UNIVERSITY OF BERGEN

INFO 284 – Machine Learning

Neural networks

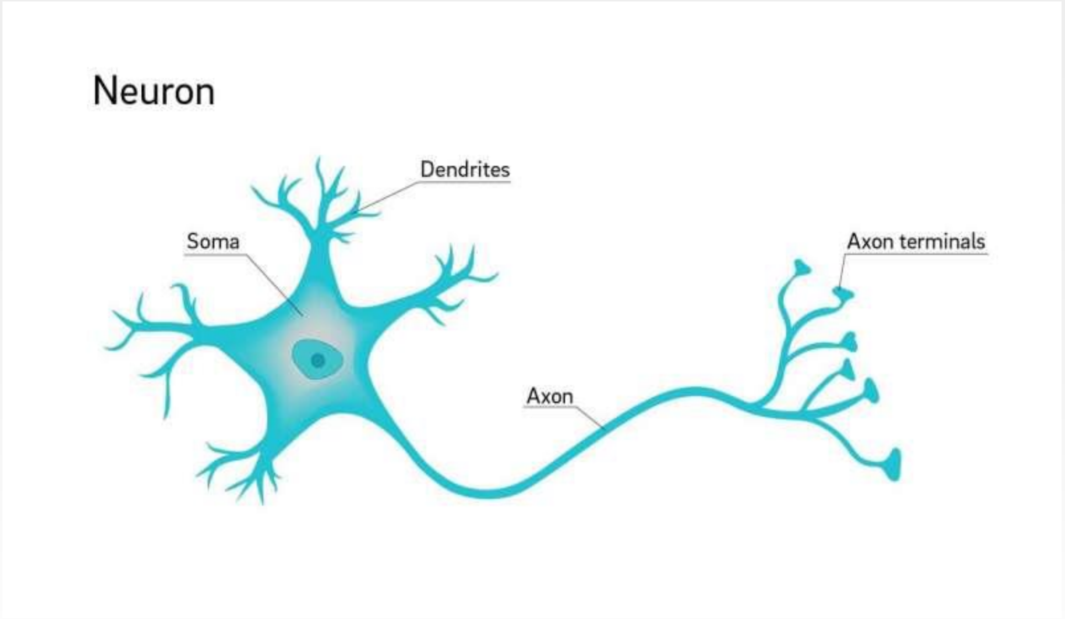
Bjørnar Tessem

UNIVERSITY OF BERGEN

Neural networks

- Class of machine learning algorithms
- Recent popularity
  - Faster computers
  - Huge datasets
  - New approaches
- Based on analogy to neurons
- Deep learning
  - computations with tensors, matrices, and vectors

The Neuron

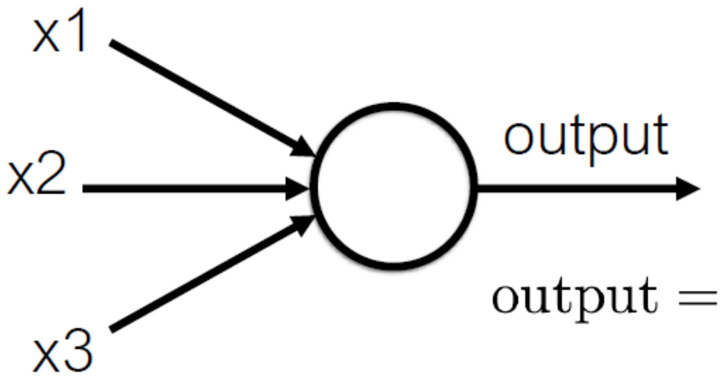


Neuron receive input at dendrites.

Process inputs in soma

Send output via axon through synapses

The Artificial Neuron



$$\text{output} = \begin{cases} 0 \text{ if } \sum_{i=1} x_i w_i \le \text{threshold} \\ 1 \text{ if } \sum_{i=1} x_i w_i > \text{threshold} \end{cases}$$

Perceptron

Workings of the perceptron

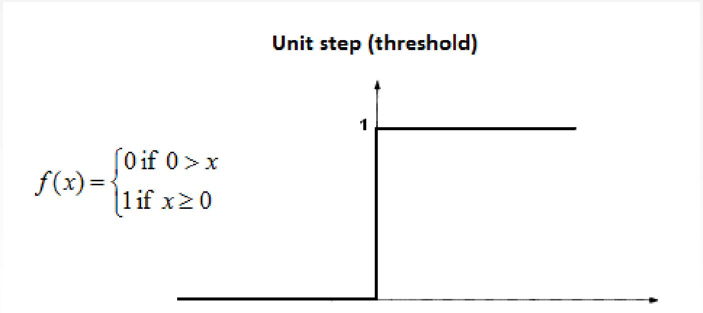
$$\text{output} = \begin{cases} 0 \text{ if } \sum_{i=1} x_i w_i \leq \text{threshold} \\ 1 \text{ if } \sum_{i=1} x_i w_i > \text{threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 \text{ if } \vec{x} \cdot \vec{w} + b \leq 0 \\ 1 \text{ if } \vec{x} \cdot \vec{w} + b > 0 \end{cases}$$

The dot product of two vectors

The threshold function

- Nice for passing information
- Problematic for learning

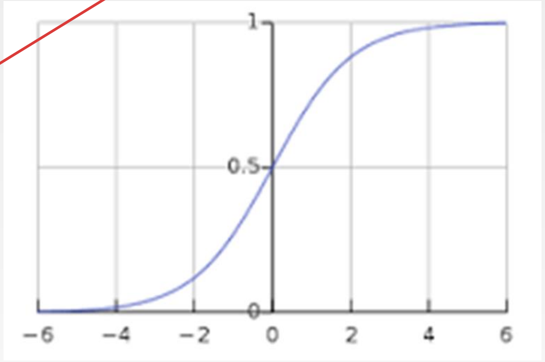


$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

Learning a Perceptron

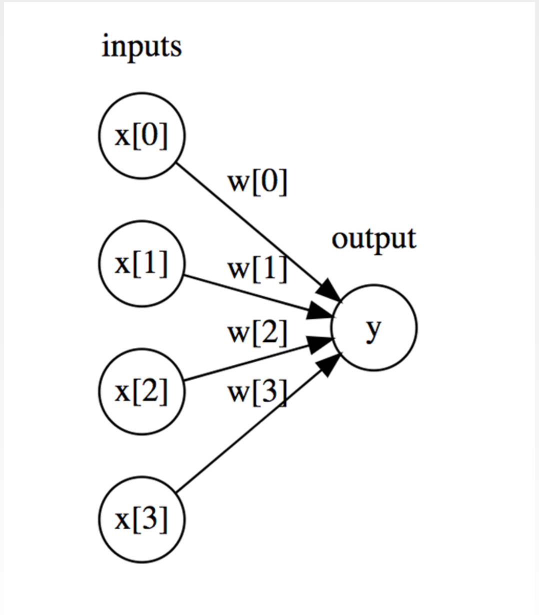
- Optimisation process
  - Adjust weights and b in small steps
  - Reduce error in output
  - Continuous/smooth functions work best
- Solution – use logistic function for output

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



$$z = \sum_{i}^{m} w_i x_i + b$$

Remember logistic regression?



$$p(\hat{y} = 1) = \frac{1}{1 + e^{w[0]*x[0]+\cdots+w[n]*x[n]+b}}$$

Sigmoid neurons behave like a logistic regression program

Alternative Activation Functions

Rectified Linear Unit:

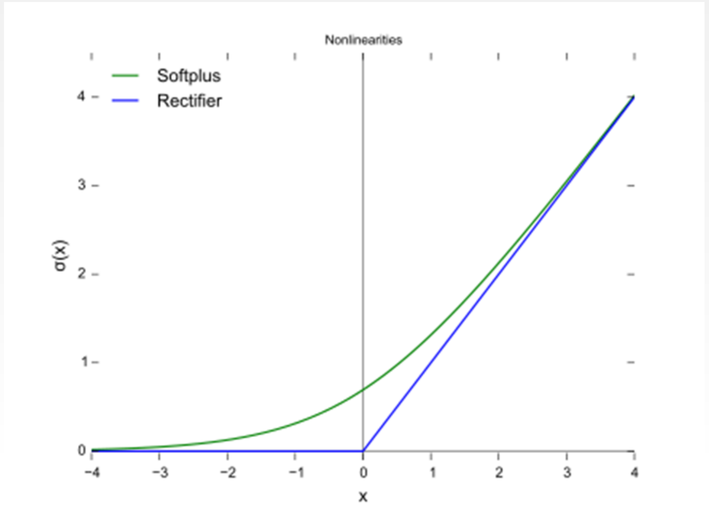
ReLU(x) = max(0,x)

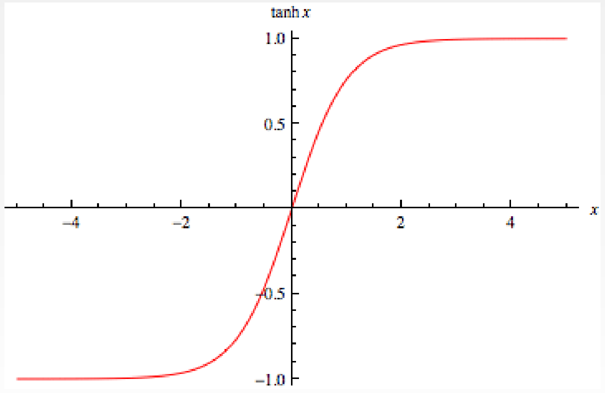
Softplus:

softplus(x) = ln (1 + e^x)

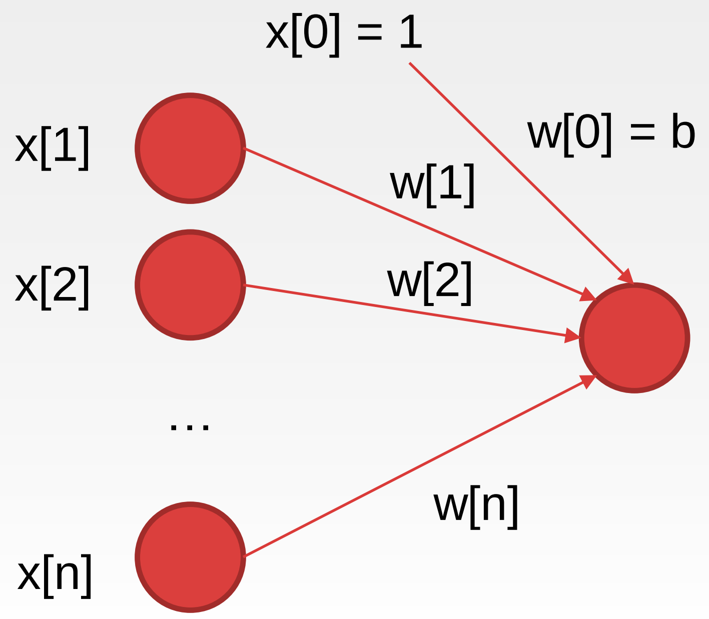
Tangens hyperbolicus

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$





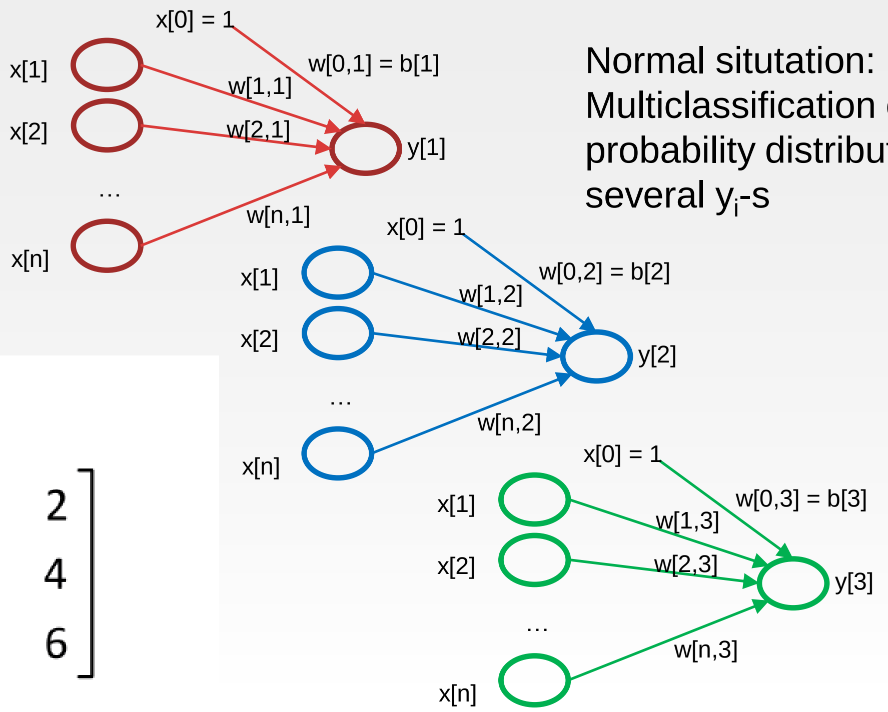
The bias input



b is also considered to be a weight

The input for b is always 1

Matrices and Perceptrons



Normal situtation:
Multiclassification or
probability distribution →
several y_i-s

$$y_i = \mathbf{w_i} \cdot \mathbf{x}$$

$$[y_1, y_2, y_3\ ] = [\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}] \cdot \mathbf{x}$$

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$$

$$\mathbf{y} = \mathbf{W^T x}$$

Matrix multiplication:
New entries are the dot products of every row in first matrix with every column in second matrix

A

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Output with logistic activation function:
$\mathbf{y} = S(\mathbf{W^T x})$

Feed forward networks

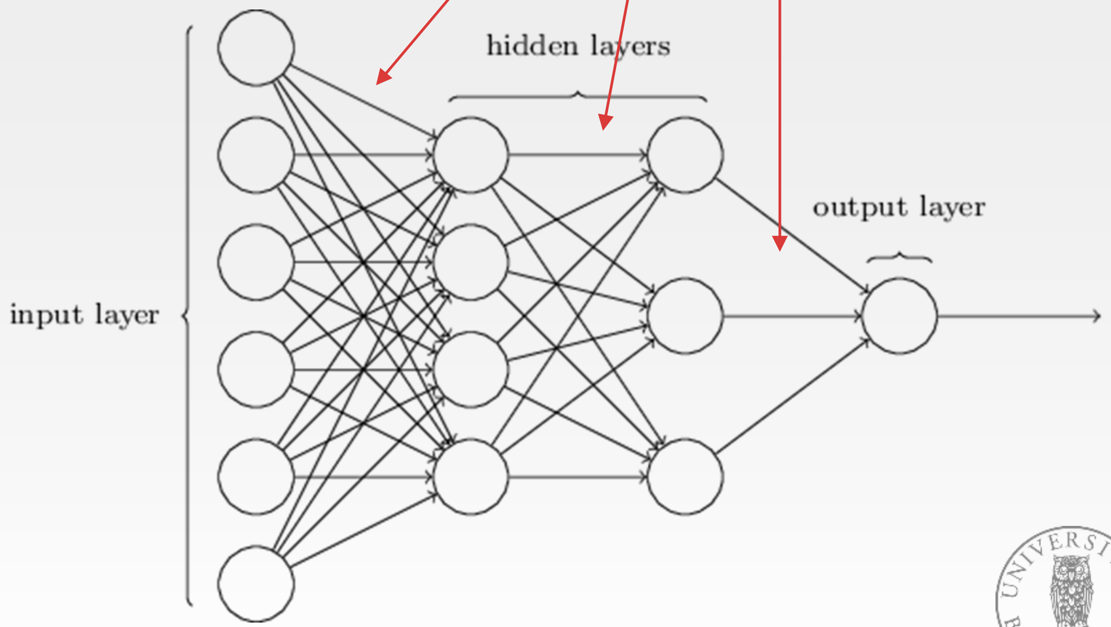
A network of neurons

OR

A sequence of linear transformations of input combined with activation functions:

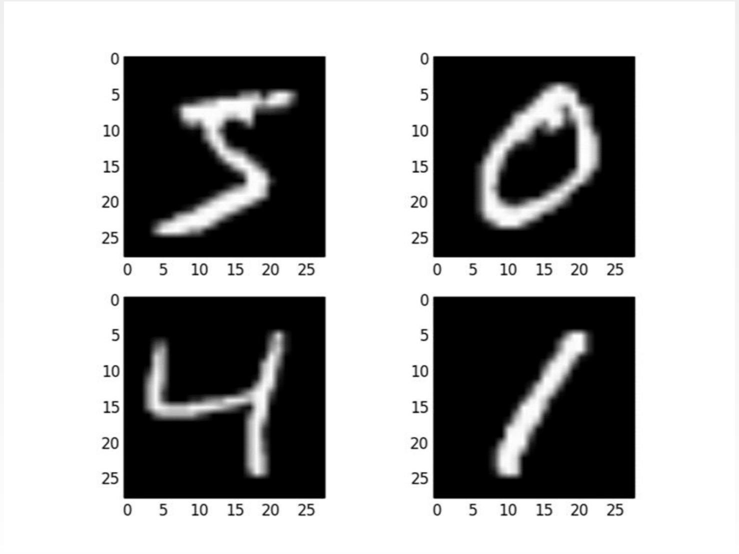
$\mathbf{y} = S(\mathbf{W}^T\mathbf{x})$

Matrices with weights for all neurons (units)



A neural network example: MNIST

http://neuralnetworksanddeeplearning.com/chap1.html

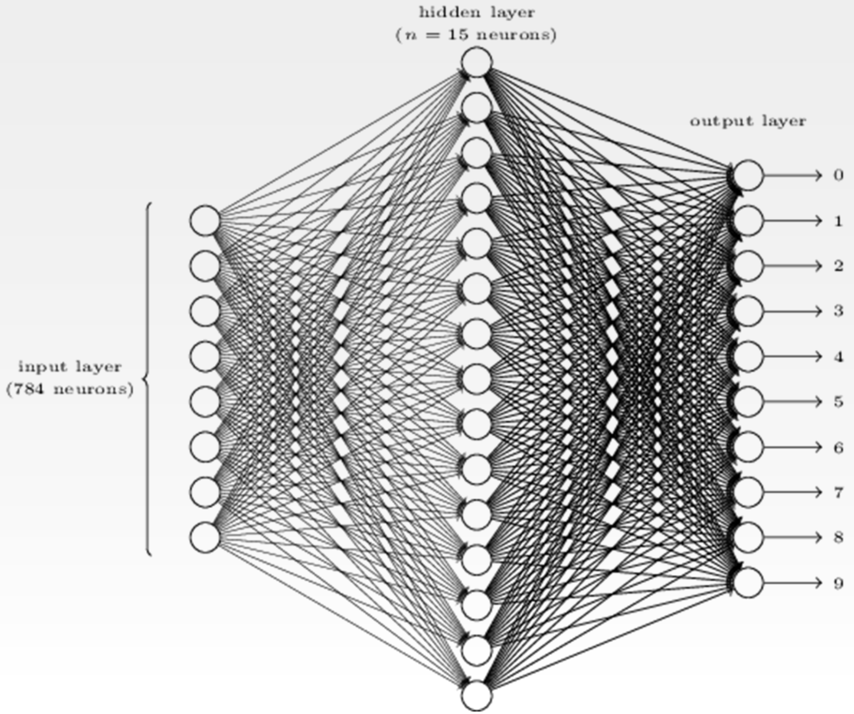


Handwritten digits: 28x28 pixels

Encode intensity of each pixel as input

Output: 10 neurones, one for each digit

A complete network



y(x)=(0,0,0,0,0,0,1,0,0,0)

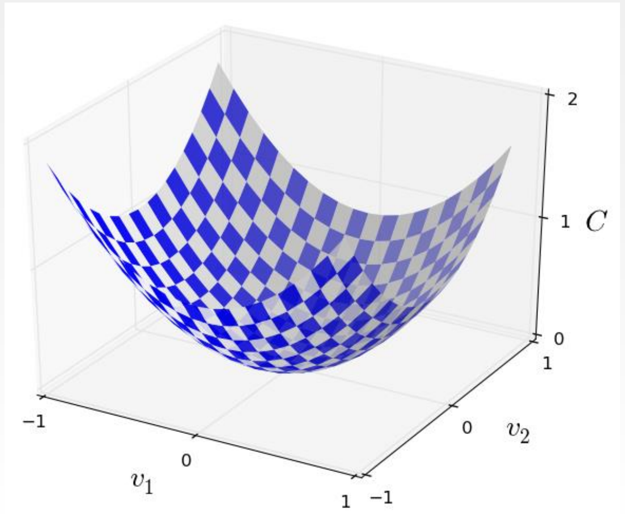
Minimize cost function:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Gradient descent

- You want to minimize some function C(v)
- Start on a random place (random values for v)
- Compute the gradient of C
  - A vector of partial derivatives
- Gradient informs us about where to change v so that C decreases
- Move towards «most steep downhill»

$$\nabla C = \left(\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2} \right)^T$$



Gradient descent

- Stepwise change of v
  - Calculate gradient
  - Change v
  - Repeat until minimum reached
- Learning rate
  - Step size
- Small steps
  - avoid going too far
  - reduce C as much as possible

$$\Delta v \approx -\eta \nabla C$$

$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\Delta C \approx -\eta ||\nabla C||^2$$

Always negative

Gradient descent in NN

- Use gradient descent to find the weights and biases that minimize the cost function
- Start from random values for weights and biases, and then change them using gradients

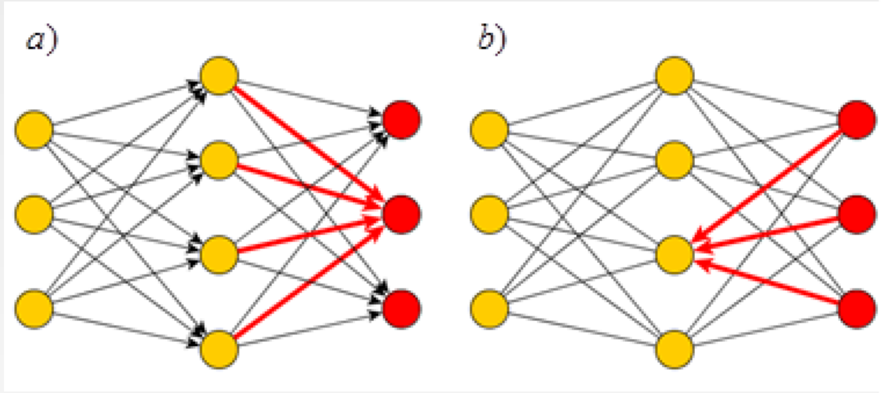
$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

- Batch, mini-batch
  - small sample of data points
  - sum up loss for gradient estimate
- Epoch
  - a single iteration on all training data

Backpropagation

- Feed forward for each data point
- From error in final output (loss) compute how much each weight in neural network influence error
  - Propagate error backwards
- Use these errors as gradient
  - Assumes continuous activation functions at each neuron



$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Some theoretical results

- A feed-forward network with a single hidden layer containing a finite number of neurons, can approximate continuous functions on compact subsets of R^N
- Simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters; however, it does not touch upon the algorithmic learnability of those parameters
- It is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators

Neural network properties

- Like all statistical models, neural networks are subject to overfitting when there are too many parameters in the model
- Complexity can be controlled by limiting the number of hidden layers, the number of neutrons in the layers, and also regularisation.
  - Regularisation – forcing random weights to become 0.0
- The weights are initially set randomly, which effects the model
- Neural networks are sensitive to great variations in input (like SVM)
- Analysing what a neural network has learned is far less transparent than decision trees or linear models
- We can visualise the outputs at different layers of the network to analyse how the weights impact the output. Stronger weight can be interpreted as higher relevance of the feature to the output

Neural network practice

- Why now?
  - Availability of annotated data
  - Increased processing power
  - Improved processing techniques
- Advantage: they can capture information contained in large amounts of data to build very complex models
- Disadvantage: takes long to train, difficult to design the right network
- Common approach: first create a network that is large enough to overfit, then shrink the network
- Divide data set in three parts, training set, validation set, test set

Optimizers

- Stochastic Gradient Descent
  - Each weight uptdate is based on a (small) random subset of the data points
- Adam – special case of Stochastic Gradient Descent
  - Learning rate is adapted for each weight
  - Remembers last learning rate for weight, and combines with error and activation values to get new learning rate
  - The default, works well with most data
  - Efficient for large data
- LBFGS – quasi-Newton method
  - Gives better results with small datasets
  - Slow on large datasets

UNIVERSITAS BERGENSIS