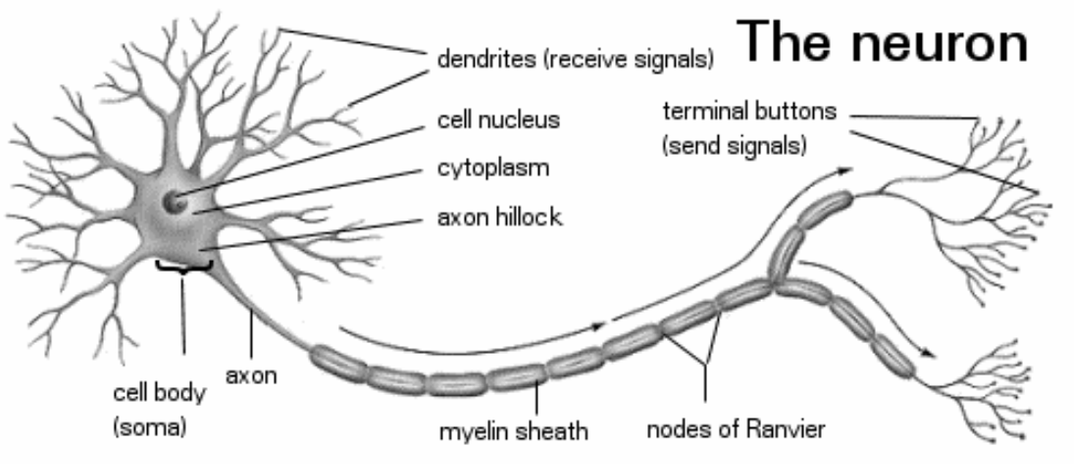
Neural Networks

Boris Flach

A. (Biological) Neurons



Physiology: cell body, dendrites, axons, synapses

Information processing:

- Electrical pulse arrives at a synapse, release of neurotransmitter evokes (local) change of membrane potential
- Two types of synapses: excitatory, inhibitory
- If several spikes arrive simultaneously $\Rightarrow$ global depolarisation of the membrane $\Rightarrow$ electr. pulse travelling along the axon is generated

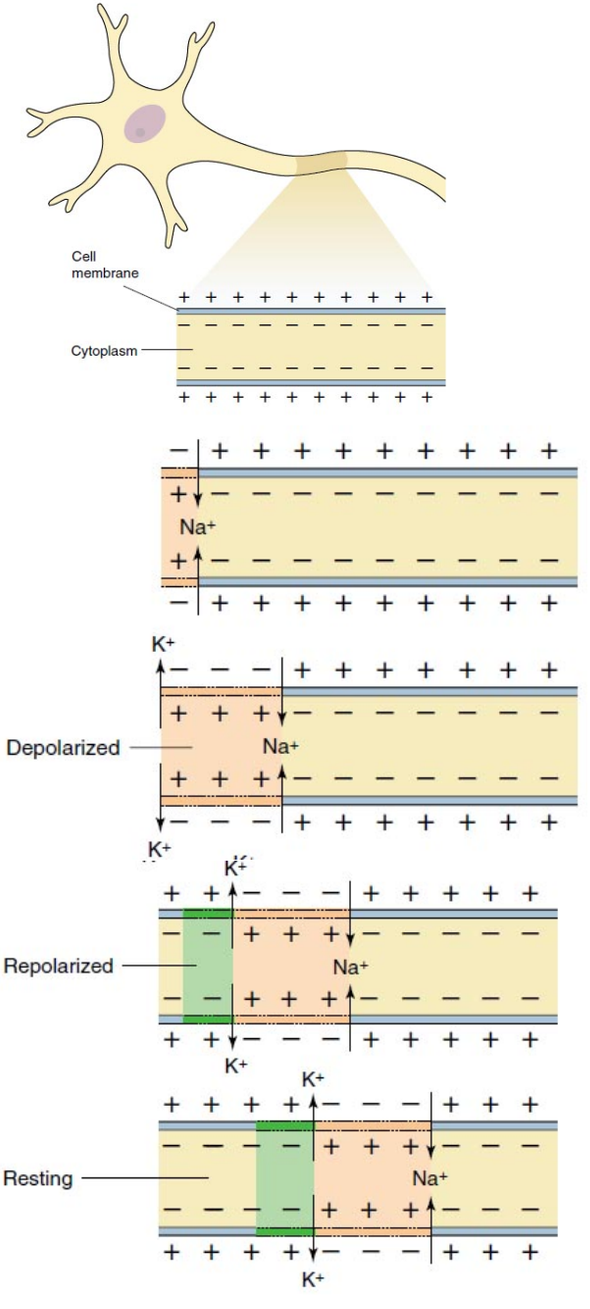
A. (Biological) Neurons

Neuron at rest: Interplay of active ion pumps (Na^+ out, K^+ in) and voltage gated ion channels leads to

- 70mV negative polarisation of the cytoplasm
- sodium excess outside and potassium excess inside

Action potential: Sufficiently strong excitation of the neuron $\Rightarrow$ depolarisation of the membrane $\Rightarrow$ electrical pulse travelling along the axon (interplay of Na^+ and K^+ voltage gated channels).

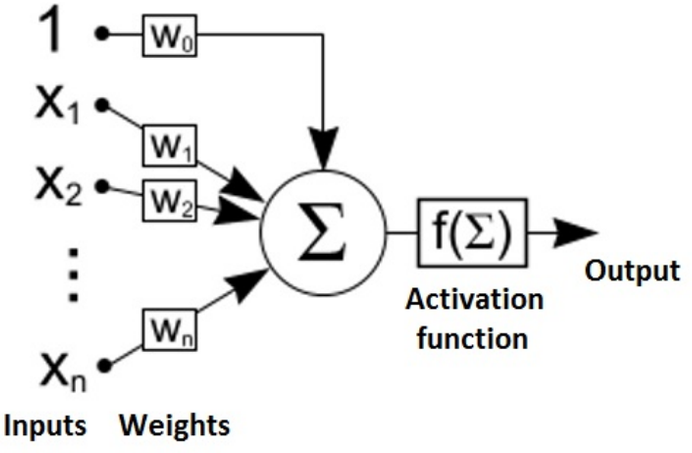
The travelling spike: reaches the synapses at the end of the axon and triggers release of neurotransmitter, which in turn changes the polarisation of other neurons.



B. Artificial Neuron Models

Binary valued threshold neuron (McCulloch and Pitts '49)

$$y = \theta(\boldsymbol{w} \cdot \boldsymbol{x} - b)$$

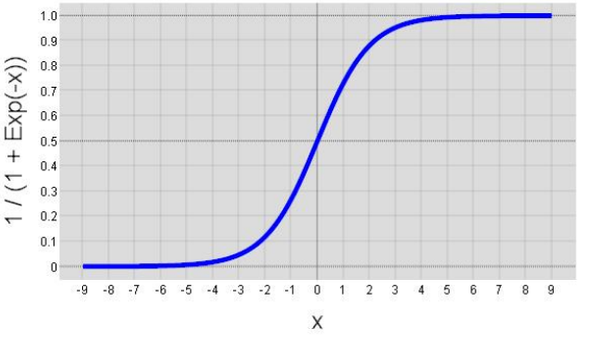


- $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ input
- $y \in \{0, 1\}$ output
- $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ weights
- $b \in \mathbb{R}$ threshold
- $\theta(z)$ - Heaviside step function

Clearly: this is a linear classifier! can be learned by Perceptron algorithm or SVM methods.

Graded response neuron:

- real valued output $y \in \mathbb{R}$
- activation function $f : \mathbb{R} \to \mathbb{R}$, asymptotically bounded, e.g. $f(z) = 1/(1 + e^{-z})$
- $y = f(\boldsymbol{w} \cdot \boldsymbol{x} - b)$



C. Neural networks as classifiers/regressors

Any **binary valued classifier** can be approximated by a three layer Feed Forward Network (FFN) of binary threshold units.

Idea:

- represent the classifier by a $\{0,1\}$-valued function defined on the feature space, approximate its support by a union of convex polyhedra.
- convex polyhedron $=$ intersection of half spaces, can be implemented by a two layer FFN.
- Hence, one more layer is needed. It consists of one neuron which implements the union via an OR function.

Problems: How to choose the appropriate size of the network? How to learn its weights and thresholds?

C. Neural networks as classifiers/regressors

Theorem 1. (Kolmogorov, 1957) Any continuous function F defined on a compact set in $\mathbb{R}^n$, e.g. $F\colon \mathbb{I}^n \to \mathbb{R}$ can be represented in the form

$$F(x_1, \ldots, x_n) = \sum_{j=1}^{2n+1} g_i\Big(\sum_{i=1}^{n} f_{ij}(x_i)\Big)$$

where functions g_i are continuous and f_{ij} are monotone and continuous functions independent of F.

Theorem 2. (Cybenko, 1989) Any continuous function F defined on a compact set in $\mathbb{R}^n$, e.g. $F\colon \mathbb{I}^n \to \mathbb{R}$ can be approximated arbitrarily well by

$$F(\boldsymbol{x}) \approx \sum_{i=1}^{k} \alpha_i f(\boldsymbol{w}_i \cdot \boldsymbol{x} - b_i)$$

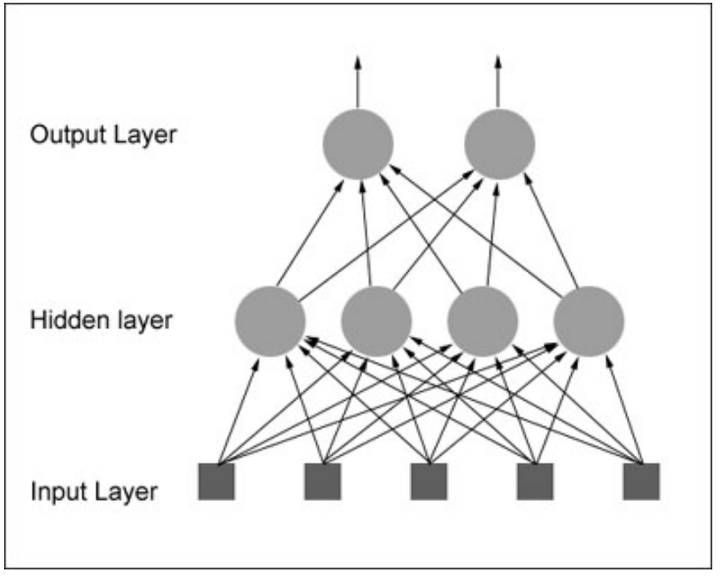
where f is a continuous sigmoid function.

Remark 1. (Lusin, 1912) Any measurable (decision) function is nearly continuous.

D. Learning a FFN by Error back propagation

Suppose we want to implement a k-valued classifier for features $\boldsymbol{x} \in \mathbb{R}^n$ by a FFN of graded response units.

- network input $\boldsymbol{x} \in \mathbb{R}^n$
- k classes $\Rightarrow$ network has k neurons in the output layer. Use "1 out of k" coding.
- training data $\mathcal{T} = \{(\boldsymbol{x}^j, \boldsymbol{y}^j) \mid j = 1, \ldots, \ell\}$.
- outputs of neurons in layer $m = 1, 2, \ldots, q$



$$y_i^{(m)} = f\Big(\sum_{j=1}^{n_{m-1}} w_{ij}^{(m)} y_j^{(m-1)} - b_i^{(m)}\Big)$$

Objective function

$$E(\mathcal{T}, W) = \sum_{(x,y)\in\mathcal{T}} \sum_{i=1}^{k} \Big[y_i - y_i^{(q)}\big(W, \boldsymbol{x}\big)\Big]^2 \to \min_W$$

Minimise $E(\mathcal{T}, W)$ by gradient descent.

D. Learning a FFN by Error Back Propagation

For simplicity: calculate the gradient of the objective function for one element $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}$ of the training data.

$$E = \sum_{i=1}^{k} \Big[y_i - y_i^{(q)}\big(W, \boldsymbol{x}\big) \Big]^2$$

Output $y_i^{(q)}\big(W, \boldsymbol{x}\big)$ is a composition of functions

$$y_i^{(q)}\big(W, \boldsymbol{x}\big) = \boldsymbol{f}_{\boldsymbol{w}^q} \circ \boldsymbol{f}_{\boldsymbol{w}^{q-1}} \circ \ldots \circ \boldsymbol{f}_{\boldsymbol{w}^1}(\boldsymbol{x}),$$

hence, apply chain rule in order to compute gradient $\nabla_{\boldsymbol{w}^m}$, $m = 1, \ldots, q$.

Notation: $h_i^{(m)} = \sum_j w_{ij}^{(m)} y_j^{(m-1)} - b_i^{(m)}$ net input of the i-th neuron in the m-th layer.

D. Learning a FFN by Error Back Propagation

We have

$$\frac{\partial E}{\partial w_{ij}^{(m)}} = \frac{\partial E}{\partial h_i^{(m)}} \cdot y_i^{(m-1)}$$

$$\frac{\partial E}{\partial b_i^{(m)}} = -\frac{\partial E}{\partial h_i^{(m)}} \cdot 1$$

Hence, the problem is to compute the derivatives w.r.t. the net inputs. This can be done recursively starting from the output layer

- output layer

$$\frac{\partial E}{\partial h_i^{(q)}} = -2\Big[y_i - y_i^{(q)}(W, \boldsymbol{x})\Big] \cdot f'\big(h_i^{(q)}\big)$$

- back propagation

$$\frac{\partial E}{\partial h_j^{(m)}} = \sum_i \frac{\partial E}{\partial h_i^{(m+1)}} \cdot \frac{\partial h_i^{(m+1)}}{\partial h_j^{(m)}} = \sum_i \frac{\partial E}{\partial h_i^{(m+1)}} \cdot w_{ij}^{(m+1)} \cdot f'\big(h_j^{(m)}\big)$$

D. Learning a FFN by Error Back Propagation

Advantages:

- handles well problems with multiple classes
- handles both classification and regression

Disadvantages:

- no guarantee to reach the global optimum
- not clear how to choose the appropriate network structure and size (network too big $\Rightarrow$ over-fitting, network too small $\Rightarrow$ not enough expressive power to handle the task)

Generalisations:

- stochastic neurons
- bidirectional interactions

$\Rightarrow$ Markov Random Fields and Deep Learning.