8.1 The Biological Neuron

8.1.1 Structural description

The brain is composed of about $10^{11}$ neurons (nerve cells) of many different types. Fig.1 is a schematic drawing of a single neuron.

Tree-like networks of nerve fiber called dendrites are connected to the cell body (or soma) where the cell nucleus is located. Extending from the cell body is a single long fiber called the axon, which usually branches into many strands, at the ends of which are the synapses - where communication with other neurons takes place. Considering inter-neuronal communication, the dendrites are used as input channels, the axon as an output channel and the synapses as the interfacing areas.

![Schematic drawing of a typical neuron](image)

Fig.1 Schematic drawing of a typical neuron

8.1.2 Functional description

The transmission of a signal from one cell to another at a synapse is a complex chemical process in which specific substances called neurotransmitters are released from the sending side to the junction. The effect is to raise or to lower the electrical potential inside the body of the receiving cell (inhibitory vs excitatory synapses). If this potential reaches a certain threshold (when the composed input from the cell’s neighboring neurons is high enough), a
pulse or an action potential (of fixed strength and duration) is sent down the axon. This pulse, in turn, branches through the axon to synaptic junctions with other cells. After firing, the cell has to wait a period of time called the refractory period before it can fire again.

\[ \text{Fig. 2 Neuronal dynamic process} \]

\section*{8.1.3 Network}

\textbf{Efficiency due to structure of cortex}

The axon of a typical neuron makes about 10 thousand synapses with other neurons (which is much smaller than the total number of neurons) where most of those inter-neuronal connections are local.

High connectivity while using (mostly) local connections is enabled thanks to the topology of the cortex (where two points which are distant on a flat map of the cortex can be closer on the real cortex due to its folded structure), thus keeping the ”wiring” needed for the connections relatively small.

\textbf{Redundancy cause robustness}

- The large number of synapses each neuron makes means that there are many terms in the sum of inputs each neuron ”computes”, which means that errors in a few terms will probably be inconsequential. This tells us that such a system can be expected to be robust and its performance will hardly be impacted by noise.

- It was experimentally shown that the same computation process takes place in parallel in several different locations in the brain. Due to continued inhibition, whole parts of the brain are redundant. When some area
of brain is damaged then there is a process of excitation enabling "backup" parts of the brain to fulfill the functions.

### 8.1.4 Learning

Learning in biological neural network (or at least on aspect of it) can be considered as the change in the impact of one neuron on the other. In other words change in the impact of the neurotransmitters in a given synapse.

Generally speaking, there are 2 types of neurotransmitters:

1. Those which are essential for learning.
2. Those which only responsible for transformation of data.

The question at had is what do we consider as data? This is an open question. Among the suggestions we can find:

- Quantity of neural transmitters.
- Neurons firing rate.

### 8.2 The artificial Neuron

#### 8.2.1 single neuron

A single neuron can be generally described as a processing unit which gets a stimuli from its neighbors and respond wrt to a given activation function.

**Fig.3 A single artificial neuron**
Static neurons

• **Threshold/Binary neuron (simple perceptron)**
  First introduced by McCulloch and Pitts[1943]. In that simple model, the single neuron is a binary threshold unit which computes the weighted sum of its inputs from other neurons and outputs one or zero according to its threshold: \( n_i(t+1) = \text{Sign}(\sum_j w_{i,j} n_j(t) - \mu_i) \). where \( n_i(t) \) is the output of neuron \( i \) at time \( t \), \( w_{i,j} \) is the weight of the synapse (or the edge) between neurons \( i \) and \( j \), \( \mu_i \) is the units’ threshold value, alternatively defined (for a biological intuition) as its resting potential.

• **Linear neuron**
  Here, the single neuron computes the weighted sum of the inputs from other neurons and outputs it (activation function is the identity function):
  \[ n_i(t+1) = \sum_j w_{i,j} n_j(t) - \mu_i. \]

• **Differentiable monotonic output (sigmoid) neuron**
  Here, the single neuron computes the weighted sum of the inputs from other neurons and activates a sigmoid function such as \( f(z) = \frac{2}{1+e^{-z}} - 1 \).
  Here output of neuron \( i \) is: \( n_i(t+1) = f(\sum_j w_{i,j} n_j(t) - \mu_i). \)

Dynamic neurons (Spiking neurons)

Adding the time element into the system. The summation is the stimulus integration over time. The problem with this kind of system is the difficulty of analysis. However, it is very powerful. It can be simplified and thus be reduced to a nearly equivalents but much more simply to analyze systems: recurrent networks.

### 8.3 Artificial Neural networks

#### 8.3.1 Architectures

• **Direction of flow**
  – Feed forward: each unit during computation gets input independent on its output.
  – Recurrent: a unit can get input from the unit depending on its output (possibly indirectly).

• **Connectivity**
  – Fully connected: every unit gets input from any other unit.
8.3. ARTIFICIAL NEURAL NETWORKS

- Fully layer connected: the neurons are divided into layers so that each unit gets input from all the neurons of previous layer.

- Generalized layer organization: the units are layered so that each units output is only dependent on the outputs of layer previous to its own layer.

**Remark** Usually we prefer sparse connection between layers. This structure is derived from the biological cortex organization. The connections between layers are local under some topological transformation. Therefore, the connections between 2 layers can be presented as some sparse matrix.

8.3.2 Capability

- **Threshold neurons:**
  A network consisting of such neurons is capable of arbitrary strong computations. It can perform any computation the digital computer can, provided the suitably chosen weights.

- **Neurons with Sigmoid Activation Function:**
  - Feed-forward neural network consisting of such neurons and with one hidden layer can approximate any continuous function to arbitrary accuracy.
  - Similarly, with two hidden layers we can approximate *any function* to arbitrary accuracy.

**Linear separability**

Single-layer (no hidden layers) Neural networks with sigmoid or threshold activation functions can represent only *linearly separable* classification rules. Thus, single neurons are unable to express more complex functions(such as XOR). Therefore we need hidden layers in order to express some complex nonlinear concepts. Nevertheless, in high dimension feature space(or artificially extended one) separating hyperplane can actually solve lots of classification problems. Linear separation is preferable for its simplicity.
In addition, linear separation is usually preferable as an atomic unit of much larger learning processes.
Some techniques(SVM) achieve the tradeoff between the input and model dimensions. Increasing the data dimensionality by polynomial transformations can cause linear separability and substantial model simplification.
8.4 Learning methods employing neural networks

8.4.1 Hebbs postulate

\[ \Delta w_{B,A} = F(y_B, X_A) \]

Biologically: when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process of metabolic change takes place in one or both cells such that A’s efficiency as one of the cells firing B is increased.

Hebbs postulate revisited.

Stent (1973), Changeux and Danchin (1976). If two neurons on either side of synapse are activated simultaneously (synchronously) then the strength of that synapse is selectively increased. In other words, those who fire together, wire together.

In general, there are three types of learning: supervised, unsupervised and reinforcement learning. We shall concentrate on the first two types.

8.5 Unsupervised learning

In this type of learning we have no information about what the output should be. Therefore, it is usually being used to find correlations, similarities, classification and feature selection of the input. For example, it conducts PCA (Oja, Sanger) and clustering (ART).

8.5.1 PCA (Principle Component Analysis)

A statistical method for reducing the dimension of a data set. Given data of dimension \( n \) we want to decrease the linear dimension of the data to \( m < n \) (compression) with leftover variance minimized. In other words if \( n \) is the dimensionality of the data (superficial dimensionality) then \( m \) is the number of independent variables sufficient to describe the data (intrinsic dimensionality).

The goal in the PCA process is to provide \( m \) vectors such that input vectors projection on their span is as close as possible to the initial one over the probability space. Formally, those vectors produce space \( V_m \) where \( \max_{z \in \mathbb{R}^n/\|z\|=1} \text{Var}(z^t(x - P_m x)) \) is minimized. We call this set of vectors the set of principle components.
8.5. UNSUPERVISED LEARNING

Brief analysis of PCA

Let $X$ be an $n$-dimensional probability space then for every $\vec{x} \in X$ define $E\vec{x} = E(x_1), ..., E(x_n)$. Which means that every coordinate in the vector is considered as a separate random variable. For any constant matrix $A$ and random vector $\vec{x}$ it follows that $E(A\vec{x}) = AE\vec{x}$.

Given $n$ random variables their covariance matrix $C$ is defined by $C_{i,j} = \text{Cov}(x_i, x_j)$. Note that $C$ is positive and symmetric by definition.

**Claim 8.1** The set of $m$ principle components is the set of the first $m$ eigenvectors of $C$ taken by the order of the corresponding eigenvalues from large to small.

**Explanation:**

First, a little algebra: For a random vector $\vec{x}$ and for every constant $\vec{w}$. It follows that $\text{Var}(\vec{w}^T \vec{x}) = \vec{w}^T C \vec{w}$.

Let $W$ be an orthogonal transformation matrix and let $\vec{y}$ be the representation of $\vec{x}$ on the new basis (consists of $W$’s columns), $y = W^T x$ (x and y are $n$-dimension random vectors). It then follows that $C'$, the covariance matrix of $\vec{y}$ satisfies $C'' = W^T C W$, where $C$ is the covariance matrix of $\vec{x}$.

If $C'$ is a diagonal matrix then $W$ is defined uniquely (with possible switch of columns) and in addition, the columns of $W$ are the eigenvectors of $C$ and the corresponding $C'$ diagonal values are their corresponding eigenvalues. First $m$ columns with largest eigenvalues define the optimal $P_{1:m}$ space of projection.

Let us note that since $C'$ is diagonal matrix, then, for each $y_i$, $\text{Var}(y_i)$ is the corresponding eigenvalue $\Delta_i$ by definition. $y_i$ are independent, since $C'$ is orthogonal.

**The recursive method of finding eigenvectors $W_j$ with monotonically decreasing eigenvalues:**

Define $w_1$ the vector a maximizing the $\text{Var}(a^T x)$ under constraint that $\sum_{i=1}^n a_i^2 = 1$. Continue the process recursively in the space orthogonal to $w_1$. $X' = X - \left\langle X, w_1 \right\rangle w_1$ is the new probability space for recursion. This way $w_2..w_n$ can be extracted. Clearly, for $w_1$ eigenvector with maximum eigenvalue can be chosen for it providing the direction of maximal variance.

Then, recursively $w_2$ eigenvector with next largest eigenvalue will be chosen since it is on the direction which maximizes the variance in $X'$ that is orthogonal to $\text{span}(w_1)$.

**Implementation**

**Definition** Oja update rule for a one-layered, single output neural network:

$$w^{k+1} = w^k + \rho(x^k - y^k)y^k,$$

where $w$ is the weights vector, $x$ is the input vector $y$ is the output and $k$ is the stage index.

Note that in Oja model, the output is defined as $y = w^T x$ (linear network).
Claim 8.2 Oja rule causes the neuron weights $w^k$ to converge to the first principle component alleles so it computes the value of the first principle component.

Explanation:
Using simplified Hebbian rule for one neuron, $w^{k+1} = w^k + \rho y^k x^k$. Compute the expected weight change $E\Delta w = \rho E(yx) = \rho E(xx^T w)$. assuming that $x$ is driven each time independently from $w$ then $E\Delta w = \rho E(xx^T)E(w)$. By definition covariance matrix $C$ is $E(xx^T)$. Continuous version of the equation is $d w = C w$, therefore $w$ is driven to infinite magnitude in the direction parallel to the eigenvector of $C$ with the largest eigenvalue.

Oja rule is derived from the following by normalizing $w$ at each step:

$$w'(k+1) = w((k) + \rho x(k)y(k))$$

$$w(k+1) = \frac{w'(k+1)}{\|w'(k+1)\|}$$

$$\|w'(k+1)\|^2 = w'^T(k+1)w'(k+1) = 1 + 2\rho y^2(k) + O(\rho^2)$$

$$w(k+1) = w'(k+1)[1 + 2\rho y^2(k) + O(\rho^2)^{\frac{1}{2}}]$$

$$w(k+1) \approx w'(k+1)[1 - \rho y^2(k)]$$

$$w(k+1) \approx [w(k) + \rho x(k)y(k)][1 - \rho y^2(k)]$$

$$w(k+1) = w(k) + \rho x(k)y(k) - \rho w(k)y^2(k) - \rho^2 y^3(k)x(k)$$

$$w(k+1) = w(k) + \rho (x(k) - y(k)w(k))y(k)$$

Definition Sanger rule for a single-layered multiple output network:

$$\Delta w_{ji} = \rho(x_j - \sum_{k=1}^{i-1} w_{kj}y_k) y_i$$

Explanation:
Here we derive the update rule for the first $m$ principle components recursively. Each component is approximated by the weights of a different neuron, so the number of the principle components is equal to the number of the (output)neurons.

For $i=1..m$ lets assume that all $w_j < i$ converge to the $j$ 's component correctly. So in order to use Oja’s rule for component $i$, discard the $x$ projection on $P_{v_{i-1}}$ from the input similarly to Graham-Shmidt process with the use of $w_1..w_{i-1}$ assuming they converge correctly to the corresponding components. This scheme is accomplished in the following way:

Assuming that we want to apply Oja’s rule to the $i$-th principal component and that $\vec{w}_1..\vec{w}_{i-1}$ are close enough to the corresponding principal components. The proper input vector for the $i$-th neuron should be (by Graham-Shmidt) $\vec{x} - \sum_{j=1}^{i-1} \vec{w}_j y_j$ where $y_j$ is the output neuron
8.5. UNSUPERVISED LEARNING

j and \( \vec{w}_j \) is the weights vector of neuron j (1 > j > i - 1). The final formula follows from direct application of Oja’s rule and orthogonality.

Example

Characterization of turtle shells.
Parameters are: \( L \) – shell length, \( W \) – shell width, \( H \) – shell height. We get as input, Data for 24 turtles.

The resulting Covariance matrix between the three random variables is:

\[
\begin{bmatrix}
L & \text{W} & C \\
451.33 & 271.17 & 168.7 \\
271.17 & 171.73 & 103.29 \\
168.7 & 103.29 & 66.65
\end{bmatrix}
\]

Eigenvectors are 0.81 -0.54 -0.20

0.49 0.83 -0.24

0.30 0.10 0.94

and variances (corresponding for each vector) are (680.4, 6.5, 2.9).

It can be seen that the first principal component accounts for 98% of the variance. Which means that the data dimensionality can be reduced from 3D to 1D without losing a lot of information. When given a vector \(< L, W, H >\) we consider \( Y = 0.81L + 0.5W + 0.31H \).

Corresponding net will thus look like:

![Corresponding net](image)

8.5.2 ICA (Independent component analysis)

In the case of Gaussian input the linear principle components are the sufficient statistics for analysis. However, in the real world data we often are interested in more interesting and
troublesome components, far from Gaussian behavior. The group of techniques denoted as ICA targets extraction of such sort of components.

8.5.3 ART(Adaptive Resonance Theory)

ART is a method for dynamic clustering of input data. Continuous training on non-stationary inputs requires dealing with the stability vs plasticity dilemma. We have to let the network adapt and still to prevent destroying past training. The basic principles of this theory are formulated in ART(Adaptive Resonance Theory).

Here we shall present the simplest algorithm, ART1:

Every training iteration consists of taking a training example $x^k$ and examining existing prototypes (represented by weight vectors $\vec{w}_j$) to see whether we can find a prototype $\vec{w}_i$ to match $x^k$ (according to similarity test based on a preset matching threshold).

When an input vector (or- pattern) $x^k$ is added to the cluster represented by $\vec{w}_i$, then $\vec{w}_i$ is modified to make it better match $x^k$. If no prototype matches $x^k$, then $x^k$ becomes the prototype for a new cluster.

The input vector $\vec{x}$ is restricted to binary values. Each learning cluster, say cluster $j$, is represented by the binary weight vector $\vec{w}_j$ of the $j$-th prototype unit.

Every time an input vector $\vec{x}$ is presented to the ART1 net, each existing prototype unit computes a normalized output: $y_j = \frac{\vec{w}_j^T \vec{x}}{\|\vec{w}_j\|^2}$ and fits it to the winner-take-all net for determining the winner unit $i$.

Subject to further verification, the weight vector of the winning unit $\vec{w}_i$ now represents a potential prototype for the input vector. The verification comes in the form of passing the following two tests:

1. In order to pass the first test the input must be close enough to the winner prototype $\vec{w}_i$: $y_i = \frac{\vec{w}_i^T \vec{x}}{\|\vec{w}_i\|^2} > \frac{\|\vec{x}\|^2}{n}$ Passing that test guarantees that a sufficient fraction of the $\vec{w}_i$ and $\vec{x}$ is matched.

2. The second test is $\frac{\vec{w}_i^T \vec{x}}{\|\vec{x}\|^2} > \rho$. Here $\vec{w}_i$ is declared to match $\vec{x}$ if a significate fraction of the ones in $\vec{x}$ appears in $\vec{w}_i$.

If these two test are passed then $\vec{x}$ joins cluster $i$, and the i-th unit weight vector ($\vec{w}_i$) is updated according to: $\vec{w}_{i\text{new}} = \vec{w}_i \ AND \ \vec{x}$.

If the unit $i$ doesn’t pass the first test then $\vec{w}_i$ is declared ”too far” from the input $\vec{x}$ and a new unit $j$ representing the new cluster $j$ is allocated and initialized by $\vec{w}_j = \vec{x}$

If the unit $i$ passes only the first test, we successively pass through all the other prototypes $\vec{w}_k$ in decreasing order of the competition function. We continue until both tests are passed Or until one of the prototypes doesn’t pass the first test (and we are finished). If we run of prototypes, we create a separate unit containing $\vec{x}$ as before.
8.6 Supervised learning

In this sort of learning the required output of the net is known. We can, therefore, directly compute the error of the network, considering the real and the expected output. In the process of training we use a list or training set of correct input-output pairs as examples, during the training we compare the network output to the correct output and change the synapses’ weights $w_{i,j}$ to minimize the difference.

Let us denote $\{\xi_{\mu}\}_{1 \leq \mu \leq p}$ as the set of inputs $\{\zeta_{\mu}\}_{1 \leq \mu \leq p}$ as the corresponding set of correct outputs and $\{O_{\mu}\}_{1 \leq \mu \leq p}$ as the corresponding set of network’s outputs.

In the following sections we will discuss a few methods for supervised learning:

8.6.1 Delta rule (perceptron learning rule)

Usually used with threshold (binary) single layered neural networks where $g(h) = \text{sign}(h)$ (where $h$ is the input to the neuron).

Here, we assume that the targets $\xi_{\mu}$ also take $\pm 1$ values thus all that matters is the sign of the net input $h_{\mu}^{i}$ to output unit $i$. We want this sign to be the same as that of $\zeta_{\mu}$ for each $i$ and $\mu$.

In other words, for an output unit $i$, the desired weight vector $\vec{w}_{i}$ should satisfy:

$$O_{i}^{\mu} = \text{sign}(\vec{w}_{i} \cdot \xi_{\mu}) = \zeta_{\mu}^{i} \text{ or } \vec{w}_{i} \cdot x_{\mu}^{i} > 0 \text{ where } x_{\mu}^{i} = \zeta_{\mu}^{i} \xi_{\mu}^{i}$$

Which means that our goal is to find, for each output unit $i$ a corresponding separating plane $\vec{w}_{i}$ between the two clusters of patterns (those who are mapped to value -1 and those to value 1) if one exists (i.e. if the problem is linear separable).

In order to approximate the appropriate weight vectors (or to find them if the problem is linear separable), in the spirit of Hebb, we add to each connection something proportional to the product of the input and the desired output:

$$\Delta_{i,k} = \begin{cases} 
2 \eta \zeta_{\mu}^{i} \xi_{\mu}^{i} & \text{if } O_{i}^{\mu} \neq \zeta_{i}^{\mu} \\
0 & \text{else}
\end{cases} \text{ where } \eta \text{ is the learning rate constant.}$$

8.6.2 Gradient descent

Assuming activation function is differentiable, in this method we search the weight space in order to find the optimal set of weights with respect to our error measure. The weights vector update is in the direction opposite to the error gradient. The magnitude of update is proportional to the learning rate. It can be proved that with suitable set of learning rates we converge to a local minima.

Using an $SSE$ error measure for the output of neuron $i$,
\[ E[W] = \frac{1}{2} \sum_i (T_i - O_i)^2 = \frac{1}{2} \sum_i (T_i - g(h_i))^2 = \frac{1}{2} \sum_i (T_i - g(\sum_k w_{ik} I_k))^2. \]

Update equation is as following:
\[ \Delta w_{i,k} = -\eta \frac{\partial E}{\partial O_i} \frac{\partial O_i}{\partial w_{ik}} = \eta [T_i - O_i] I_k g'(h_i) \]

Where \( h_i \) is the field \( i \) and \( g \) is the activation function, \( O_i \) is the output of neuron \( i \), \( T_i \) is the required output as provided by supervised learning and \( \eta \) is the learning rate constant.

From this analysis the back-propagation algorithm for any weight \( w_{i,j} \) based on gradient descent for multi-layer network follows directly from the chain rule for derivation.

### 8.6.3 Back propagation

Given a sigmoid-neurons net, Let us denote \( I^u \) as the \( u \)-th input vector, \( V^m \) as the output vector of neurons on layer \( m \), \( h^m \) as the input vector of neurons on layer \( m \) (where \( V_i^m = g(h_i^m) = g(\sum_j w_{i,j}^m V_j^{m-1}) \) and \( w_{i,j}^m \) as the weight of the synapse between neuron \( j \) on layer \( m-1 \) to neuron \( i \) on layer \( m \).

Taking \( SSE \) as our error measure, The update equation will be:
\[ \Delta w_{i,j}^m = \eta \delta_{i,j}^m V_j^{m-1} \]

Where for the output layer \( M \): \( \delta_{i,j}^M = g'(h_i^M)[T_i^u - O_i^u] \)

And where for the other layers: \( \delta_{i,j}^{m-1} = g'(h_i^{m-1}) \sum_j w_{i,j}^m \delta_{j}^m \)

### The algorithm

1. Initialize all weights to small random values.
2. Choose a pattern \( I^u \) and apply it to the input layer:
   \( V_k^0 = I_k^u \) for all \( k \).
3. Propagate the signal forward through the network:
   \( V_i^m = g(h_i^m) = g(\sum_j w_{i,j}^m V_j^{m-1}) \) for each neuron \( i \) and layer \( m \).
4. Compute the deltas for the output layer:
   \( \delta_{i}^M = g'(h_i^M)[T_i^u - V_i^M] \)
5. Compute the deltas for the preceding layers by propagating the error backwards:
   \( \delta_{i,j}^{m-1} = g'(h_i^{m-1}) \sum_j w_{i,j}^m \delta_{j}^{m-1} \) for \( m = M, M-1, \ldots, 2 \) and for each \( i \)
6. Update all weights:
   \( \Delta w_{i,j}^m = \eta \delta_{i,j}^m V_j^{m-1} \)
7. Go back to step 2 (repeat for the next input pattern $I^{a+1}$)

### 8.6.4 RBF (Radial Basis Function)

In this model, we use networks with one hidden layer.

**Fig.4 Schematic drawing of an RBF models’ network**

- The output computation for neurons is defined as follows:
  \[ y_l(\vec{x}) = \sum_{j=1}^{J} W_{l,j} Z_j(\vec{x}) \quad \text{where} \quad Z_j(\vec{x}) = \exp\left(-\frac{\|\vec{x}-\mu_j\|^2}{2\sigma_j^2}\right) \]

  \textbf{Explanation:}
  \(\vec{x}\) is the input vector, \(W_{l,j}\) is the weight of the synapse between hidden layer neuron \(j\) and output neuron \(l\), \(Z_j\) is the output of hidden layer neuron \(j\), \(\mu_j\) represents the receptive field center of hidden layer unit \(j\), \(\sigma_j\) is the std deviation of \(j\)-th hidden unit’s receptive field.

- The error estimation for output unit \(l\) is defined as follows:
  \[ E_l(t) = (y_l(t) - Y_t^l)^2 = (\sum_{j=1}^{J} W_{l,j}(t) Z_j - Y_t^l)^2 \]

  \textbf{Explanation:}
  \(Y_t^l\) is the correct output on stage \(t\), \(y_l(t)\) is the network’s output.

- The learning rule for weights update for RBF is derived from gradient descent analysis:
  \[ \frac{\partial E_l(t)}{\partial w_{l,j}} = 2(y_l(t) - Y_t^l) Z_j \]
  and defined as follows:
  \[ w_{l,j}(t + 1) = w_{l,j}(t) - \rho \frac{\partial E_l(t)}{\partial w_{l,j}} \]

  \textbf{Note: only the synaptic weights of output neurons are modified.}

Intuitively, the receptive fields represent proper clusters of data, thus the architecture can be employed for unsupervised learning and clustering as well.
8.7 Model selection

Given specific net architecture we should carefully select the number of neurons and synapses. The complexity of the model should be powerful enough to capture the objective function, but should be limited to avoid a state of overfitting. If we chose a net with a too large number of free dimensions (too much synapses), it can result in just absorbing the training set with poor results over the test set. For proper network selection the classical statistical methods, for example cross-validation, can be used. A very common technique is weight penalty integrated into the error function of supervised learning.

8.8 Summary

- Artificial neural networks are inspired by the learning processes that take place in biological systems.
- Artificial neurons and neural networks try to imitate the working mechanisms of their biological counterparts.
- Learning can be perceived as an optimization process.
- Biological neural learning happens by the modification of the synaptic strength. Artificial neural networks learn in the same way.
- The synapse strength modification rules for artificial neural networks can be derived by applying mathematical optimization methods.

8.8.1 Some examples for Real world applications

- Pattern recognition
- Industrial inspection
- Fault diagnosis
- Image recognition
- Target recognition
- Speech recognition
- Natural language processing
- Handwriting recognition
8.9 References

- *Prof. Victor Brailovski* lecture notes
- *Mr. Alon Keinan* lecture presentations
- *Mohamad Hassoun* "Fundamentals of ANN"
- *Hertz Krog and Palmer* "Introduction to the theory of neural computation"