

Artificial Neural Networks

- 1 Artificial Neural Networks
 - Properties
 - Applications
 - Classical Examples
 - Biological Background
- 2 Single Layer Networks
 - Limitations
 - Training Single Layer Networks
- 3 Multi Layer Networks
 - Possible Mappings
 - Backprop algorithmen
 - Practical Problems
- 4 Generalization

1 Artificial Neural Networks

- Properties
- Applications
- Classical Examples
- Biological Background

2 Single Layer Networks

- Limitations
- Training Single Layer Networks

3 Multi Layer Networks

- Possible Mappings
- Backprop algorithmen
- Practical Problems

4 Generalization

Artificial Neural Networks (ANN)

- Inspired from the nervous system
- Parallel processing

Artificial Neural Networks (ANN)

- Inspired from the nervous system
- Parallel processing

We will focus on **one** class of ANNs:

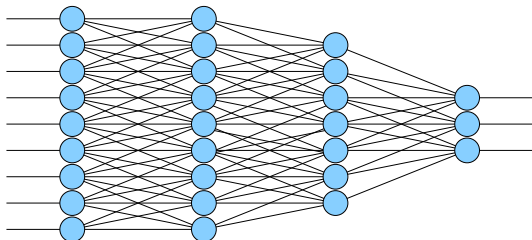
Feed-forward Layered Networks

Artificial Neural Networks (ANN)

- Inspired from the nervous system
- Parallel processing

We will focus on **one** class of ANNs:

Feed-forward Layered Networks



Applications

Operates like a general "Learning Box"!

Applications

Operates like a general "Learning Box"!

Classification



Applications

Operates like a general "Learning Box"!

Function Approximation



Applications

Operates like a general "Learning Box"!

Multidimensional Mapping



Classical Examples

Classical Examples

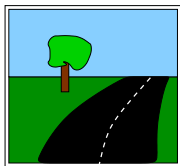
ALVINN

Autonomous driving

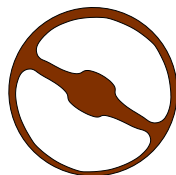
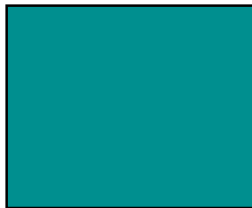
Classical Examples

ALVINN

Autonomous driving



Video image

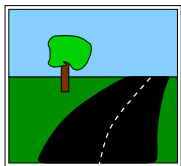


Steering

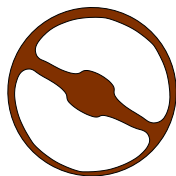
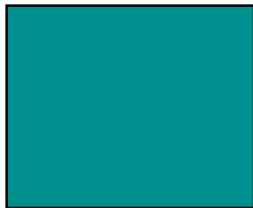
Classical Examples

ALVINN

Autonomous driving



Video image



Steering

Trained to mimic the behavior of human drivers

Classical Examples

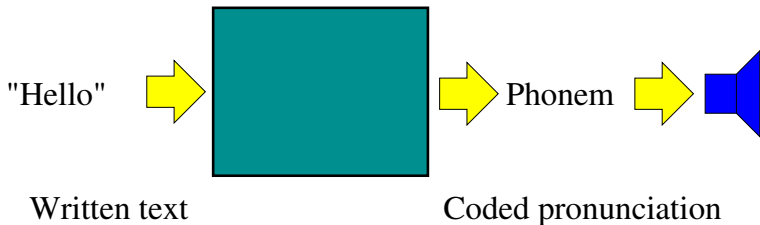
NetTalk

Speech Synthesis

Classical Examples

NetTalk

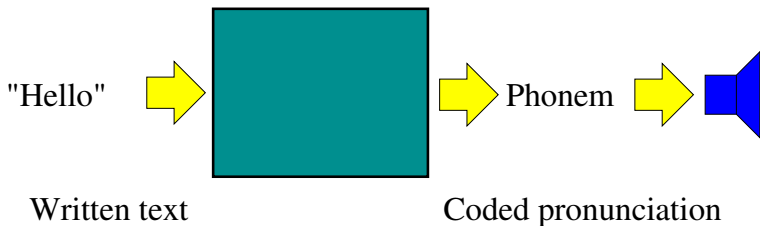
Speech Synthesis



Classical Examples

NetTalk

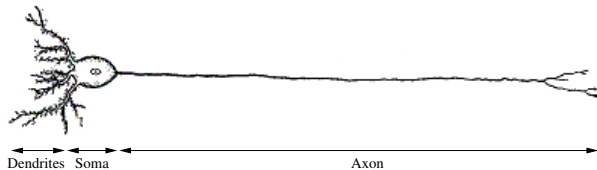
Speech Synthesis



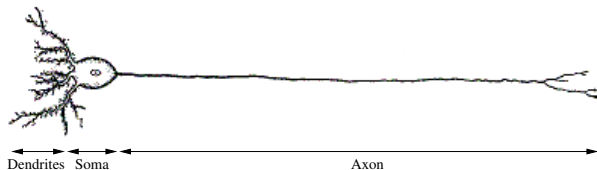
Trained using a large database of spoken text

How do real neurons (nerve cells) work?

How do real neurons (nerve cells) work?

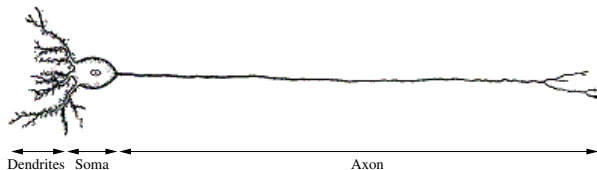


How do real neurons (nerve cells) work?



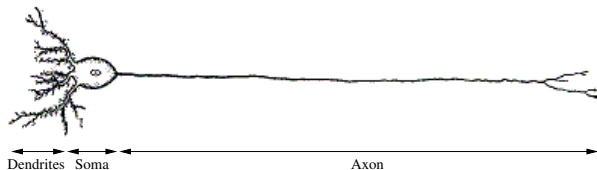
- Dendrites
- Soma (Cell Body)
- Axon

How do real neurons (nerve cells) work?



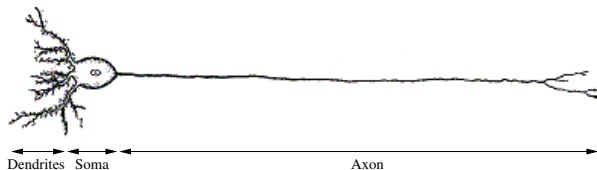
- Dendrites
Passive reception of (chemical) signals
- Soma (Cell Body)
- Axon

How do real neurons (nerve cells) work?



- Dendrites
Passive reception of (chemical) signals
- Soma (Cell Body)
Summing, Thresholding
- Axon

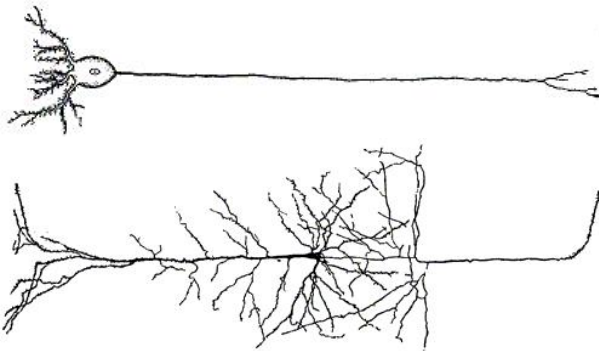
How do real neurons (nerve cells) work?



- **Dendrites**
Passive reception of (chemical) signals
- **Soma (Cell Body)**
Summing, Thresholding
- **Axon**
Active pulses are transmitted to other cells

Nerve cells can vary in shape and other properties

Nerve cells can vary in shape and other properties

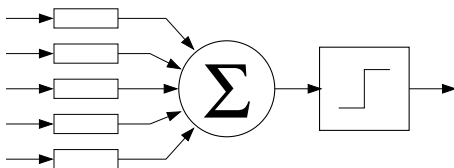


ANN-caricatures

(simplified view of the neural information processing)

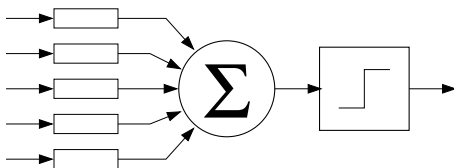
ANN-caricatures

(simplified view of the neural information processing)



ANN-caricatures

(simplified view of the neural information processing)

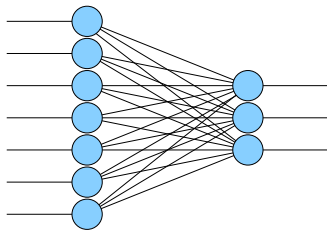


- Weighted input signals
- Summing
- Thresholded output

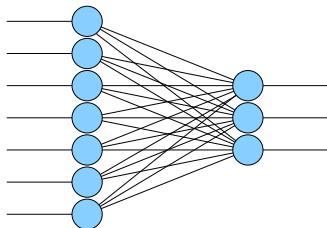
- 1 Artificial Neural Networks
 - Properties
 - Applications
 - Classical Examples
 - Biological Background
- 2 Single Layer Networks
 - Limitations
 - Training Single Layer Networks
- 3 Multi Layer Networks
 - Possible Mappings
 - Backprop algorithmen
 - Practical Problems
- 4 Generalization

What do we mean by a *Single Layer Network*?

What do we mean by a *Single Layer Network*?

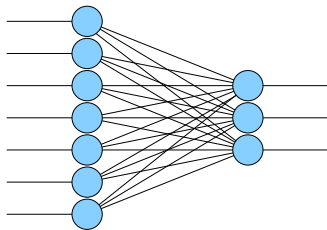


What do we mean by a *Single Layer Network*?



Each cell operates independently of the others!

What do we mean by a *Single Layer Network*?

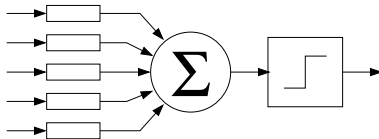


Each cell operates independently of the others!

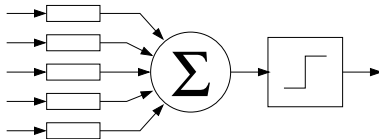
It is sufficient to understand what **one** cell can compute

What can a single "cell" compute?

What can a single "cell" compute?

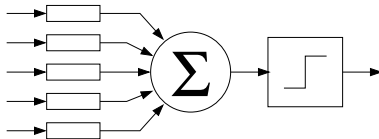


What can a single "cell" compute?



- \vec{x} Input in vector format
- \vec{w} Weights in vector format
- Output

What can a single "cell" compute?



- \vec{x} Input in vector format
- \vec{w} Weights in vector format
- o Output

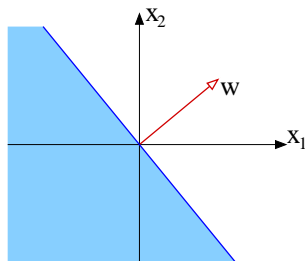
$$o = \text{sign} \left(\sum_i x_i w_i \right)$$

$$o = \text{sign} \left(\sum_i x_i w_i \right)$$

Geometrical interpretation

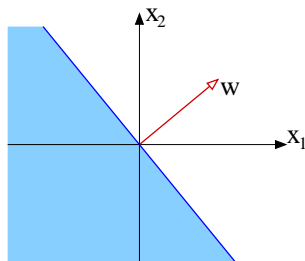
$$o = \text{sign} \left(\sum_i x_i w_i \right)$$

Geometrical interpretation



$$o = \text{sign} \left(\sum_i x_i w_i \right)$$

Geometrical interpretation



Separating hyper plane
Linear separability

Learning in ANNs

What does learning mean here?

Learning in ANNs

What does learning mean here?

The network structure is normally fixed

Learning in ANNs

What does learning mean here?

The network structure is normally fixed

Learning means finding the **best weights** w_i

Learning in ANNs

What does learning mean here?

The network structure is normally fixed

Learning means finding the **best weights** w_i

Two good algorithms for single layer networks:

- Perceptron Learning
- Delta Rule

Perceptron Learning

Perceptron Learning

- Incremental learning

Perceptron Learning

- Incremental learning
- Weights only change when the output is wrong

Perceptron Learning

- Incremental learning
- Weights only change when the output is wrong
- Update rule: $w_i \leftarrow w_i + \eta(t - o)x_i$

Perceptron Learning

- Incremental learning
- Weights only change when the output is wrong
- Update rule: $w_i \leftarrow w_i + \eta(t - o)x_i$
- Always converges if the problem is solvable

Delta Rule (LMS-rule)

Delta Rule (LMS-rule)

- Incremental learning

Delta Rule (LMS-rule)

- Incremental learning
- Weights always change

Delta Rule (LMS-rule)

- Incremental learning
- Weights always change
- $w_i \leftarrow w_i + \eta(t - \vec{w}^T \vec{x})x_i$

Delta Rule (LMS-rule)

- Incremental learning
- Weights always change
- $w_i \leftarrow w_i + \eta(t - \vec{w}^T \vec{x})x_i$
- Converges only in the mean

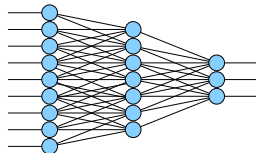
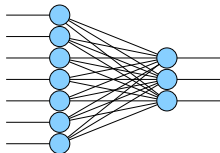
Delta Rule (LMS-rule)

- Incremental learning
- Weights always change
- $w_i \leftarrow w_i + \eta(t - \vec{w}^T \vec{x})x_i$
- Converges only in the mean
- Will find an optimal solution even if the problem can not be fully solved

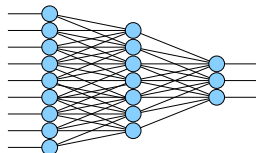
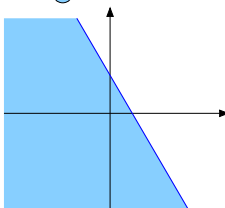
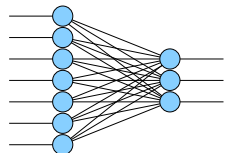
- 1 Artificial Neural Networks
 - Properties
 - Applications
 - Classical Examples
 - Biological Background
- 2 Single Layer Networks
 - Limitations
 - Training Single Layer Networks
- 3 Multi Layer Networks
 - Possible Mappings
 - Backprop algorithmen
 - Practical Problems
- 4 Generalization

What is the point of having multiple layers?

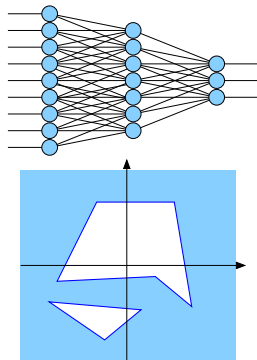
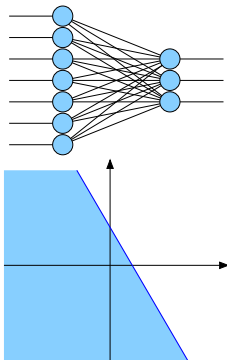
What is the point of having multiple layers?



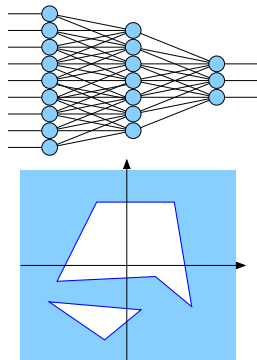
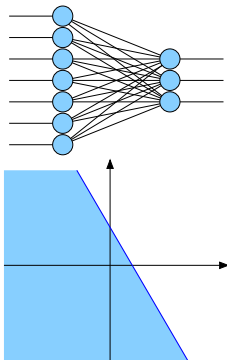
What is the point of having multiple layers?



What is the point of having multiple layers?

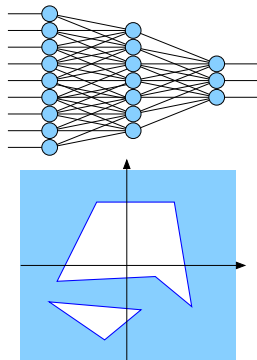
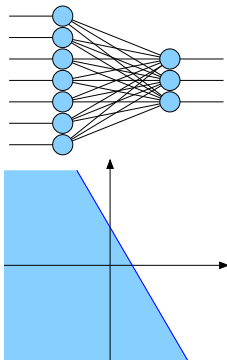


What is the point of having multiple layers?



A two layer network can implement **arbitrary decision surfaces**

What is the point of having multiple layers?



A two layer network can implement **arbitrary decision surfaces**
...provided we have *enough hidden units*

Will it be even better with more layers?

Will it be even better with more layers?

- Two layers can describe **any** classification
- Two layers can approximate **any** "continuous" function

Will it be even better with more layers?

- Two layers can describe **any** classification
- Two layers can approximate **any** "continuous" function
- Three layers can sometimes do the same thing more efficiently

Will it be even better with more layers?

- Two layers can describe **any** classification
- Two layers can approximate **any** "continuous" function
- Three layers can sometimes do the same thing more efficiently
- More than three layers are rarely used

How can we train a multi layer network?

How can we train a multi layer network?

Neither perceptron learning, nor the delta rule can be used

How can we train a multi layer network?

Neither perceptron learning, nor the delta rule can be used

Fundamental problem:

When the network gives the wrong answer
there is no information on in which direction
the weights need to change to improve the result

How can we train a multi layer network?

Neither perceptron learning, nor the delta rule can be used

Fundamental problem:

When the network gives the wrong answer
there is no information on in which direction
the weights need to change to improve the result

Fundamental trick:

Use threshold-like, but continuous functions

How can we train a multi layer network?

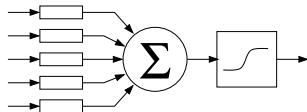
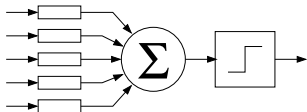
Neither perceptron learning, nor the delta rule can be used

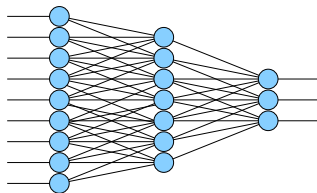
Fundamental problem:

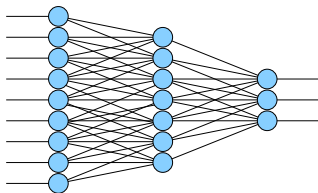
When the network gives the wrong answer
there is no information on in which direction
the weights need to change to improve the result

Fundamental trick:

Use threshold-like, but continuous functions

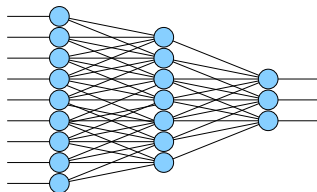






Basic idée:

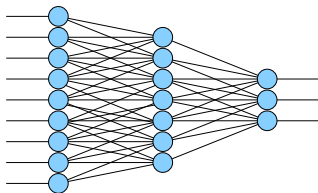
Minimize the error (E) as a function of **all** weights (\vec{w})



Basic idée:

Minimize the error (E) as a function of **all** weights (\vec{w})

- 1 Compute the direction in weight space where the error increases the most $\text{grad}_{\vec{w}}(E)$



Basic idée:

Minimize the error (E) as a function of **all** weights (\vec{w})

- 1 Compute the direction in weight space where the error increases the most $\text{grad}_{\vec{w}}(E)$
- 2 Change the weights in the opposite direction

$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

Normally one can use the error from each example separately

Normally one can use the error from each example separately

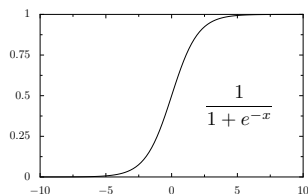
$$E = \frac{1}{2} \sum_{k \in \text{Out}} (t_k - o_k)^2$$

Normally one can use the error from each example separately

$$E = \frac{1}{2} \sum_{k \in \text{Out}} (t_k - o_k)^2$$

A common "threshold-like function" is

$$\rho(y) = \frac{1}{1 + e^{-y}}$$



The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j$$

The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j \quad w_{ji} \leftarrow w_{ji} + \eta \delta_i x_j$$

The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j \quad w_{ji} \leftarrow w_{ji} + \eta \delta_i x_j$$

Output layer:

$$\delta_k = o_k \cdot (1 - o_k) \cdot (t_k - o_k)$$

The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j \quad w_{ji} \leftarrow w_{ji} + \eta \delta_i x_j$$

Output layer:

$$\delta_k = o_k \cdot (1 - o_k) \cdot (t_k - o_k)$$

Hidden layers:

$$\delta_h = o_h \cdot (1 - o_h) \cdot \sum_{k \in \text{Out}} w_{kh} \delta_k$$

The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j \quad w_{ji} \leftarrow w_{ji} + \eta \delta_i x_j$$

Output layer:

$$\delta_k = o_k \cdot (1 - o_k) \cdot (t_k - o_k)$$

Hidden layers:

$$\delta_h = o_h \cdot (1 - o_h) \cdot \sum_{k \in \text{Out}} w_{kh} \delta_k$$

The error δ propagates backwards through the layers
Error backpropagation (BackProp)

Things to think about then using BackProp

Things to think about then using BackProp

- Sloooow
Normal to require thousands of iterations through the dataset

Things to think about then using BackProp

- Sloooow
Normal to require thousands of iterations through the dataset
- Gradient following
Risk of getting stuck in local minima

Things to think about then using BackProp

- Sloooow
Normal to require thousands of iterations through the dataset
- Gradient following
Risk of getting stuck in local minima
- Many parameters
 - Step size η
 - Number of layers
 - Number of hidden units
 - Input and output representation
 - Initial weights

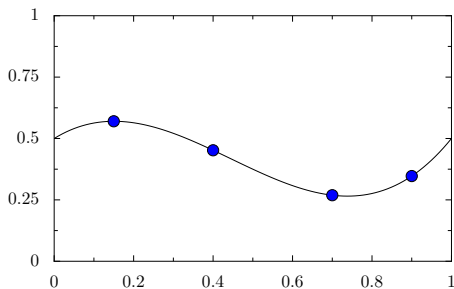
- 1 Artificial Neural Networks
 - Properties
 - Applications
 - Classical Examples
 - Biological Background
- 2 Single Layer Networks
 - Limitations
 - Training Single Layer Networks
- 3 Multi Layer Networks
 - Possible Mappings
 - Backprop algorithmen
 - Practical Problems
- 4 Generalization

Generalization

The net normally *interpolates* smoothly between the data points

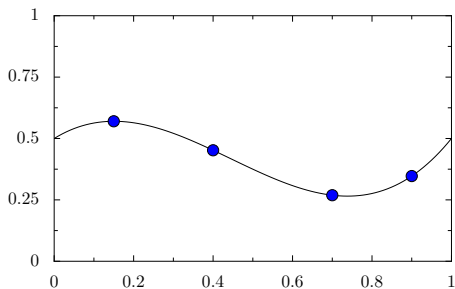
Generalization

The net normally *interpolates* smoothly between the data points



Generalization

The net normally *interpolates* smoothly between the data points



Results in good generalization

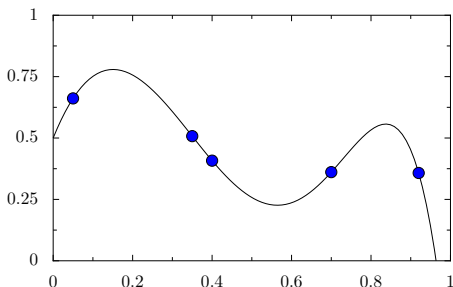
Risk for overfitting (överinlärning)!

Risk for overfitting (**överinlärning**)!

If the network has too many degrees of freedom (weights), the risk increases that learning will find a "strange" solution

Risk for overfitting (överinlärning)!

If the network has too many degrees of freedom (weights), the risk increases that learning will find a "strange" solution



Limiting the number of hidden units tends to improve generalization

Limiting the number of hidden units tends to improve generalization

