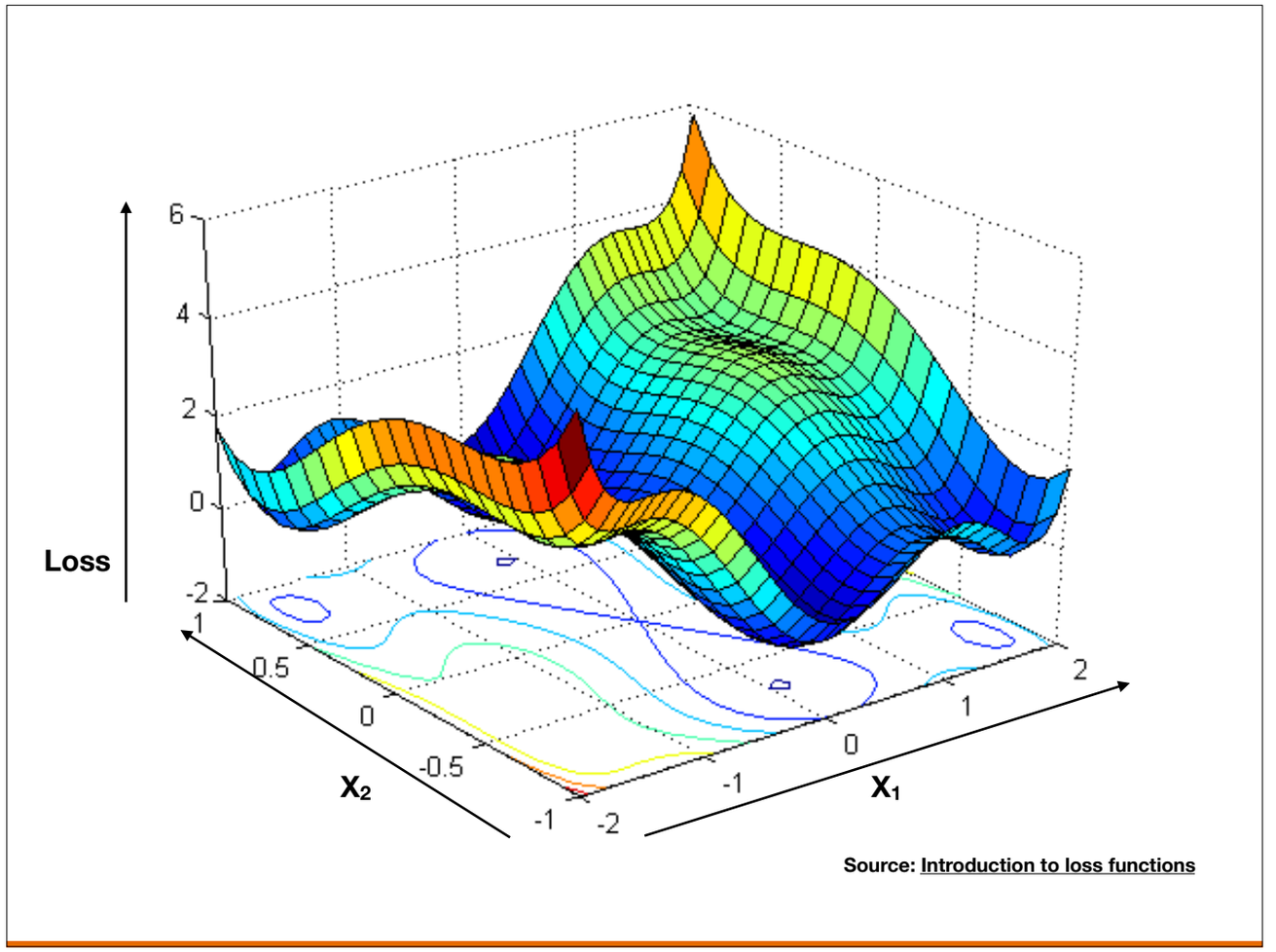
Updating weights

- How do we systematically update the weights to reduce loss?
- If our loss and network are made by using differentiable functions, we simply **differentiate the total loss w.r.t. the weights** and update the weights with that information.
- There are lots of finer details to this but the important part to know is that **each weight contributes to the total loss**.
- This means that our loss function over the whole dataset is high-dimensional function, as it is a function of every weight in the network.



Total loss $= J(\boldsymbol{w})$

All the weights in a single vector



Source: Introduction to loss functions

- This is the loss surface for two weights: x1 and x2
- We want to go down the loss surface, and need to follow the slope

Updating weights

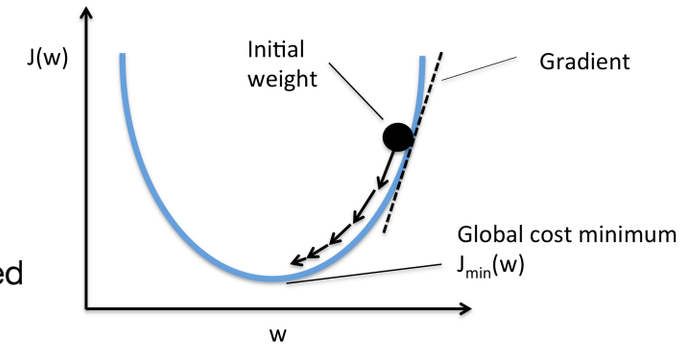
(batch) Gradient descent

- We apply this procedure a few times, in each iteration we update the weights s.t.

$$w_i := w_i - \mu \frac{\partial J(w)}{\partial w_i}$$

where μ is a value in (0;1] called the **learning rate**

- This algorithm is called **(batch) gradient descent** (GD).
- The loss (and therefore the gradient) is computed over **all elements** in the dataset.

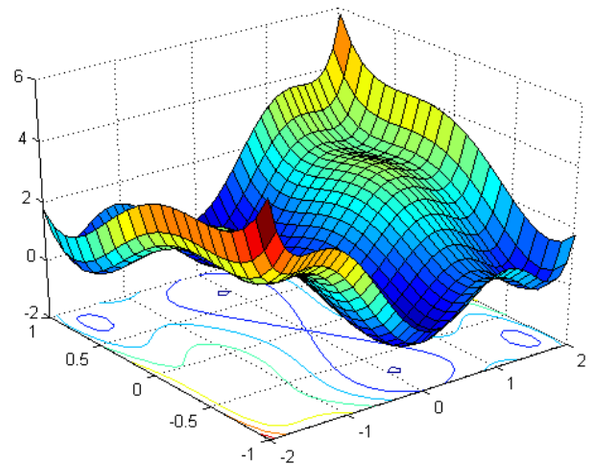


Source: Gradient optimization

Updating weights

(Batch) Gradient descent

- We are not guaranteed to find a global minimum of the loss function using GD.
- In practice, it tends to find pretty good local minima.

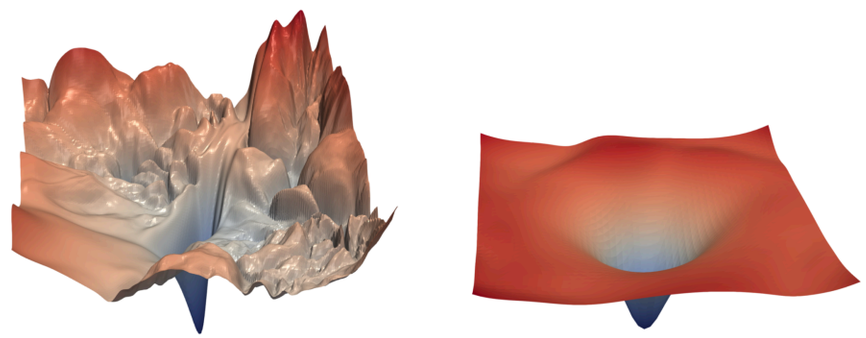


Source: Introduction to loss functions

- Global optimum is not necessarily what we want, because we are probably overfitting a lot at that point

Updating weights

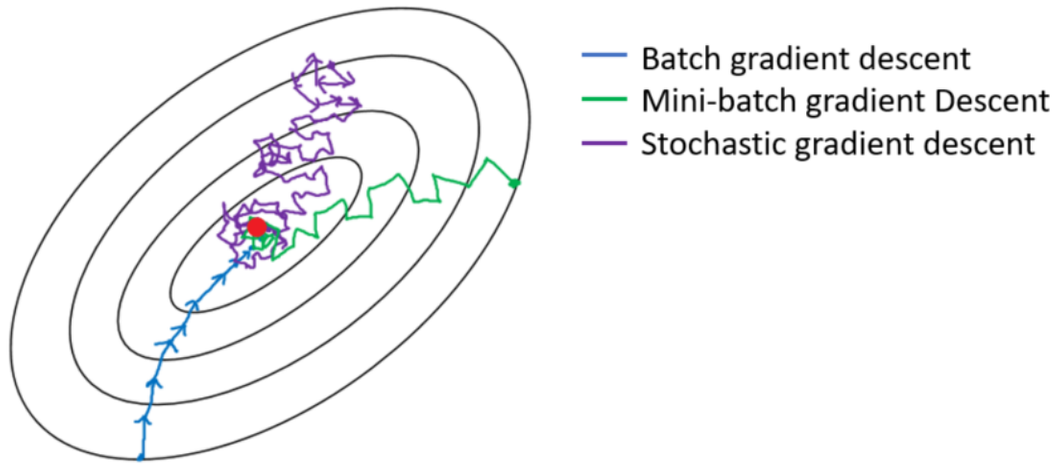
- Batch gradient descent is time-consuming
- Batch gradient descent expects convex loss surfaces
- Use **mini-batch gradient descent** instead
- Batch size = 1 is **stochastic gradient descent**



- Batch gradient descent expects relatively smooth loss surfaces like the one on the right, but more often our surface looks like the one on the left
- For reasons of performance we use mini-batches of data rather than ordinary batch gradient descent
- It will cause our total loss to not always decrease, but runs faster.
- Batch size = m (the whole dataset) is equivalent to GD.
- Batch size = 1 is called stochastic gradient descent.
- We iterate through the dataset many times, the number of times is the epoch, another hyperparameter.

Updating weights

Mini-batch gradient descent

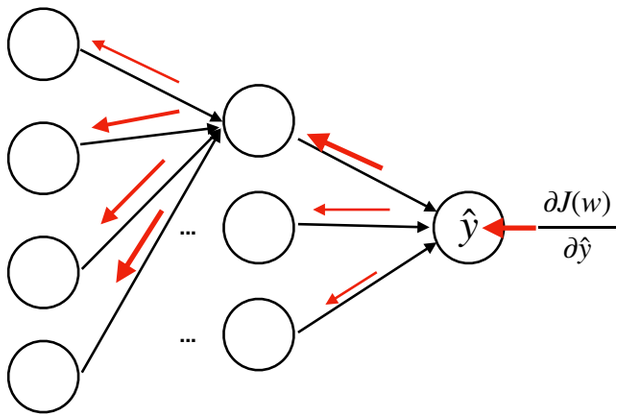


Source: Andrew Ng, deep learning course, Coursera

Applying GD in DL

Backpropagation

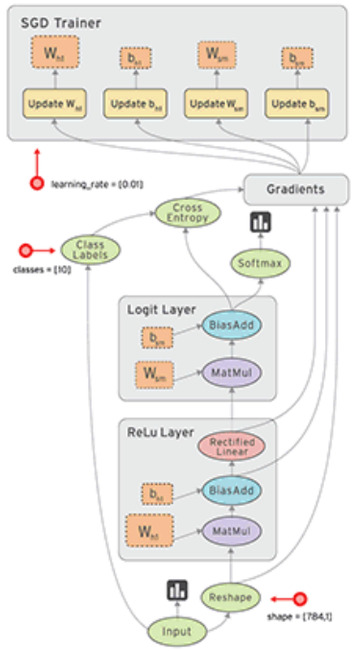
- In the context of DL we need to compute the gradient for each layer.
- We do this by applying the **chain rule** of derivatives.
- This algorithm is known as **backpropagation**.



- Backprop sends stronger "error signals" to nodes which contributed a lot to the overall error, and will update these weights more strongly than others

Keras -> TensorFlow

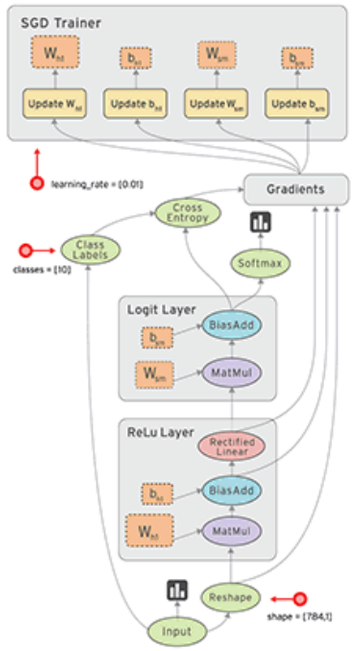
- We do not need to worry at all about updating these weights and differentiating since this is done by the framework (TensorFlow).
- It is still important to know what is happening when you need to debug your network.



Source:
https://www.tensorflow.org/guide/graphs

Keras -> TensorFlow

- We do not need to worry at all about updating these weights and differentiating since this is done by the framework (TensorFlow).
- It is still important to know what is happening when you need to debug your network.



Source:
https://www.tensorflow.org/guide/graphs

Creating neural networks

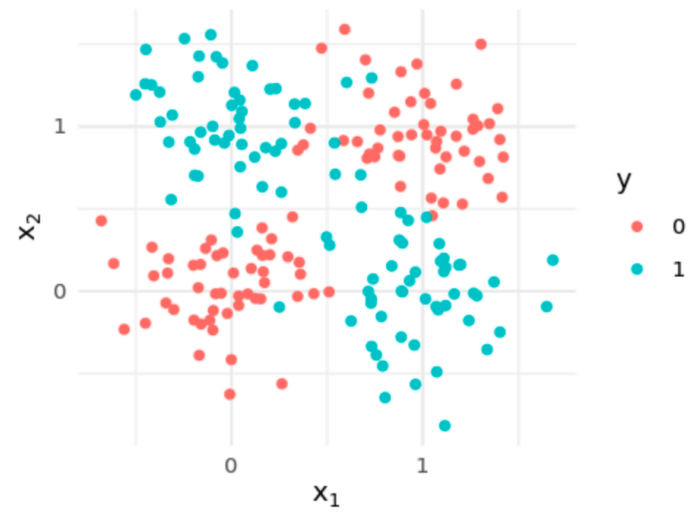
- Many frameworks exist; **TensorFlow**, **CNTK**, **Torch**, **Keras**, **Theano**, **Caffe**, ...
- We will use **Keras** (https://keras.io/), and **TensorFlow** backend.
- **Eager** and **lazy** evaluation.

TensorFlow

theano

The problem

- Adam optimiser instead of SGD



- Adam: Adaptive moment optimization
  - Instead of a single learning rate, we have a learning rate per parameter
  - We incorporate something similar to a momentum term, where past gradients have some effect on the current gradient as well. This speeds up training.
- Adam is quite robust and reliable, and usually quite fast
- We are solving a modified XOR problem with Keras

Hands-on

Go to https://dba.projects.sda.surfsara.nl/

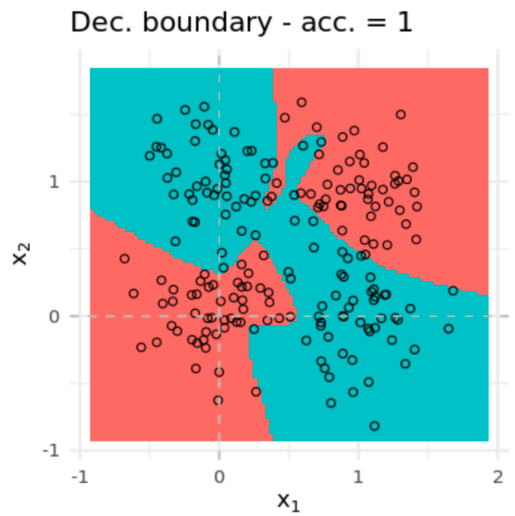
Notebook: `02a-keras-on-xor.ipynb`

Break at 11:00

Second part at 11:10

Evaluating performance

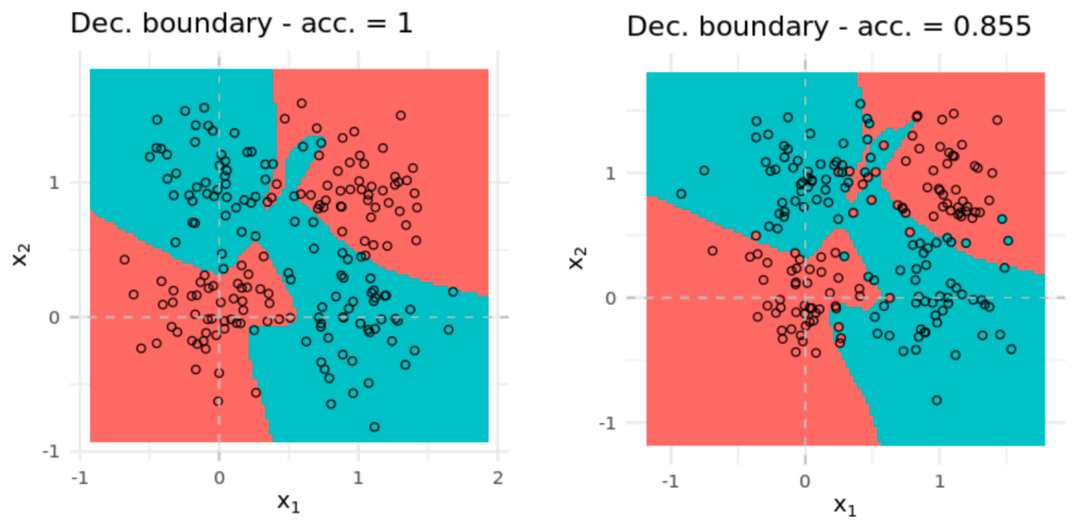
- Exercise 5's decision boundary **does not generalise**
- Use a **test set** to estimate performance on unseen examples



- In the notebook we saw that we were able to perform very well (even perfectly!) on our training dataset.
- But the problem is that you would not expect this decision boundary to generalise well to new examples.
- To test such generalisation we need to evaluate our model against a dataset which it was not trained on, the test set. In this case, we see that the decision boundary is far too specific to the original data set, and shows severely reduced performance on our test set (right).

Evaluating performance

- Exercise 5's decision boundary **does not generalise**
- Use a **test set** to estimate performance on unseen examples



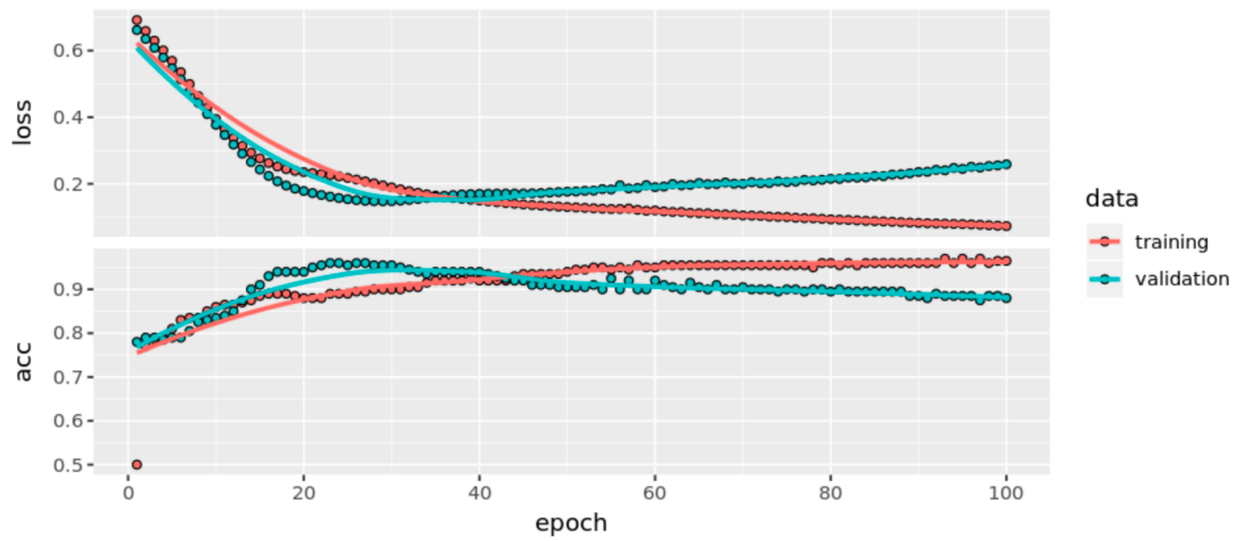
- In the notebook we saw that we were able to perform very well (even perfectly!) on our training dataset.
- But the problem is that you would not expect this decision boundary to generalise well to new examples.
- To test such generalisation we need to evaluate our model against a dataset which it was not trained on, the test set. In this case, we see that the decision boundary is far too specific to the original data set, and shows severely reduced performance on our test set (right).

Evaluating performance

- **Hyperparameters** affect performance
  - learning rate
  - batch size
  - # layers
  - # neurons
- We need to test combinations manually
- We test different hyperparameters on a **validation set**

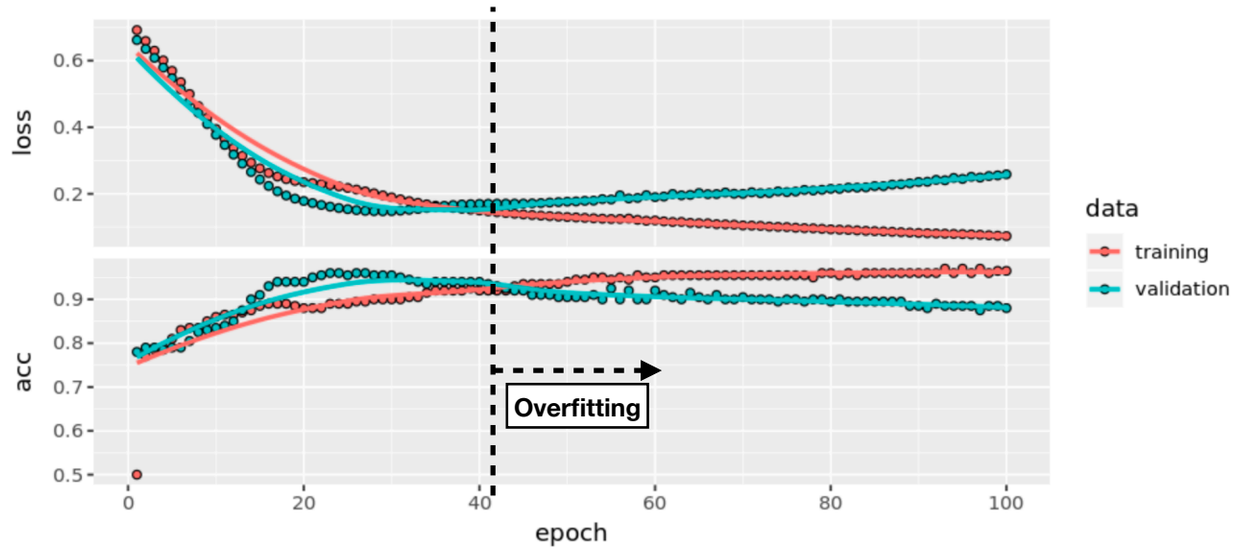
Our model contains various hyperparameters which we need to set which affect our performance.
learning rate
batch size
layers
neurons
To get the best results on the test set we might need to test many different combinations of them.
Testing all these combinations on the test set is not a great idea as we want to know how well our model actually performs on never seen before data.
To fix this we test different hyperparameters on a third dataset, a validation / development set.

Early stopping



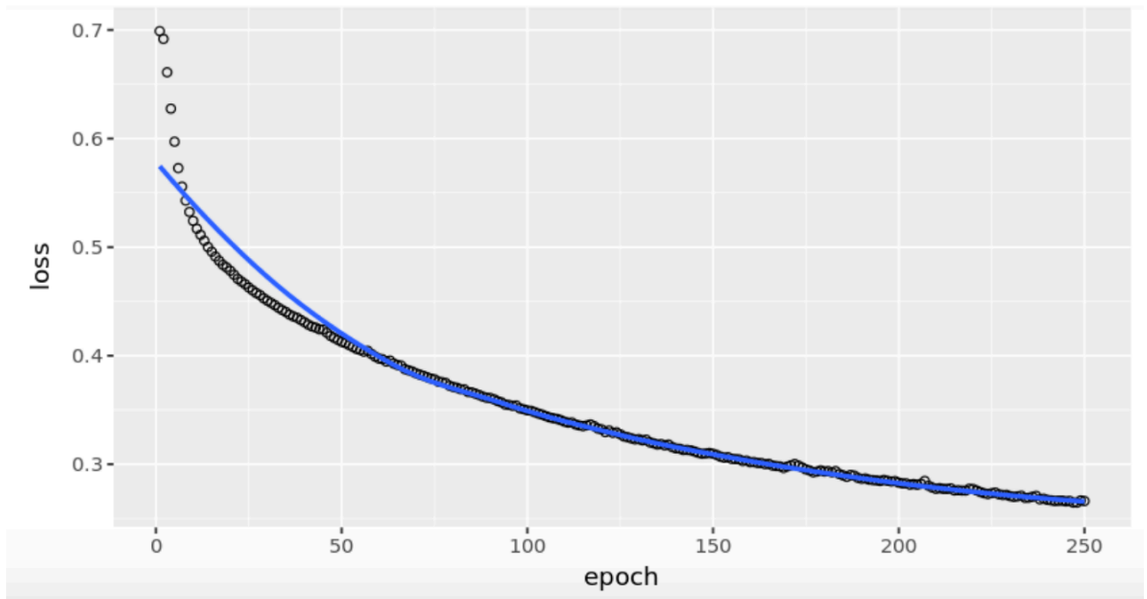
- Given sufficient capacity, the loss on the training set will keep decreasing
- The validation set loss will decrease up to a point, after which it will start increasing again. We start overfitting.
- One way to control overfitting is to do early-stopping: identify the point at which we start overfitting, and stop training there. This is what we will do in the assignment.
- Early-stopping is generally discouraged in favour of dropout and regularisation: techniques we will discuss next session.

Early stopping



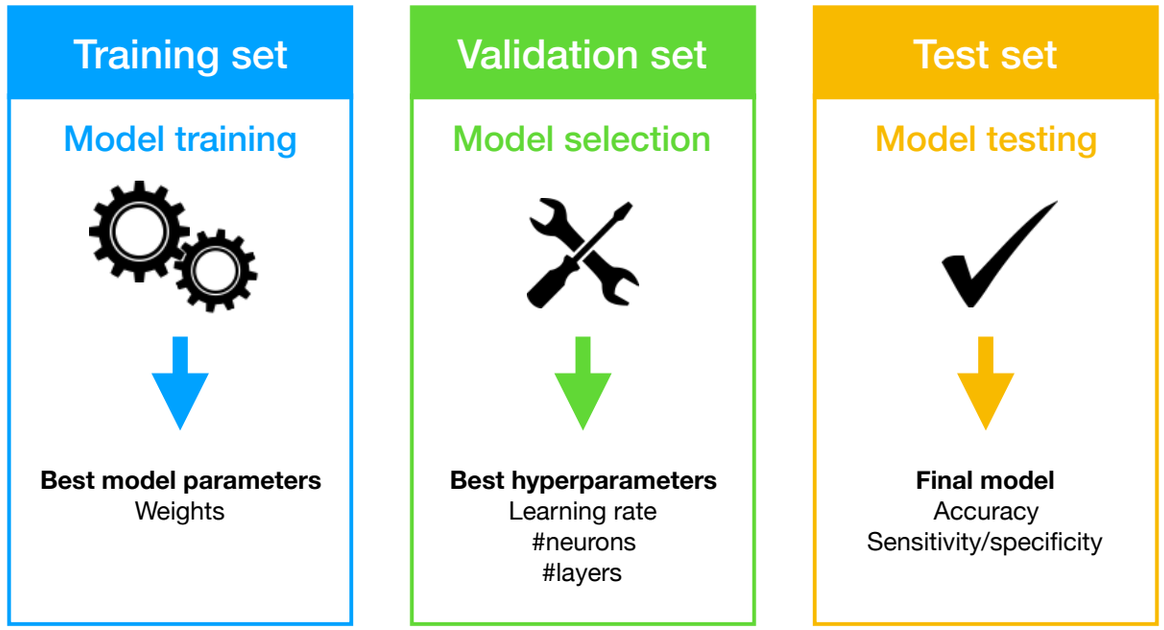
- Given sufficient capacity, the loss on the training set will keep decreasing
- The validation set loss will decrease up to a point, after which it will start increasing again. We start overfitting.
- One way to control overfitting is to do early-stopping: identify the point at which we start overfitting, and stop training there. This is what we will do in the assignment.
- Early-stopping is generally discouraged in favour of dropout and regularisation: techniques we will discuss next session.

Underfitting

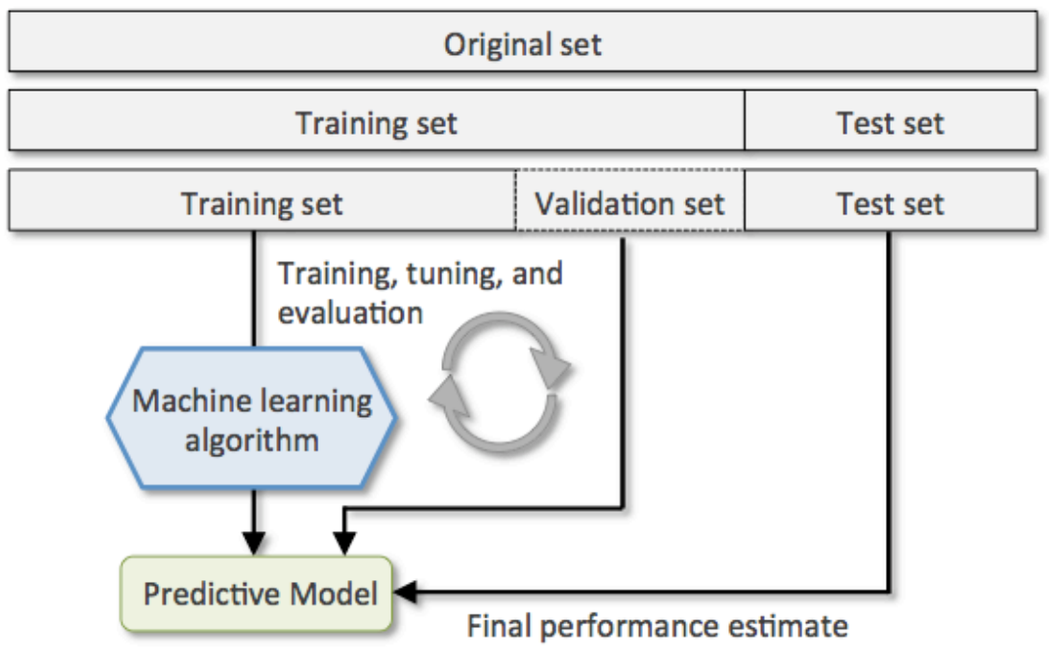


- Even after many epochs, the loss does not get close to 0. The model does not have enough capacity. Increase the number of layers and/or neurons.

Training, validation, testing



- Specific to deep learning are the tweaking of the different hyperparameters, and how overfitting/underfitting develops given model capacity and optimization

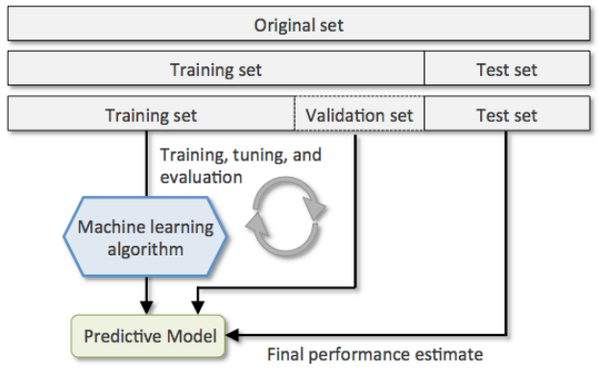


Source: Cross Validation & Ensembling

- Minimise information leakage across the different sets
- Validation set should not know anything about the training set and vice versa, same for the test set

Evaluating performance

- We generate the validation and test sets ourselves by splitting the original data set.
- With smaller datasets we might split to train/val/test as 70/10/20.
- With larger (millions of examples) it can be closer to 98/1/1.



Source: Cross Validation & Ensembling

Overfitting / underfitting

- In the previous notebook we did not expect our model to generalise well. In this case our model was **overfitting** the data.
- When we overfit, we see a **low training error** but **high validation / test error**.
- Similarly, if we see a **high training error**, we might be **underfitting** the data. We need to increase model capacity.
- We can test if we are underfitting by adding additional layers / more neurons and see if the training error goes down.

Training process

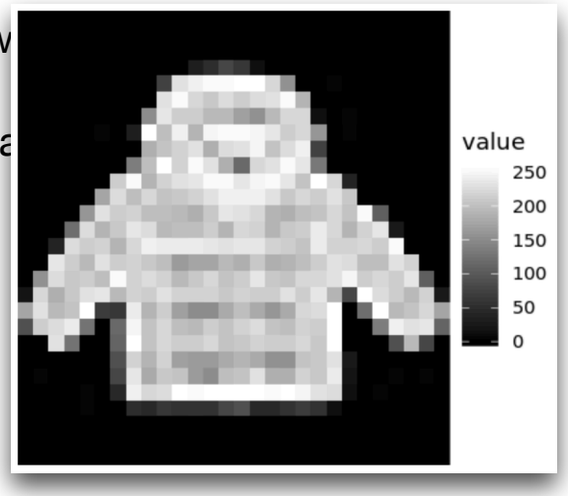
1. Load the data
2. Split the data into a training, validation and test set
3. Normalise the data
4. Build a simple, initial model
5. Improve the model such that it has sufficient capacity
6. Perform early stopping and evaluate the model on the test set

Normalisation

- Features need to have the same range
- Usually between 0 and 1 or z-scaled
- Normalise based on **training** set

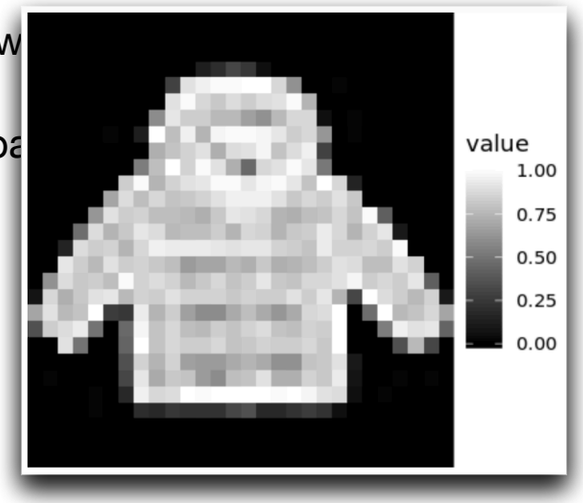
Normalisation

- Features need to have the same range
- Usually betw
- Normalise ba



Normalisation

- Features need to have the same range
- Usually betw
- Normalise ba



Normalisation

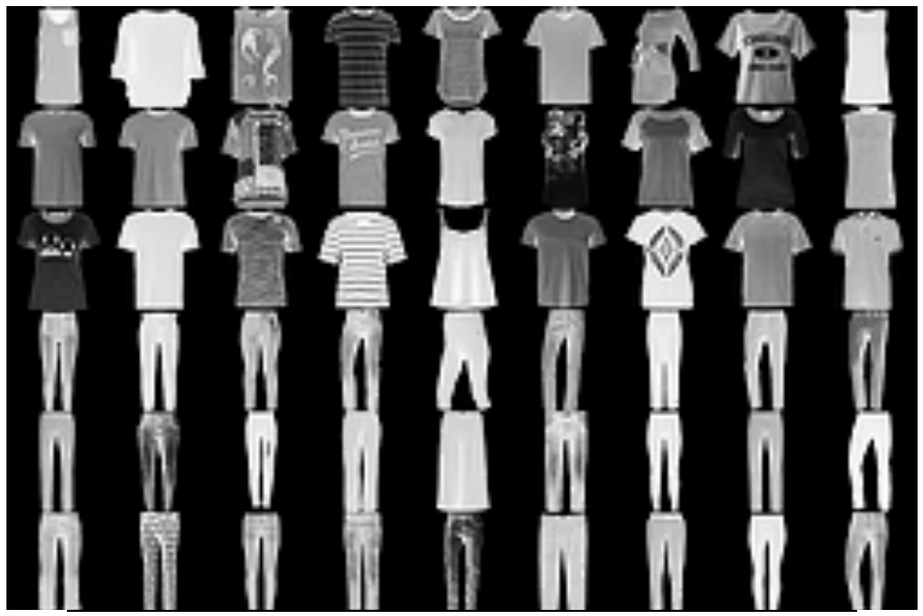
- Features need to have the same range
- Usually between 0 and 1 or z-scaled
- Normalise based on **training** set

The problem

Because MNIST is too easy

The problem

Because MNIST is too easy



Hands-on

Go to https://dba.projects.sda.surfsara.nl/

Notebook: `02b-keras-on-fashion-mnist.ipynb`

Wrap-up and assignment at 12:20

Assignment

- Credit card fraud detection
- PCA data, and charged amount
- **Classify into fraud or no fraud**

PCA1	PCA2	PCA3	PCA4	···	PCA26	PCA27	PCA28	amount
6.459225	1.186582	0.7396468	6.1330009	···	-3.7332062	0.1133679	-0.500350	2.99
5.957488	3.642683	-1.1402842	1.0664525	···	-2.7456614	-0.1427765	1.011820	1.98
-3.989565	-4.949568	-1.3111061	-1.9001151	···	2.4765595	0.6774153	-3.646334	1937.66
3.154211	4.160147	3.7704519	-1.9010526	···	3.0102923	4.4402675	2.107858	320.05
-2.092206	4.513400	-2.3695016	0.8038143	···	0.0908369	0.5200707	-3.247023	11.50
-4.831574	2.219494	1.7124901	0.9707451	···	1.6855244	-2.6501245	-3.865835	50.00

Assignment

In the environment: `assignment-1.ipynb`

Exercises:

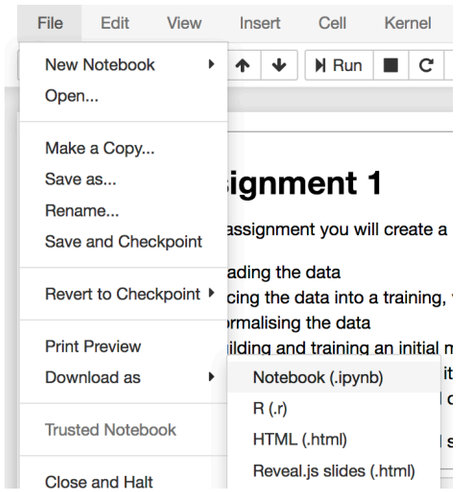
1. Load the data
2. Split the data into a training, validation and test set
3. Normalise the data by **z-scaling**
4. Build and train an initial model
5. Improve the model such that it has sufficient capacity
6. Perform early stopping, evaluate your model on the test set

- Structure is similar to last notebook
- In bold is things that are new, but will be explained in the notebook

Assignment

`assignment-1.ipynb`

- Fill out notebook as instructed
- File -> Download as -> Notebook
- Upload to Canvas assignment
- **Deadline: April 1**



- Fill out the notebook as instructed, and upload it to canvas
- Download it using File -> Download as -> Notebook (.ipynb)
- Upload it to the Canvas assignment
- **Deadline is April 1st**