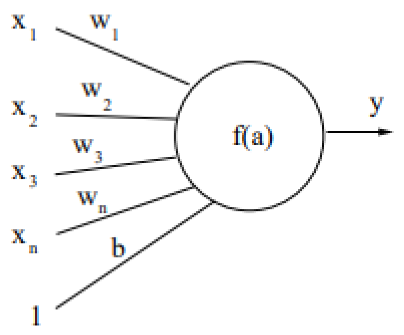
Artificial neural nets and deep learning

Chapter 1 Multilayer feedforward networks and backpropagation

1.1 Multilayer perceptrons

1.1.1 Biological neurons and McCulloch-Pitts model

A simple and popular model for biological neurons is the McCulloch-Pitts model. Be aware that this is a strong mathematical abstraction of reality.



$$a = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

$$y = f(a)$$

Figure 1.2: *McCulloch-Pitts model of neuron.*

The neuron is modelled as a simple nonlinear element which takes a weighted sum of incoming signals. Biologically this corresponds to the firing of a neuron depending on gathered information of incoming signals that exceeds a certain threshhold value.

1.1.2 Multilayer perceptrons

A single neuron is not very powerful. However, if organized into a layered network, you get a model that is able to approximate general continuous nonlinear functions. Such a network has one or more hidden layers and an output layer. A **multilayer perceptron (MLP)** with one hidden layer can be seen below.

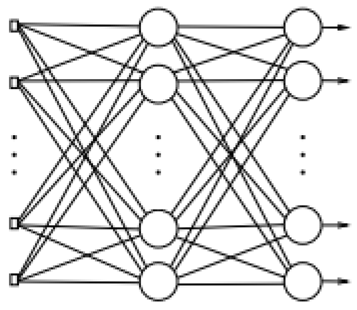


Figure 1.3: *Multilayer perceptron with one hidden layer.*

Depending on the application we can choose the linear activation function. A linear activation function is a function that calculates the weighted sum of the input values and returns the same sum as output. Below you can see some non-linear activation functions.

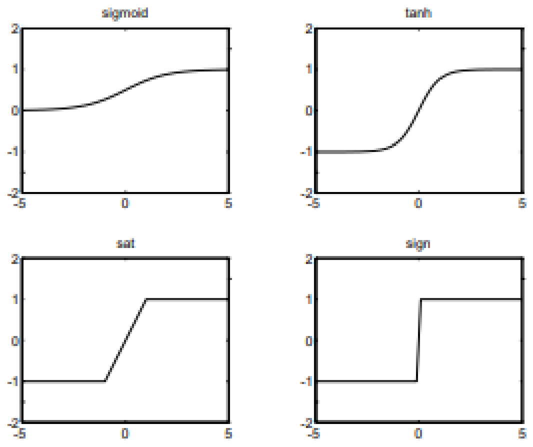


Figure 1.4: *Some typical activation functions.*

1.1.3 Radial basis function networks (RBF networks)

Radial basis function networks are a type of artificial neural network that use radial basis functions as **activation functions**. The radial basis functions used in RBF networks are typically Gaussian functions or other functions that have a peak at a particular points in space and decrease with distance from that point. RBF networks are particularly well-suited for problems where the data has a high degree of non-linearity, such as function approximation and pattern recognition.

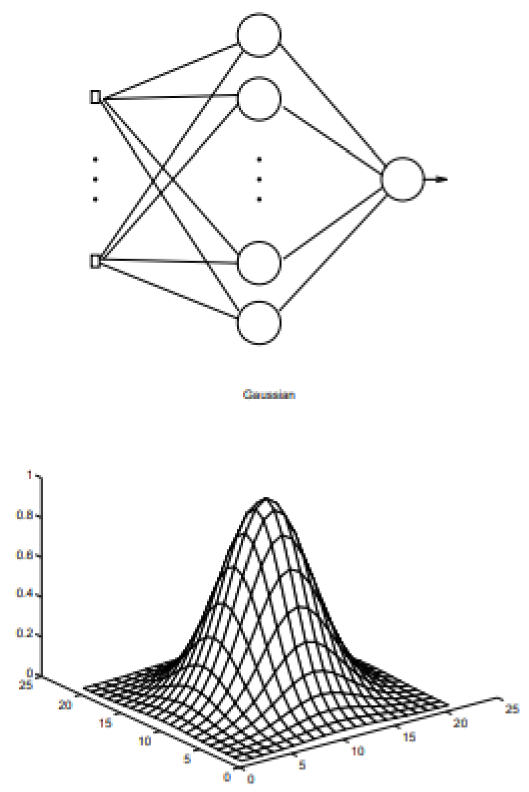


Figure 1.6: *Radial basis function network with Gaussian activation function.*

1.2 Universal approximation theorems

1.2.1 Neural nets are universal approximators

Don't know if the next part is important but I'm gonna try to explain it anyway.

Kolmogorov's theorem, also known as the universal approximation theorem. It states that a feedforward neural network with a single hidden layer and a finite number of neurons can approximate any continuous function.

A **feedforward neural network** is a type of artificial neural network in which the information only flows in one direction, from the input layer to the output layer without any loops or feedback connections. Each neuron in a feedforward neural network is connected to all neurons in the previous layer, and each connection is associated with a weight that determines the strength of the connection.

Sprecher Theorem: 🙃

Leshno Theorem: A standard multilayer feedforward NN with logically bounded piece

1.2.2 The curse of dimensionality

Neural networks avoid the curse of dimensionality in the sense that the approximation error becomes independent from the dimension of the input space, which is not the case for polynomial expansions.

1.3 Backpropagation training

1.3.1 Generalized delta rule

1.4 Single neuron case: perceptron algorithm

belangrijk?

1.5 Linear versus non-linear separability

The perceptron has serious limitations. A simple example is the XOR gate, the perceptron is unable to find a decision line which correctly separates the two classes for this problem. By adding a hidden layer the XOR problem can be solved because nonlinear decision boundaries can be realized.

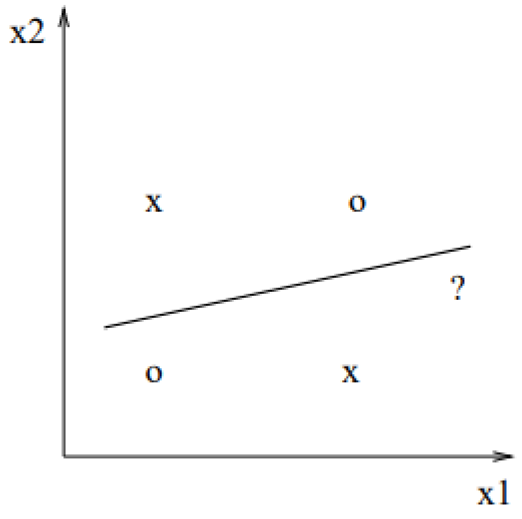
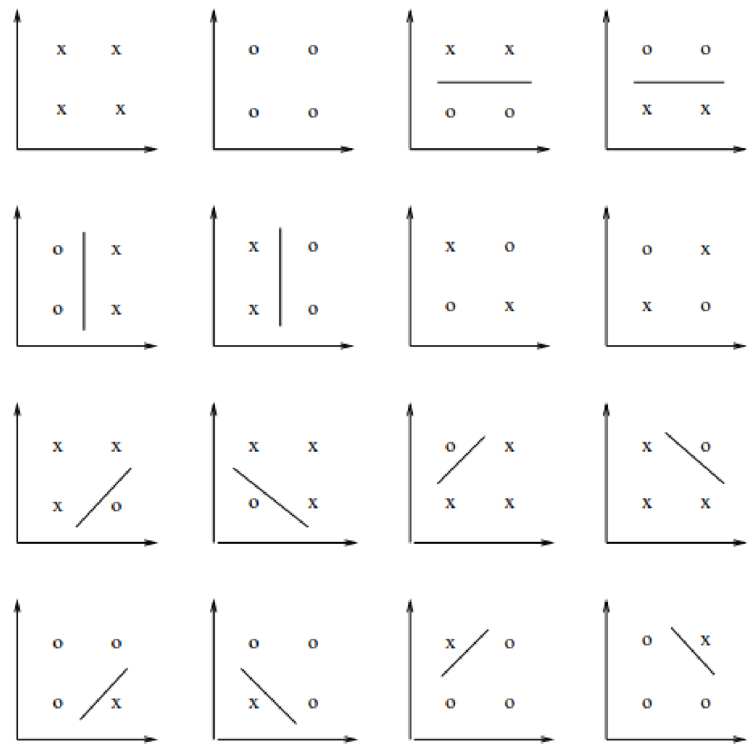


Figure 1.10: *Exclusive-OR (XOR) problem.*

By means of a hidden layer one can realize convex regions, and furthermore by means of two hidden layers non-connected and non-convex regions can be realized. The universal approximation ability of neural networks makes it also a powerful tool in order to solve classification problems.

For ***N*** points there are 2^N possible dichotomies.

Example:
$N = 4, d = 2$: $2^4 = 16$ dichotomies
14 dichotomies are linearly separable (everything but XOR)



From perceptron to multilayer perceptron (MLP) classifier:

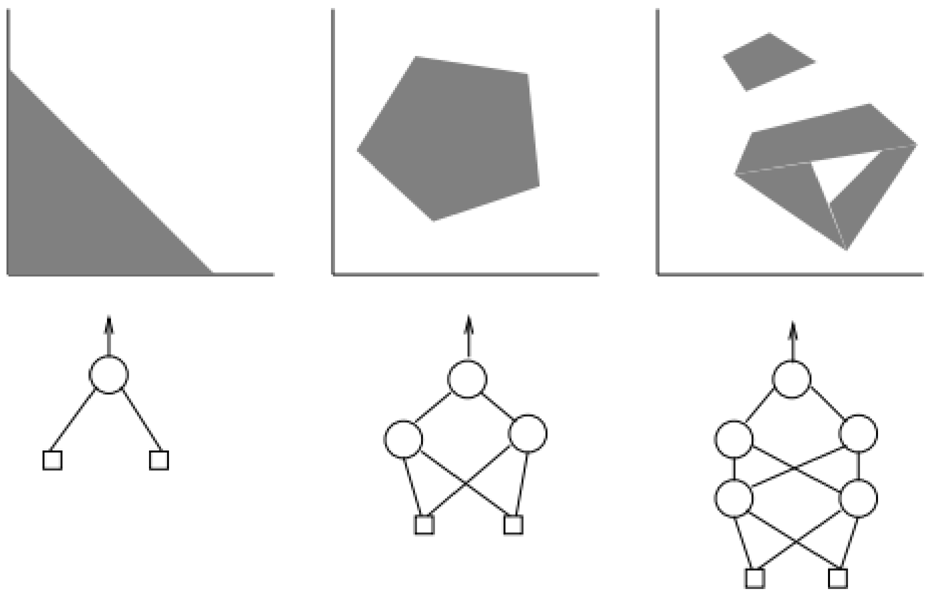


Figure 1.11: *From perceptron to multilayer perceptron (MLP) classifiers.*

One neuron (perceptron): Linear **separation**

One hidden layer: Realization of **convex regions**

Two hidden layers: Realization of **non-convex regions**

A convex region is region where any two points within the region can be connected by a line segment that lies entirely within the region.

1.6 Multilayer perceptron classifiers

Classification problems

Popular approach: Solve classification problem as a regression problem with class labels as targets.

Chapter 2 Training of feedforward neural networks

2.1 Learning and optimization

2.1.1 From steepest descent to Newton method

In neural network training, the goal is to minimize a cost function that measures the difference between the network's output and the actual outputs. This is done using an optimization algorithm, which adjusts the network's weight iteratively to reduce the cost function. A cost function is a function that tells us how far off our predictions are from the actual values.

The **steepest descent algorithm (gradient descent)** is a simple and widely used optimization method. It updates the weights in the direction of the negative gradient of the cost function. Unfortunately, it can be slow and inefficient for high-dimensional problems as it may require many iterations to converge to a minimum. To fix this we use more advanced methods such as **conjugate gradient** and **quasi-Newton**.

Newton's method is a powerful optimization method that uses second-order information about the cost function to find the minimum more quickly. It uses the cost function locally as a quadratic function and uses this to compute the direction and step-size for the weights updates. However this can be computationally expensive as this requires the inversion of the Hessian matrix, which can be large and dense in high-dimensional problems.

A **Hessian matrix** is a tool used in optimization to help understand how a function changes in different directions. It's like a table of numbers that tells you how quickly the function is changing. By analyzing the properties of the matrix you can determine whether a function has a maximum or minimum value.

2.1.2 Levenberg-Marquardt method

The method is an again an optimization algorithm commonly used for nonlinear least square problems, including neural network training. It combines the advantages of both **steepest descent (gradient descent)** and the **Gauss-Newton** methods, making it more efficient that either method alone. The method is widely used in neural network training because it is efficient, converges quickly and is less sensitive to initialization conditions and local minima.

2.1.3 Quasi-Newton methods

Quasi-Newton methods are a class of optimization algorithms that approximate the Hessian matrix without computing its exact values. An example of such a method is the **Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm**.

The **BFGS** algorithm maintains an approximation of the inverse Hessian matrix. Compared to the steepest descent and newton methods, the algorithm is more efficient and can converge faster. It has the advantage of avoiding the need to compute the exact Hessian matrix which is computationally expensive and difficult to compute.

We can improve on the **BFGS** by avoiding the direct inversion f the Hessian matrix by using the **Davidon-Fletcher-Powell formula**. This will result into a superlinear speed of convergence.

2.2 Methods for large scale problems

The previous are all good and dandy for small optimization problems they quickly become infeasible on large scale problems. That is why they developed methods more suitable for solving large scale optimization problems.

One such method is the **conjugate gradient (CG)** method. This only required the first-order derivatives and has low memory requirements. However it may converge slowly or even fail to converge. The advantages of conjugate gradient methods are that they are faster than backpropagation and that no storage of matrices is required.

Another method is the **limited memory BFGS method**. which is a method we previously mentioned but uses a limited amount of memory to approximate the Hessian matrix.

2.3 Overfitting problem

The overfitting problem we all know is especially severe in neural networks because they have many degrees of freedom, which makes it easy to memorize the training data instead of learning the underlying patterns.

We can use some techniques to prevent overfitting such as **regularization** or **ridge regression, cross validation and early stopping.**

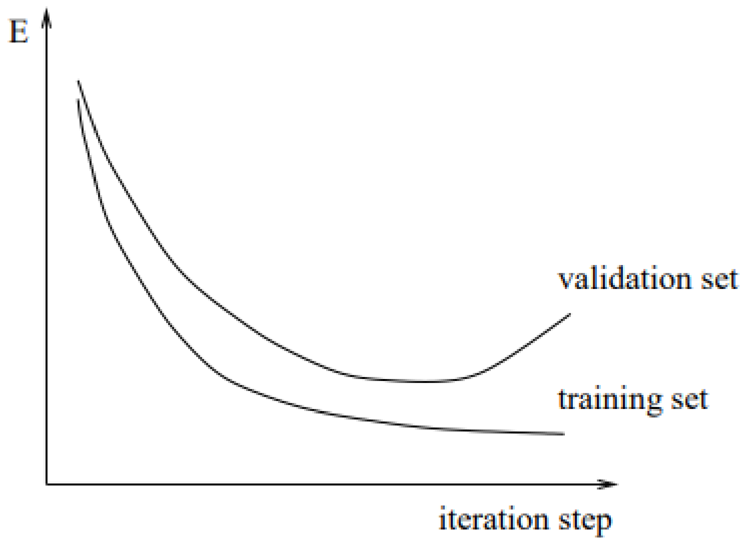
2.4 Regularization and early stopping

2.4.1 Regularization

Regularization is a technique used to prevent overfitting by adding a penalty term to the cost function. This penalty term restricts the weight values, making them smaller and closer to zero. This results in a simpler model that is less likely to overfit the training data.

2.4.2 Early stopping and validation set

Early stopping is another technique used to prevent overfitting in which the training of the neural network is stopped before it reach the global minimum of the cost function. The idea is to monitor the performance of the network on a validation set during training and to stop the training process when the validation error starts to increase.


E
validation set
training set
iteration step