Unit 5

Artificial Neural Networks

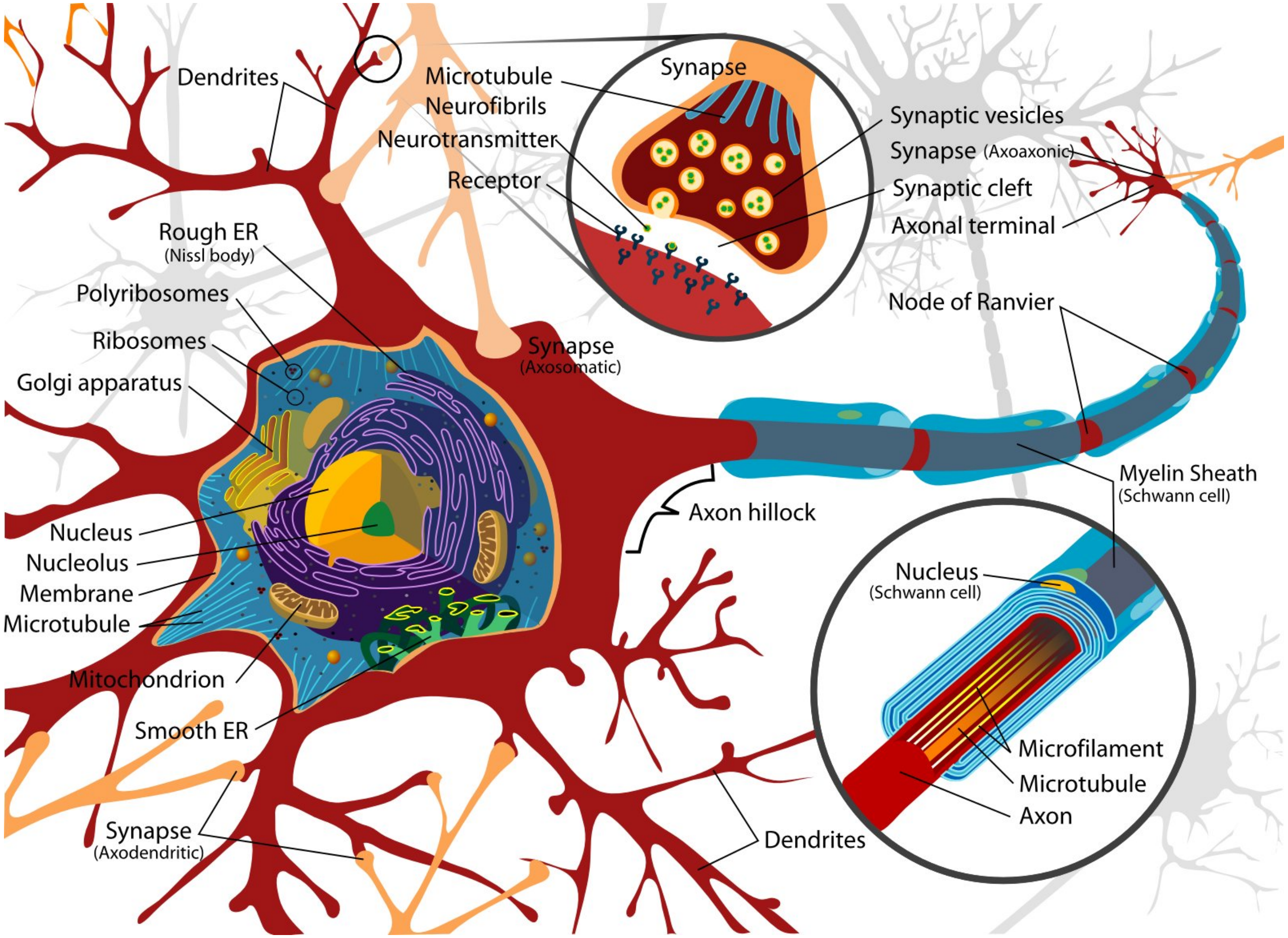
Introduction

- 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

- 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

Neurophysiological Background (cont'd)



[public domain; from Wikimedia Commons]

Feed-Forward Neural Networks

- Note that this is only a very brief and superficial 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:

$$g(\mathbf{x};\mathbf{w},\theta)=\begin{cases}+1 & \text{if } \sum_{i=1}^{d} w_i\cdot x_i>\theta\\ -1 & \text{otherwise}\end{cases} \tag{1}$$

- 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 Perceptron Learning Algorithm

1. Given: data set $\mathbf{Z} = \{(\mathbf{x}^i, y^i) \mid i = 1, \ldots, l\}$, where $\mathbf{x}_i \in \mathbb{R}^d$, $y^i \in \{+1, -1\}$; learning rate σ; initial weight vector $\mathbf{w}$
2. For $k = 1, \ldots, l$ do:
   - If $g(\mathbf{x}^k; \mathbf{w}, \theta) = -1$ and $y^k = +1$
     - $\mathbf{w} := \mathbf{w} + \sigma \cdot \mathbf{x}^k$
     - $\theta := \theta - \sigma$
   - Else if $g(\mathbf{x}^k; \mathbf{w}, \theta) = +1$ and $y^k = -a$
     - $\mathbf{w} := \mathbf{w} - \sigma \cdot \mathbf{x}^k$
     - $\theta := \theta + \sigma$
3. Return to 2. if stopping condition not fulfilled
4. Output: vector of weights $\mathbf{w} \in \mathbb{R}^d$, threshold θ

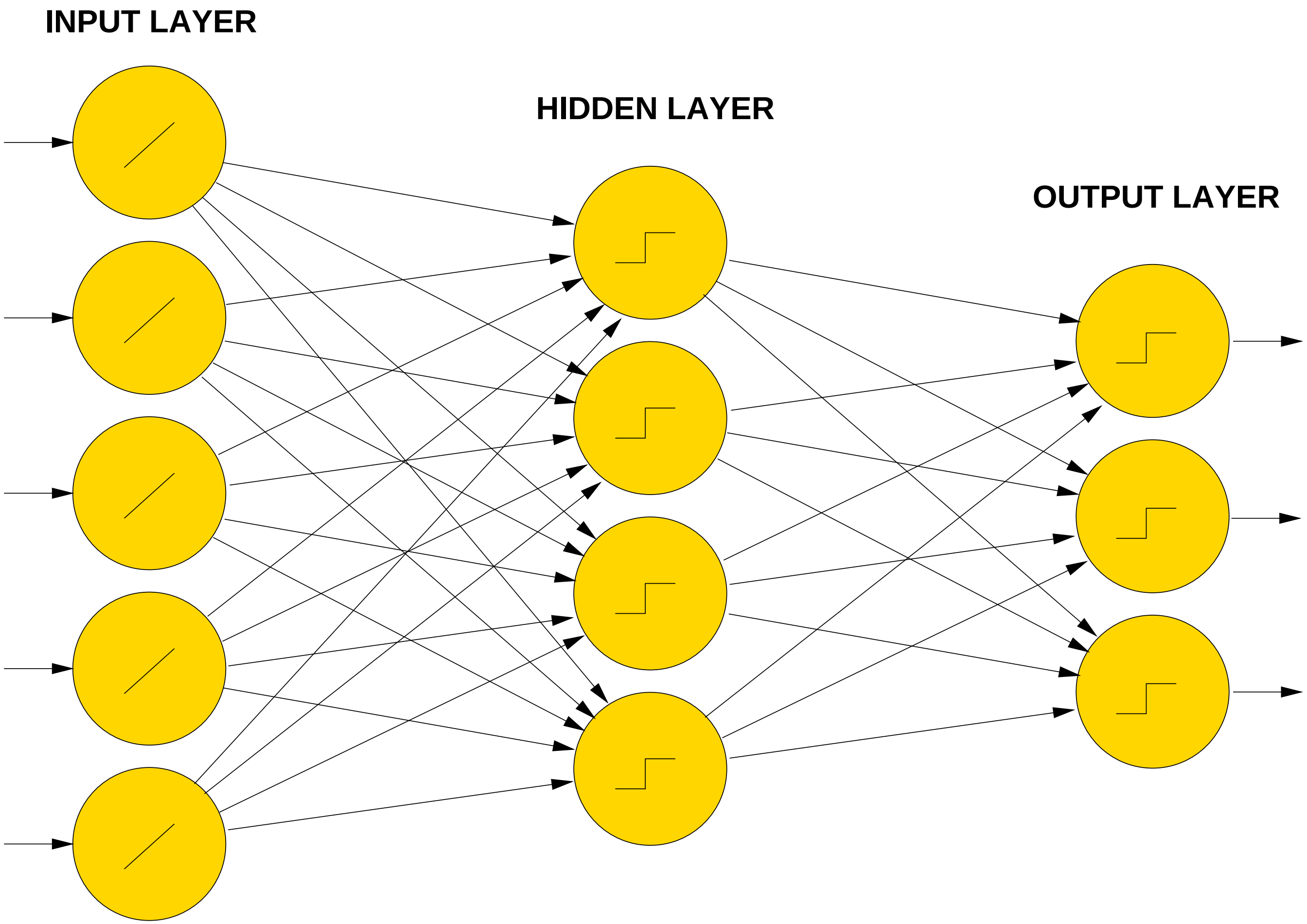
Perceptrons and Linear Separability

- In case that the data set $\mathbf{Z}$ is linearly separable in $\mathbb{R}^d$, 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 $(x_1, \ldots, x_d)$, 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)



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 *back-propagation 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

$$g(\mathbf{x}; \mathbf{w}, \theta) = \varphi\Big(\sum_{i=1}^{d} w_i \cdot x_i + \theta\Big) \qquad (2)$$

- For -1/+1 data, a common choice of the activation function is the *hyperbolic tangent* $\varphi(x) = \tanh(\beta x)$ (which is nothing else but a transformation of the sigmoid function to the interval $[-1, +1]$; β is a steepness parameter)

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 $|U_1| = d$ 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 $\mathbf{x} = (x_1, \ldots, x_d)$
- 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 $j = 2, \ldots, m$:

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

The Backpropagation Algorithm (Basic Variant, aka "Vanilla Backprop")

1. Given: data set $\mathbf{Z} = \{(\mathbf{x}^i, \mathbf{y}^i) \mid i = 1, \ldots, l\}$, where $\mathbf{x}^i \in \mathbb{R}^d$ and $\mathbf{y}^i \in [-1, +1]^K$, learning rate σ, some network topology (with initial weights), activation function φ
2. Select a sample $(\mathbf{x}, \mathbf{y}) \in \mathbf{Z}$ 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:
     - $\delta_u := \varphi'(\text{net}_u) \cdot \sum\limits_{v \in \overline{U}(u)} \delta_v \cdot W(u, v)$
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

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

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

The Backpropagation Algorithm (Batch Variant)

1. Given: data set $\mathbf{Z} = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \ldots, l\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $\mathbf{y}_i \in [-1, +1]^K$, 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
         - $\delta_u := \varphi'(\text{net}_u) \cdot \sum_{v \in \overline{U}(u)} \delta_v \cdot W(u, v)$
   (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

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

i.e. the sum of squared errors w.r.t. the training set $\mathbf{Z}$.

Multi-Layer Perceptrons Applied to Classification

- Assume we are given a data set data set $\mathbf{Z} = \{(\mathbf{x}^i, y^i) \mid i = 1, \ldots, l\}$
- If we have a binary classification problem, i.e. $y_i \in \{-1, +1\}$, we can solve it with a single output neuron ($K = 1$),
- If we are given a problem with $K > 2$ classes, i.e. $y_i \in \{1, \ldots, K\}$, we can use a network with K output neurons. The labels have to be mapped to K-dimensional output vectors $\mathbf{y}^i$ in the following way:

$$y_j^i = \begin{cases} +1 & \text{if } y_i = j \\ -1 & \text{otherwise} \end{cases}$$

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

- The approach proposed above corresponds to a one-versus-the-rest approach (compare with Unit 4); the final classification is determined in a winner-takes-it-all fashion
- The weights connecting the $m-1$-st layer to individual output neurons are optimized independently of each other; however, as the discriminant functions are on the same scale, this is not posing a difficulty

Multi-Layer Perceptrons Applied to Regression

- Assume we are given a data set data set $\mathbf{Z} = \{(\mathbf{x}^i, y^i) \mid i = 1, \ldots, l\}$, where $\mathbf{x}^i \in \mathbb{R}^d$ and $\mathbf{y}^i \in \mathbb{R}^K$ (in the simplest case $K = 1$)
- There are two ways to make multi-layer perceptrons usable for regression:
  1. Transforming/scaling all desired output vectors $\mathbf{y}^i$ to $[-1, +1]^K$
  2. 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; in this case, the outputs of the $m-1$-st layer can be understood as basis functions; the output is a linear combination of these basis functions

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

- The backpropagation algorithm works for both variants without any modification
- Multi-layer perceptrons are universal approximators, however, this is only a theoretical result with minor practical value

Complexity of Neural Networks

- In the architecture presented here, the numbers of hidden layers and neurons have to be fixed a priori
- The complexity of a neural network increases with an increasing number of hidden layers/neurons
- There are several regularization methods (e.g. early stopping, weight decay, noise injection) to additionally control/limit the complexity of a neural network
- In principle, cross-validation can be used to estimate optimal design parameters, but this may be tedious because of longer training times and higher dimensionality of design parameters

Pro’s and Con’s of Artificial Neural Networks

Advantages:

- Universal
- Easy to apply

Disadvantages:

- Black box
- Large effort for training
- Solution is not guaranteed to be a global minimum, but only a local one