Unit 16

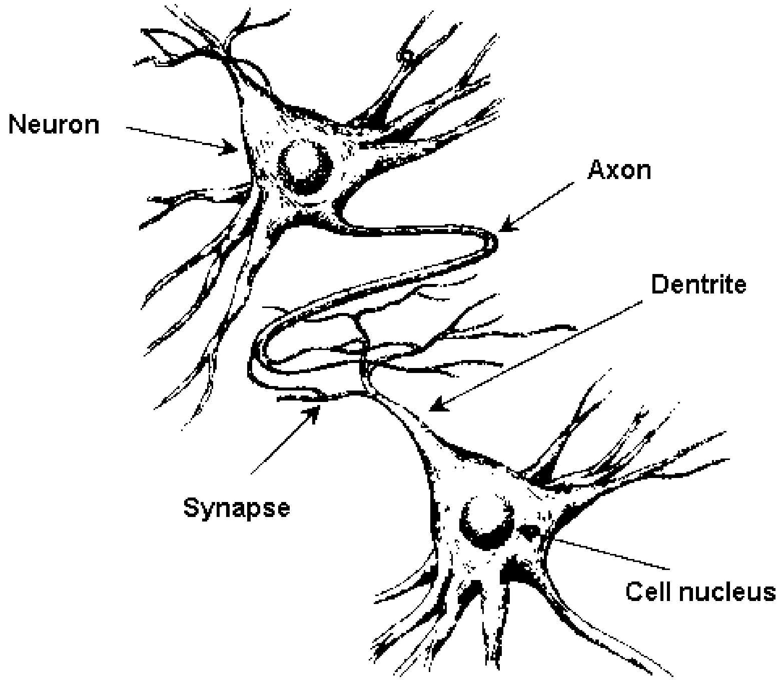
Feed-Forward Artificial Neural Networks

Introduction

- The most universal and versatile classifier is still the human brain
- Starting in the 1940ies, ideas for creating intelligent systems by mimicking the function of nerve/brain cells have been developed
- An *artificial neural network* is a parallel processing system with small computing units (*neurons*) that work similarly to nerve/brain cells

Neurophysiological Background

(cf. [Nauck, Klawonn & Kruse, 1994])



- Every neuron (nerve or brain cell) has a certain electric charge
- Electric charge of connected neurons may raise or lower this charge (by means of transmission of ions through the synaptic interface)
- As soon as the charge reaches a certain threshold, an electric impulse is transmitted through the cell's axon to the neighboring cells
- In the synaptic interfaces, chemicals called neurotransmitters control the strength to which an impulse is transmitted from one cell to another

Feed-Forward Neural Networks

- Note that this is only a very brief overview of the topic!
- We restrict to *feed-forward neural networks*, i.e. simple static input-output systems without any feedback loops between neurons or system dynamics
- Within this class, we consider perceptrons and multi-layer perceptrons (along with the backpropagation algorithm)

Perceptrons

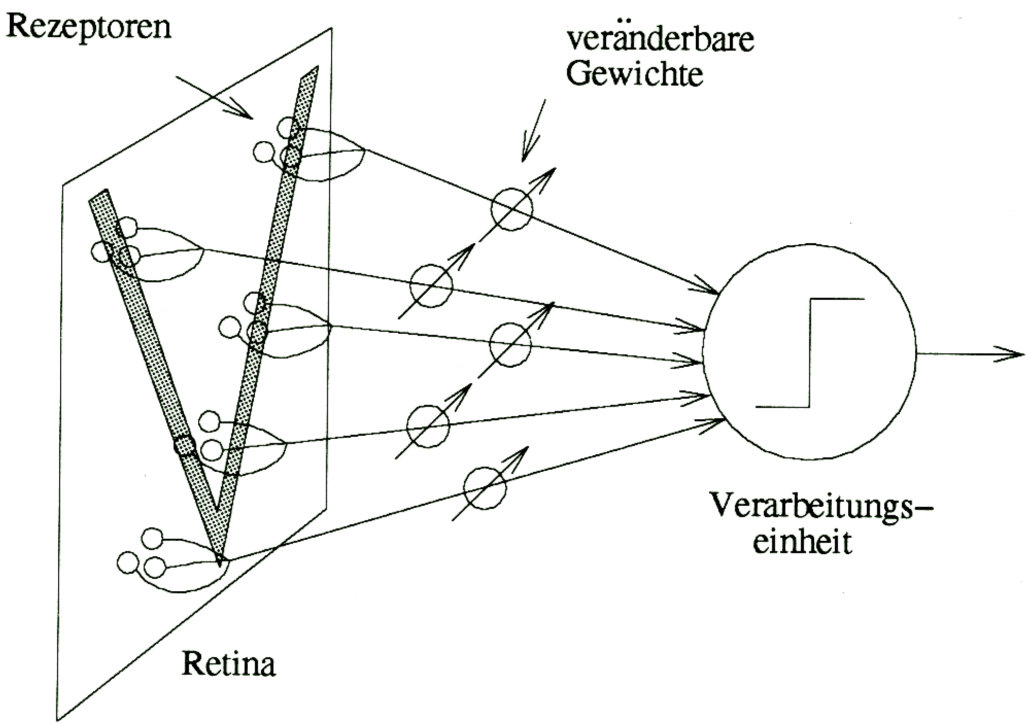
- A perceptron is a simple threshold unit with the following I/O function:

$$f_{\mathbf{w},\theta}(x_1,\ldots,x_p) = \begin{cases} 1 & \text{if } \sum_{i=1}^{p} w_i \cdot x_i > \theta \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

- In analogy to the biological model, the inputs x_i correspond to the charges received from connected cells through the dentrites, the weights w_i correspond to the properties of the synaptic interface, and the output corresponds to the impulse that is sent through the axon as soon as the charge exceeds the threshold θ

The Retina Metaphor

(cf. [Nauck, Klawonn & Kruse, 1994])



The Perceptron Learning Algorithm

1. Given: data set $\{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \{0, 1\}$; learning rate σ; initial weight vector $\mathbf{w}$

2. For $k = 1, \ldots, n$ do:
   - If $f_{\mathbf{w},\theta}(\mathbf{x}_k) = 0$ and $y_k = 1$
     - $\mathbf{w} := \mathbf{w} + \sigma \cdot \mathbf{x}_k$
     - $\theta := \theta - \sigma$
   - Else if $f_{\mathbf{w},\theta}(\mathbf{x}_k) = 1$ and $y_k = 0$
     - $\mathbf{w} := \mathbf{w} - \sigma \cdot \mathbf{x}_k$
     - $\theta := \theta + \sigma$

3. Return to 2. if stopping condition not fulfilled

4. Output: vector of weights $\mathbf{w} \in \mathbb{R}^p$, threshold θ

Perceptrons and Linear Separability

- In case that the two sets

$$X_0 = \{\mathbf{x}_i \mid y_i = 0\} \text{ and } X_1 = \{\mathbf{x}_i \mid y_i = 1\}$$

are linearly separable in $\mathbb{R}^p$, the perceptron learning algorithm terminates and finally solves the learning task

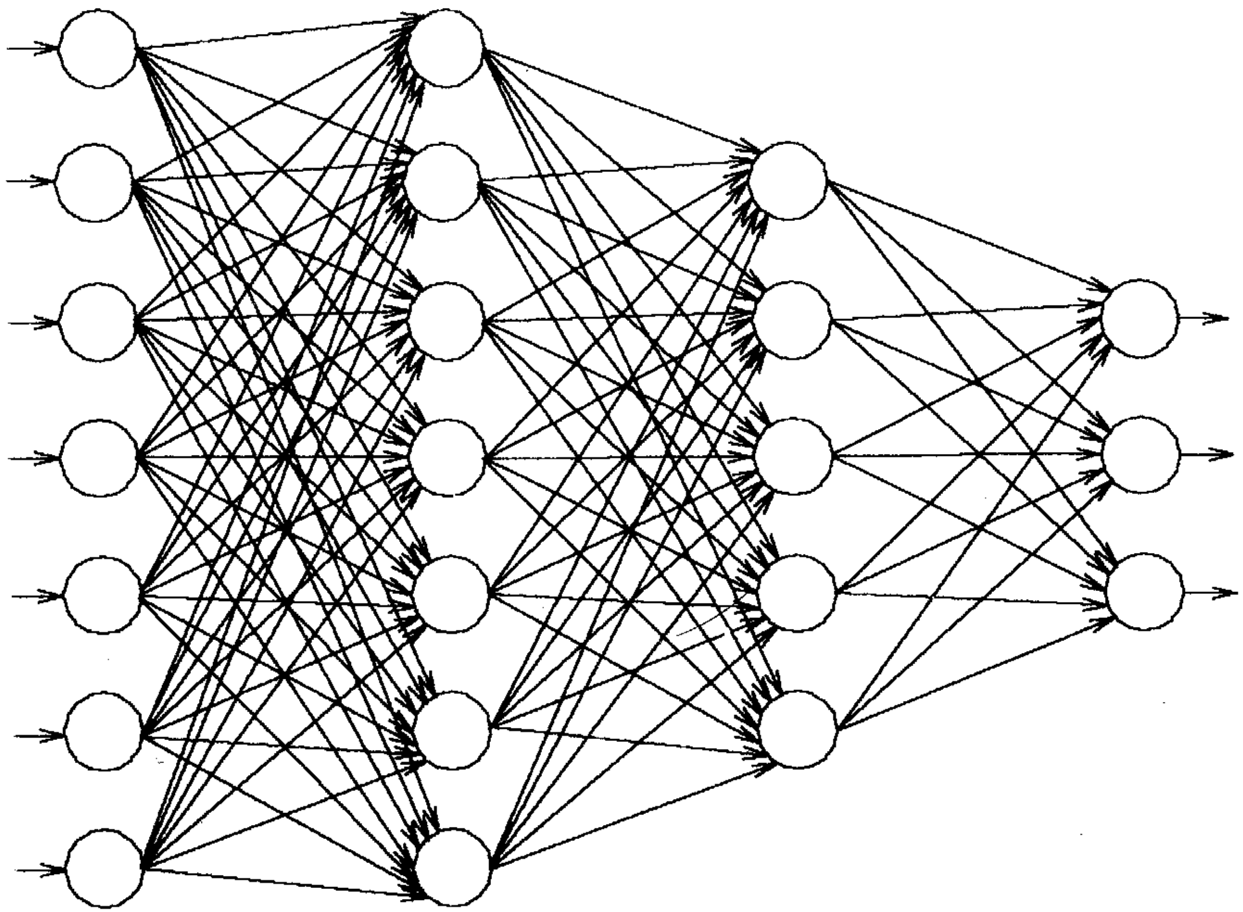
- Note that the solution is not unique and that the learning algorithm just gives one arbitrary solution
- **Perceptrons cannot solve classification tasks that are not linearly separable!**

Multi-Layer Perceptrons

- The only solution to the limitation of linear separability is to introduce intermediate layers
- A multi-layer perceptron is a feed-forward artificial neural network consisting of a certain number of layers of perceptrons
- The output of such a network is computed in the following way: The outputs of the first layer are initialized with the net input $(x_1, \ldots, x_p)$, then the outputs of the other neurons are computed layer by layer using Formula (1)
- The “only problem” is how to find appropriate weights and thresholds that solve a given classification problem

Multi-Layer Perceptrons (cont'd)

(cf. [Nauck, Klawonn & Kruse, 1994])



Some Historical Remarks

- Minsky and Papert, the pioneers of perceptrons, conjectured in the late 1960ies that a training algorithm for multi-layer perceptrons—even if one could be found—is computationally infeasible and that, therefore, the study of multi-layer perceptrons is not worthwhile
- Because of this conjecture, the study of multi-layer perceptrons was almost halted until the mid of the 1980ies

Some Historical Remarks (cont'd)

- In 1986, Rumelhart and McClelland first published the *backpropagation algorithm* and, thereby, proved Minsky and Papert wrong
- It turned out later that the backpropagation algorithm had already been discovered by Werbos in 1974 in his dissertation

Continuous Activation Functions

- The first important idea is to replace the discontinuous threshold function in (1) by a differentiable threshold-like (sigmoid) function φ. Then the output of a neuron is computed as

$$f_{\mathbf{w},\theta}(x_1,\ldots,x_p) = \varphi\Big(\sum_{i=1}^{p} w_i \cdot x_i + \theta\Big) \tag{2}$$

- A common choice is the *logistic function*, i.e. (β is a predefined steepness parameter)

$$f_{\mathbf{w},\theta}(x_1,\ldots,x_p) = \left(1 + \exp\big(-\beta \cdot (\sum_{i=1}^{p} w_i \cdot x_i + \theta)\big)\right)^{-1}$$

Network Topology (1/3)

- Assume that the network has m distinct layers. Let us denote the set of neurons in the j-th layer with U_j
- The first layer (input layer) has $|U_1| = p$ neurons. The output of the k-th input neuron is just the k-th component of the input vector. So, input neurons just propagate the input leaving it unchanged.
- The output layer has $|U_m| = K$ neurons

Network Topology (2/3)

- Given a neuron u, the set of preceding neurons it receives input from is denoted with $\underline{U}(u)$ and the set of consecutive neurons it sends output to is denoted with $\overline{U}(u)$
- For the sake of simplification, we can interpret the bias θ as a weight as well: let us introduce an auxiliary neuron $\tilde{u}$ that always produces a constant output 1; assume that $\tilde{u}$ sends output to every neuron in the network except the ones in the input layer
- Denote $U = U_1 \cup \cdots \cup U_m \cup \{\tilde{u}\}$

Network Topology (3/3)

- Special properties:
  - if $u \in U_1$, $\underline{U}(u) = \emptyset$
  - if $u \in U_m$, $\overline{U}(u) = \emptyset$
  - if $u \in U_j$ (for some $j = 1, \ldots, m-1$), $\overline{U}(u) = U_{j+1}$
  - if $u \in U_j$ (for some $j = 2, \ldots, m$), $\underline{U}(u) = U_{j-1} \cup \{\tilde{u}\}$
  - $\overline{U}(\tilde{u}) = U \backslash U_1$, $\underline{U}(\tilde{u}) = \emptyset$
- Given two neurons u, v such that $v \in \overline{U}(u)$ (and, therefore, $u \in \underline{U}(v)$), the weight connecting u and v is denoted with $W(u, v)$

Computing the Output

- Assume we are given an input vector $\mathbf{x} = (x_1, \ldots, x_p)$
- Let us denote the output of a neuron u with o_u
- Then the output is computed layer by layer in the following way:
  - if u is the k-th neuron in the input layer U_1, then $o_u = x_k$
  - $o_{\tilde{u}} = 1$
  - if $u \in U_j$ for some $j = 2, \ldots, m$:

$$o_u = \varphi(\text{net}_u), \text{ where } \text{net}_u = \sum_{v \in \underline{U}(u)} o_v \cdot W(v, u)$$

The Backpropagation Algorithm (Basic Variant)

1. Given: data set $X = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{y}_i \in [0,1]^K$, learning rate σ, some network topology (with initial weights), activation function φ
2. Select some input $\mathbf{x} \in X$ and propagate it through the network to compute all outputs o_u ($u \in U$)
3. For all $u_k \in U_m$ (u_k is the k-th output neuron) do:
   - $\delta_{u_k} := \varphi'(\text{net}_{u_k}) \cdot (y_k - o_{u_k})$
4. For $j = m-1, \ldots, 2$, step -1 do:
   - For all $u \in U_j$ do:
     - $\delta_u := \varphi'(\text{net}_u) \cdot \sum_{v \in \overline{U}(u)} \delta_v \cdot W(u,v)$
5. For all $v \in U_2 \cup \cdots \cup U_m$ and all corresponding $u \in \underline{U}(v)$:
   - $W(u,v) := W(u,v) + \sigma \cdot o_u \cdot \delta_v$
6. Return to 2. if stopping condition not fulfilled
7. Output: set of weights

Interpreting the Backpropagation Algorithm

- The term backpropagation is motivated by the fact that the errors $(y_k - o_{u_k})$ are backwards propagated through the network (by means of the values δ_u)
- This trick solves the problem that we do not know a desired output for neurons in intermediate layers
- It can be shown that one backpropagation step is one gradient descent step to minimize the error measure

$$E_{\mathbf{x}} = \sum_{k=1}^{K} (y_k - o_{u_k})^2,$$

i.e. the squared error w.r.t. sample $\mathbf{x}$

Enhancing the Backpropagation Algorithm

- Different samples may suggest different (contradicting) changes to the weights; in order to avoid instable oscillating behaviors, low learning rates must be used
- Convergence of this basic variant, therefore, is usually slow
- Batch training is an elegant solution to this issue

The Backpropagation Algorithm (Batch Variant)

1. Given: data set $X = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{y}_i \in [0,1]^K$, learning rate σ, some network topology (with initial weights), activation function φ
2. Set all $\Delta W(u,v) = 0$ ($u \in U \backslash U_m$ and $v \in U_2 \cup \cdots \cup U_m$)
3. For all $\mathbf{x} \in X$
   (a) propagate $\mathbf{x}$ through the network to compute all outputs o_u ($u \in U$)
   (b) For all $u_k \in U_m$ (u_k is the k-th output neuron):
      - $\delta_{u_k} := \varphi'(\text{net}_{u_k}) \cdot (y_k - o_{u_k})$
   (c) For $j = m - 1 \cdots, 2$, step -1, do:
      - For all $u \in U_j$ do
         - $\delta_u := \varphi'(\text{net}_u) \cdot \sum\limits_{v \in \overline{U}(u)} \delta_v \cdot W(u,v)$
   (d) For all $v \in U_2 \cup \cdots \cup U_m$ and all corresponding $u \in \underline{U}(v)$:
      - $\Delta W(u,v) := \Delta W(u,v) + \sigma \cdot o_u \cdot \delta_v$
4. For all $v \in U_2 \cup \cdots \cup U_m$ and all corresponding $u \in \underline{U}(v)$:
   - $W(u,v) := W(u,v) + \Delta W(u,v)$
5. Return to 2. if stopping condition not fulfilled
6. Output: set of weights

Interpreting the Backpropagation Algorithm (cont'd)

It can be shown that the batch variant of the backpropagation algorithm performs a gradient descent with respect to the global error measure

$$E = \sum_{\mathbf{x} \in X} E_{\mathbf{x}} = \sum_{\mathbf{x} \in X} \sum_{k=1}^{K} (y_k - o_{u_k})^2,$$

i.e. the sum of squared errors w.r.t. the sample set X

Multi-Layer Perceptrons Applied to Classification

- Assume we are given a data set data set $X = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{c_1, \ldots, c_K\}$
- Then any multi-layer perceptron with p input neurons and K output neurons can, in principle, be used to solve the classification problem given by X
- In order to bring the data set into the right format, we replace the desired outputs y_i by binary vectors $\mathbf{y}$; if $y_i = c_l$ then

$$\mathbf{y}_i = (0, \ldots, 0, \underbrace{1}_{l\text{-th pos.}}, 0, \ldots, 0)$$

Multi-Layer Perceptrons Applied to Pattern Classification

- Multi-layer perceptrons can be used with feature values as inputs (as in all previous considerations as well)
- However, they can be used on (downsampled) image data as well, where the number of input neurons must be the number of pixels
- Notorious example: character recognition

Multi-Layer Perceptrons Applied to Regression

- Assume we are given a data set data set $X = \{(\mathbf{x}_i, y_i) \mid i = 1, \ldots, n\}$, where $\mathbf{x}_i \in \mathbb{R}^p$ and $\mathbf{x}_i \in \mathbb{R}^K$ (in the simplest case $K = 1$)
- It is clear that only multi-layer perceptron with p input neurons and K output neurons can be used to solve the classification problem given by X

Multi-Layer Perceptrons Applied to Regression (cont'd)

- There are two ways to make multi-layer perceptrons usable for regression:
  - Transforming/scaling all desired output vectors $\mathbf{y}_i$ to $[0, 1]^K$
  - Using so-called linear neurons in the output layer, i.e., for $u \in U_m$, $\varphi(x) = x$ is used, while the other neurons remain unchanged

  The backpropagation algorithm can be left unchanged for both variants

- Multi-layer perceptrons are universal approximators, however, this is only a theoretical result with minor practical value

Overfitting, Accuracy, Generalization

- In the architecture presented here, the numbers of intermediate layers and neurons have to be fixed a priori
- Too many intermediate neurons may result in overfitting effects
- Too few intermediate neurons usually result in a weak classification/regression accuracy
- The methods presented in Unit 12 (accuracy measures, holdout, cross validation) can be used to measure these effects, but only a posteriori

Advantages of Artificial Neural Networks

- Universal
- Easy to apply

Disadvantages of Artificial Neural Networks

- Black box
- Large effort for training
- Design variables can only be chosen heuristically/by trial and error