Artificial Neural Networks

Pawel Herman

Department of Computational Biology

School of Computer Science and Communication

KTH Royal Institute of Technology

BEST summer school

Stockholm, June 9-13 2014

Introduction to ANNs - origins

- Information processing paradigm inspired by the brain
- A structure of highly interconnected processing units working together
- Knowledge is acquired from data by a learning process, it is represented in a distributed fashion over the connections
- ANNs are massively parallel and fault tolerant
- They are capable of finding structure from complex and imprecise data
- The first model of an artificial neuron by McCulloch and Pitts (1943)

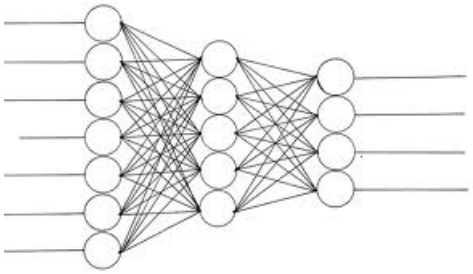
> *"Neural networks do not perform miracles. But if used sensibly they can produce some amazing results."*

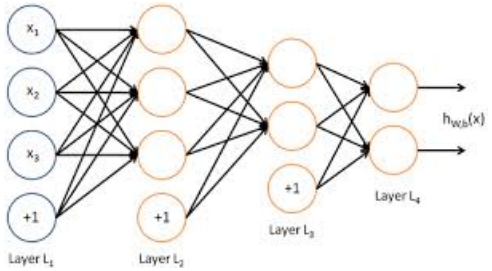
A wide range of problems addressed with ANNs

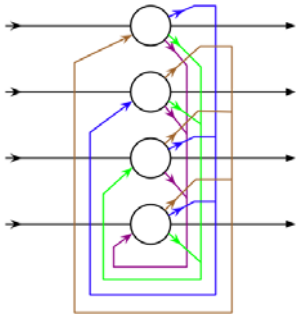
- Classification and pattern recognition
- Regression (interpolation)
- Input-output mapping (association)
- Clustering
- Data coding
- Signal processing, filtering
- System identification
- etc.....

Basic components

- Topology, network architecture, connectivity
- Nodes and their activation function
- Learning rule for weights that characterise connections
- Data
  - corresponding ANN input and assumption about its nature
  - corresponding ANN output

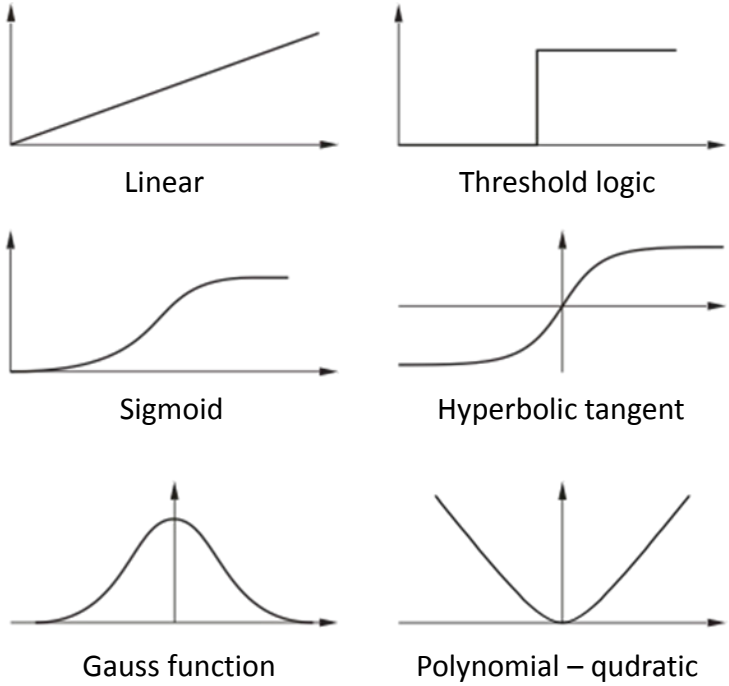






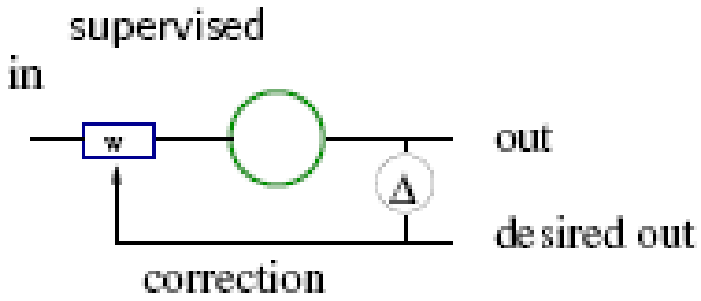
Basic components

- Topology, network architecture, connectivity
- Nodes and their activation function
- Learning rule for weights that characterise connections
- Data
  - corresponding ANN input and assumption about its nature
  - corresponding ANN output



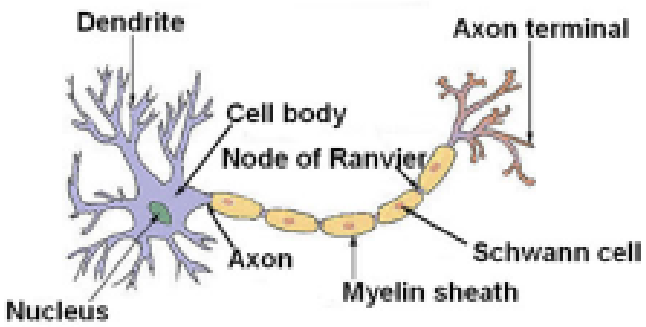
Basic components

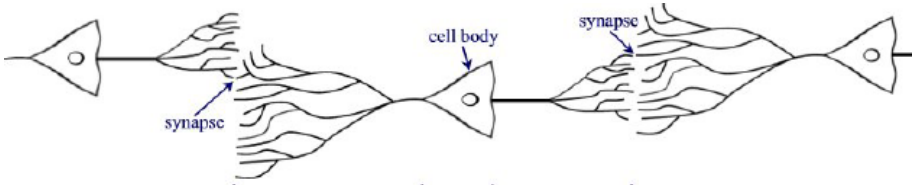
- Topology, network architecture, connectivity
- Nodes and their activation function
- Learning rule for weights that characterise connections
- Data
  - corresponding ANN input and assumption about its nature
  - corresponding ANN output

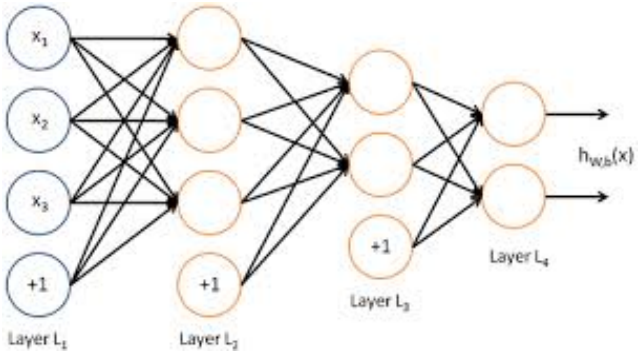


E. Fransen

Biological inspiration

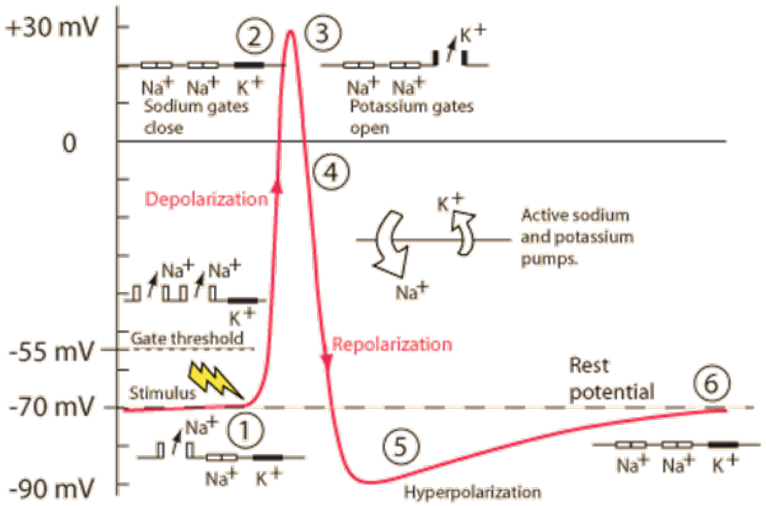


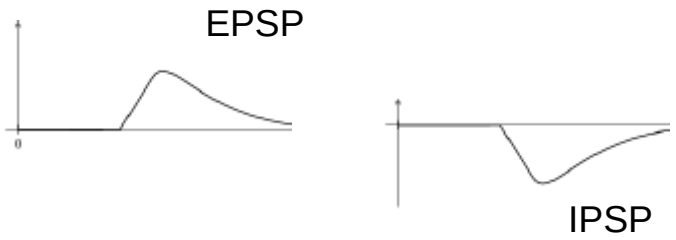




Communication via action potentials

- complex biophysical machinery reposnsible for generation of action potentials (nerve impulses called spikes)
- spikes travel along the axon
- synapses translate spikes into analog postsynaptic potentials (PSPs)
  - spikes release neurotransmitters
  - transmitters activate receptors
  - receptors open ion channels
  - channels produce PSP (excitatory or inhibitory)
  - learning affects this cascade of events (parameters)





E. Fransen

Biological neurons

- all-or-nothing logic of spikes
- multiple inputs and ouputs for each neuron cell
- analog transmission in synapses with the weight in connection
- summation (integration) of postsynaptic potentials in cell body
- if the potential becomes lower than threshold (sufficient depolarisation), spike is generated
- transfer function how this summed input contributes to spike generation (relation between the sum and firing frequency)

Processing in biological and artificial neural networks

- information processing based on local information
- learning affects weights, which effectively account for memories (they store memories)
- weights are undergo changes depending on learning rules
- weights can be positive (excitatory effect) and negative (inhibitory effect)
- although parallel processing in networks is fast, learning takes place on longer time scales.
- there is inherent tolerance to errors in inputs, weights and outputs

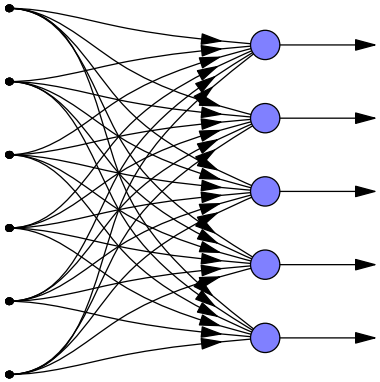
No free lunch

"… any two optimisation algorithms are equivalent when their performance is averaged across all possible problems"

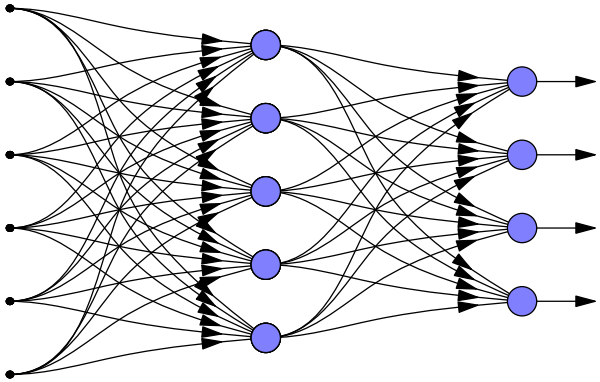
Wolpart and Macready

Connection Topologies

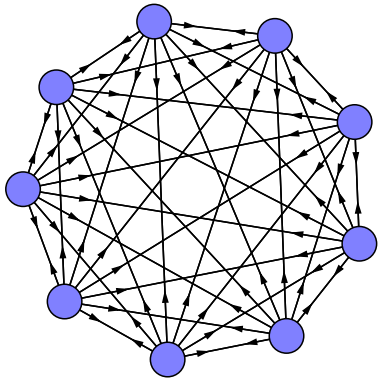
Single Layer



Multi Layer

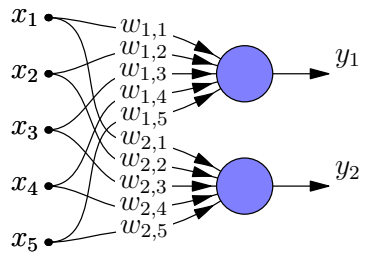


Recurrent



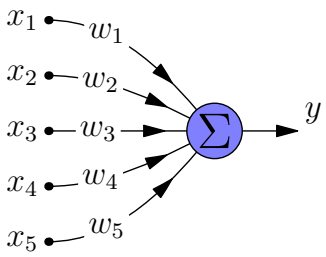
Linear Feed-Forward Networks

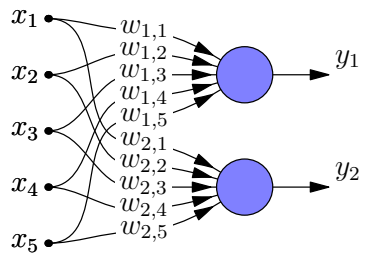
What can be computed by a linear network?



Linear Feed-Forward Networks

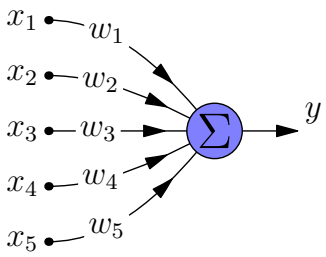
What can be computed by a linear network?

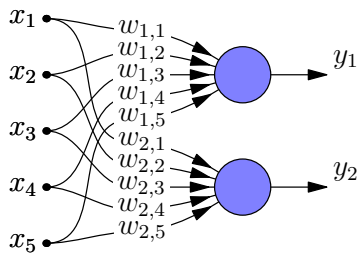




Linear Feed-Forward Networks

What can be computed by a linear network?

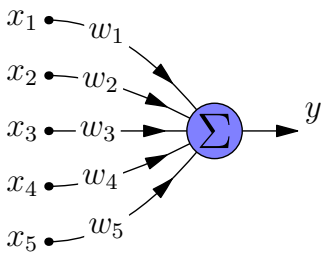


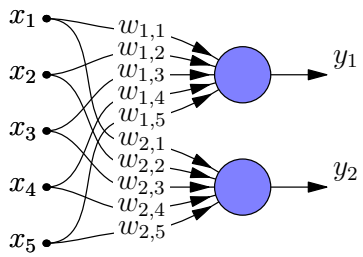


$$y = \vec{w}^T \cdot \vec{x}$$

Linear Feed-Forward Networks

What can be computed by a linear network?



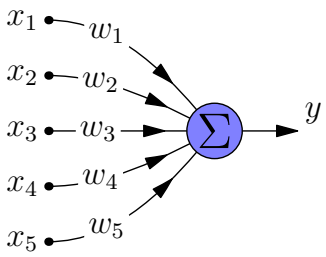


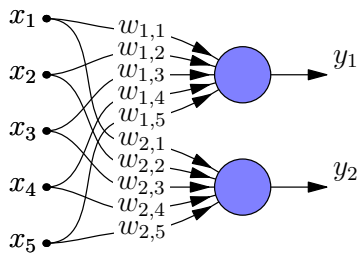
$$y = \vec{w}^T \cdot \vec{x}$$

$\vec{w}$ — Weight Vector

Linear Feed-Forward Networks

What can be computed by a linear network?





$$y = \vec{w}^T \cdot \vec{x}$$

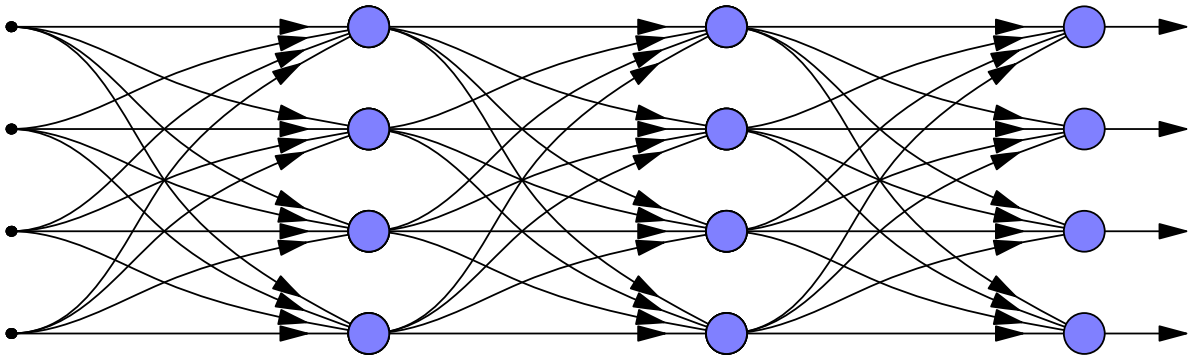
$$\vec{y} = W\vec{x}$$

$\vec{w}$ — Weight Vector

W — Weight Matrix

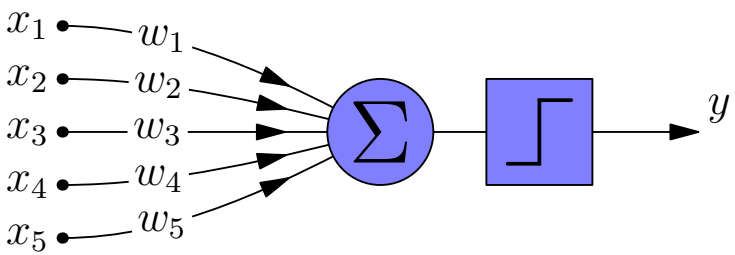
Linear Feed-Forward Networks

What happens if we concatenate several linear networks?



Thresholded Neurons

TLU — Threshold Logic Unit

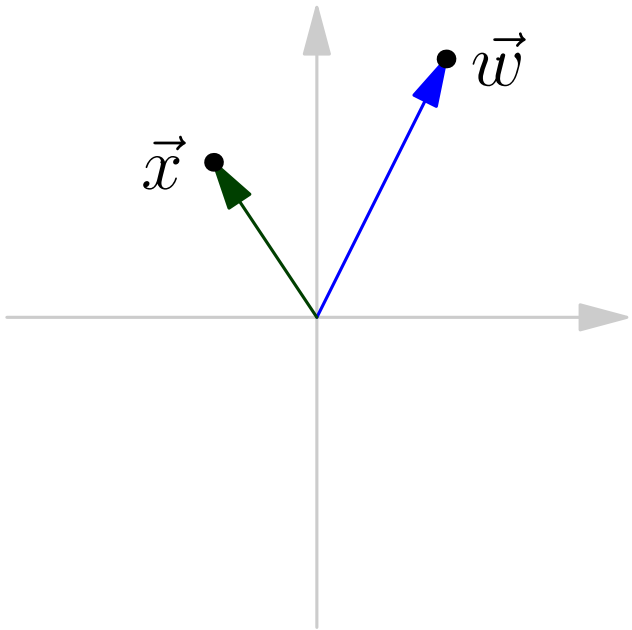


$$y = \begin{cases} 1 & \text{when } \sum_i w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

- Binary output
- Classifies input patterns

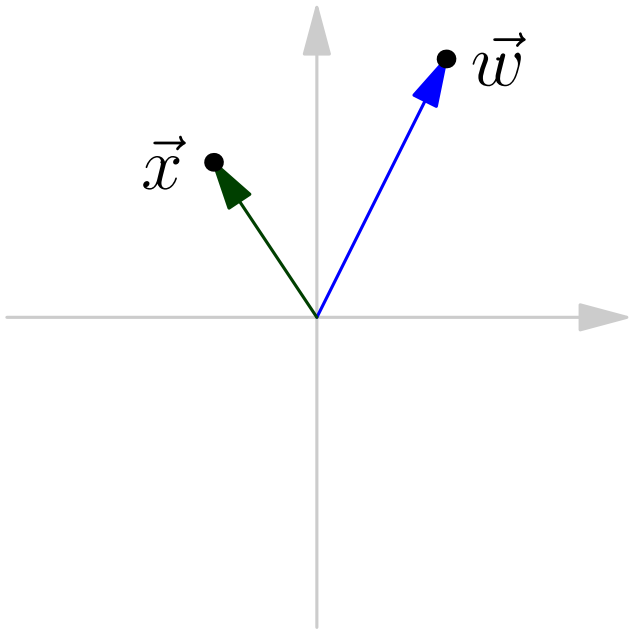
Linear Separation

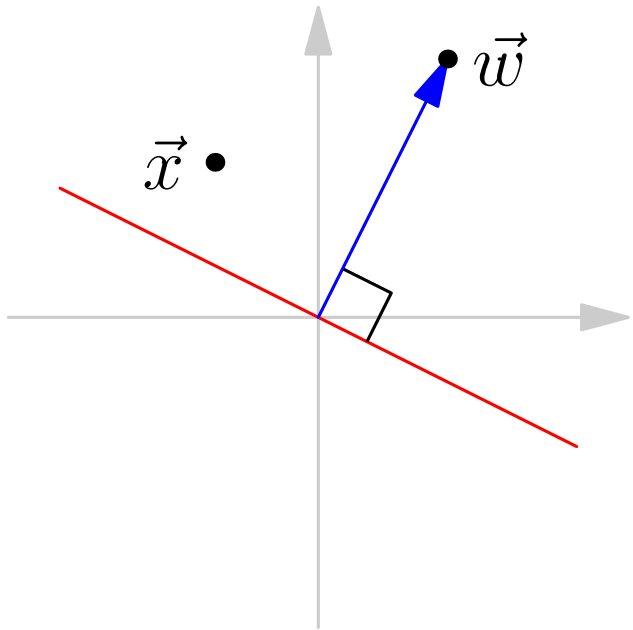
How does the classification work?



Linear Separation

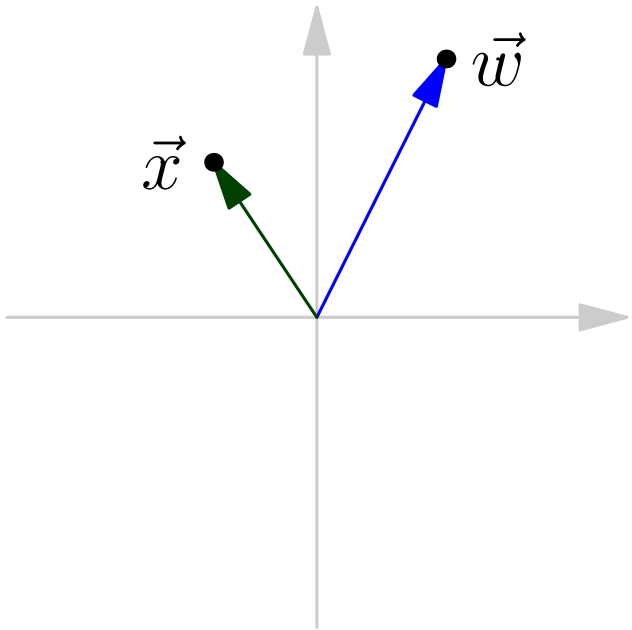
How does the classification work?

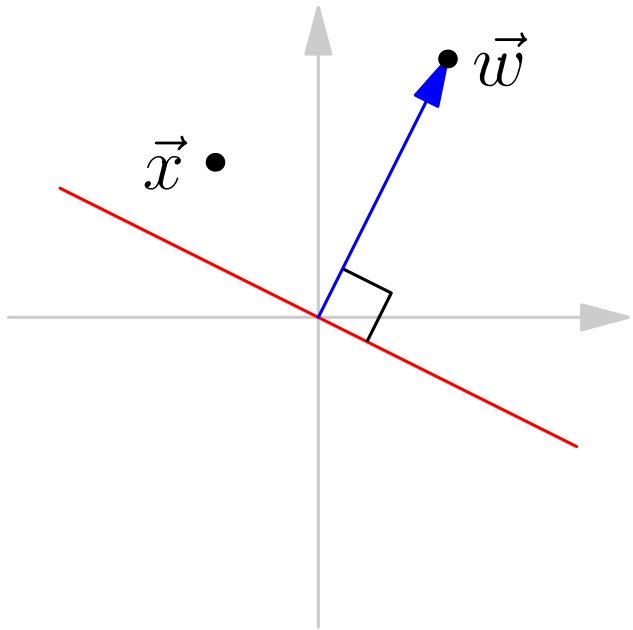




Linear Separation

How does the classification work?

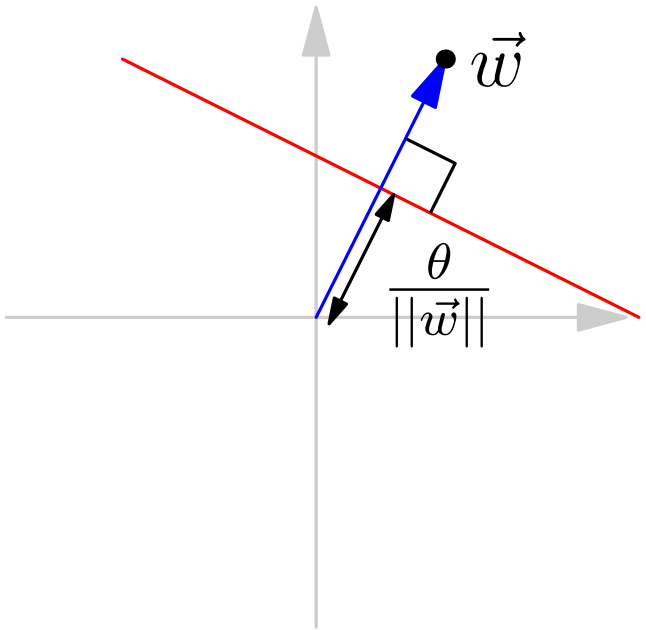




Linear Separation

Linear Separation

Regulation of the threshold θ



$$y = \begin{cases} 1 & \text{when } \sum_i w_i x_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

The separating hyper-plane can be arbitrarily positioned

Linear Separation

Important trick: The variable threshold can be substituted by an extra weight from a constant input

$$\sum_i w_i x_i > \theta$$

$$w_0 \cdot 1 + \sum_i w_i x_i > 0 \qquad w_0 = -\theta$$

Linear Separation

Important trick: The variable threshold can be substituted by an extra weight from a constant input

$$\sum_i w_i x_i > \theta$$

$$w_0 \cdot 1 + \sum_i w_i x_i > 0 \qquad w_0 = -\theta$$

Why?

Linear Separation

Important trick: The variable threshold can be substituted by an extra weight from a constant input

$$\sum_i w_i x_i > \theta$$

$$w_0 \cdot 1 + \sum_i w_i x_i > 0 \qquad w_0 = -\theta$$

Why? Regulation of the threshold does not have to be treated as a special case

Perceptron Learning

Training of a Thresholded Network: Perceptron Learning

Perceptron Learning

Training of a Thresholded Network: Perceptron Learning
Basic Principle: Weights are changed whenever a pattern is erroneously classified

Perceptron Learning

Training of a Thresholded Network: Perceptron Learning
Basic Principle: Weights are changed whenever a pattern is erroneously classified

When the result $= 0$, should be $= 1$

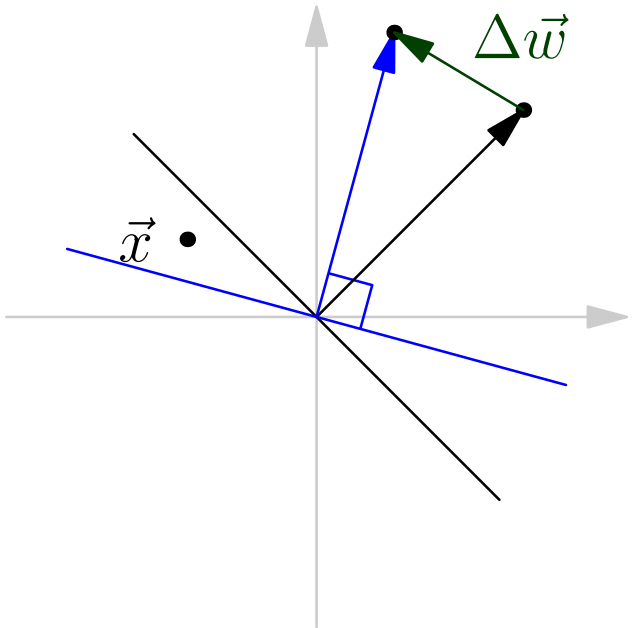
$$\Delta\vec{w} = \eta\vec{x}$$

When the result $= 1$, should be $= 0$

$$\Delta\vec{w} = -\eta\vec{x}$$

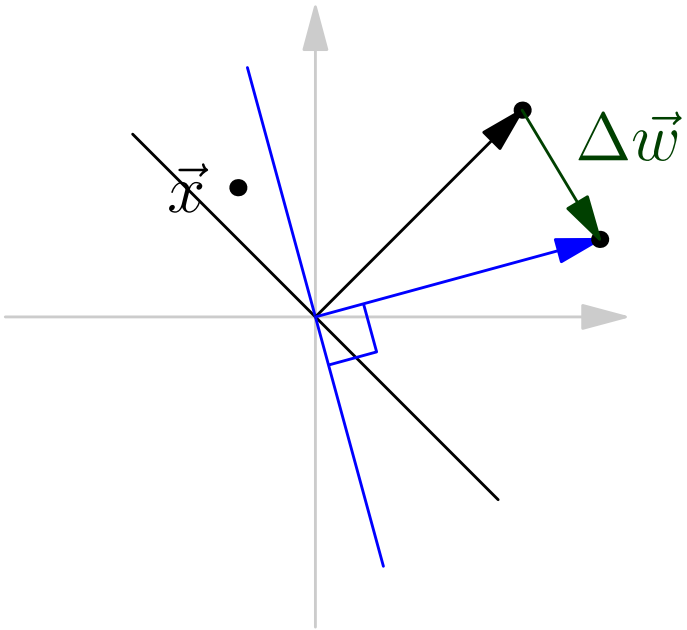
Perceptron Learning

When the result $= 0$, should be $= 1$: $\Delta\vec{w} = \eta\vec{x}$



Perceptron Learning

When the result $= 1$, should be $= 0$: $\Delta\vec{w} = -\eta\vec{x}$



Perceptron Learning

Convergence Theorem

If a solution exists for a finite training dataset then perceptron learning always converges after a finite number of steps

Perceptron Learning

Convergence Theorem

If a solution exists for a finite training dataset then perceptron learning always converges after a finite number of steps

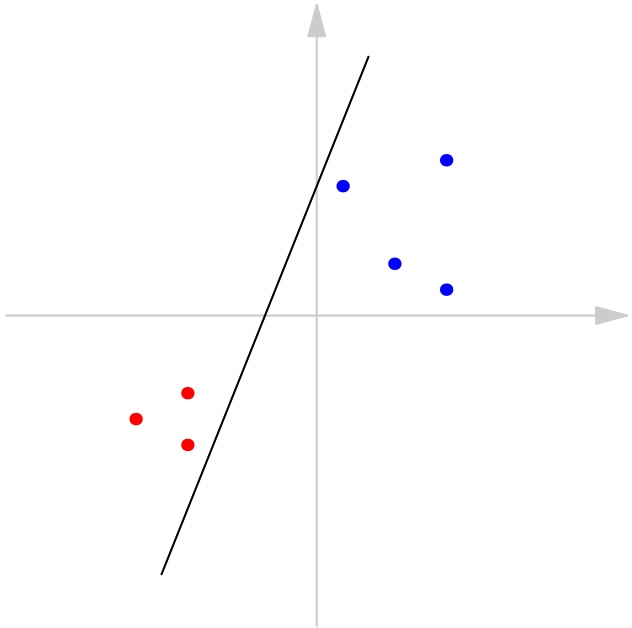
Independent of step size (η)

Perceptron Learning

Problem: Learning terminates unnecessarily early

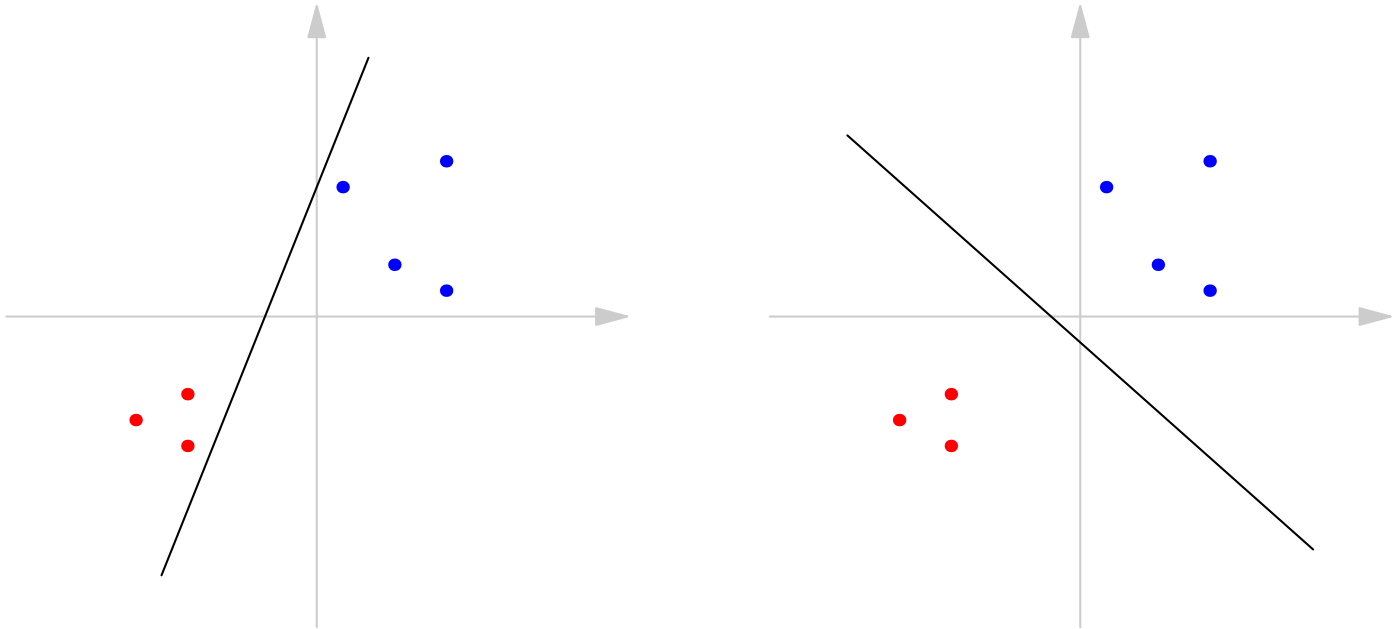
Perceptron Learning

Problem: Learning terminates unnecessarily early



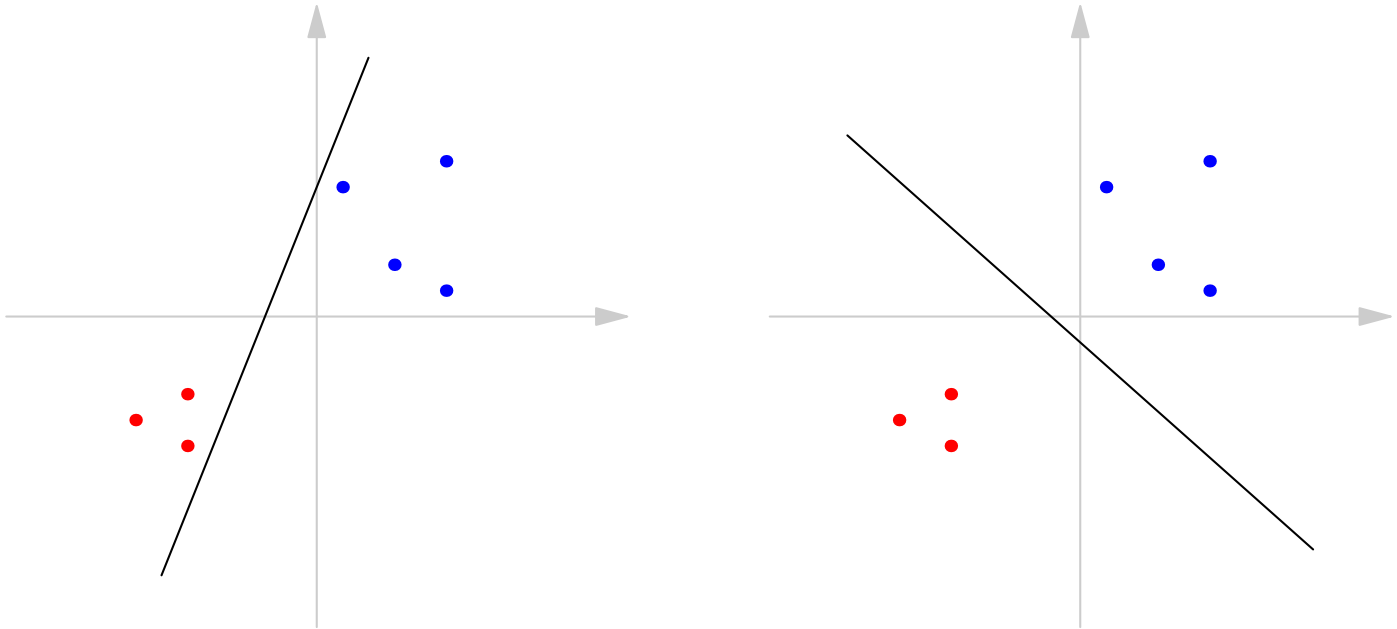
Perceptron Learning

Problem: Learning terminates unnecessarily early



Perceptron Learning

Problem: Learning terminates unnecessarily early



Bad when patterns are only approximately similar to those used during training

Delta Rule

Delta-rule (Widrow-Hoff rule)

Delta Rule

Delta-rule (Widrow-Hoff rule)
Use symmetric target values $\{-1, 1\}$

Delta Rule

Delta-rule (Widrow-Hoff rule)
Use symmetric target values $\{-1, 1\}$
Measure the error before thresholding

$$e = t - \vec{w}^T \vec{x}$$

Delta Rule

Delta-rule (Widrow-Hoff rule)
Use symmetric target values $\{-1, 1\}$
Measure the error before thresholding

$$e = t - \vec{w}^T \vec{x}$$

Find the weights which minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction
Gradient direction:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}}$$

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction
Gradient direction:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = e \frac{\partial e}{\partial \vec{w}}$$

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction
Gradient direction:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = e\frac{\partial e}{\partial \vec{w}} = e\frac{\partial(t - \vec{w}^T\vec{x})}{\partial \vec{w}}$$

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction
Gradient direction:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = e\frac{\partial e}{\partial \vec{w}} = e\frac{\partial(t - \vec{w}^T\vec{x})}{\partial \vec{w}} = -e\vec{x}$$

Delta Rule

Minimize the cost-function

$$\mathcal{E} = \frac{e^2}{2}$$

Simple algorithm: Steepest Decent
Gradient = direction in which the error increases most
Steepest Decent $\Rightarrow$ Move in the opposite direction
Gradient direction:

$$\frac{\partial \mathcal{E}}{\partial \vec{w}} = e\frac{\partial e}{\partial \vec{w}} = e\frac{\partial (t - \vec{w}^T \vec{x})}{\partial \vec{w}} = -e\vec{x}$$

Delta Rule:

$$\Delta \vec{w} = \eta e \vec{x}$$

Training of Thresholded Single-Layer Networks

- Perceptron Learning

- Delta Rule

Training of Thresholded Single-Layer Networks

- Perceptron Learning

$$\Delta\vec{w} = \eta e\vec{x} \qquad \text{where } e = t - y$$

- Delta Rule

Training of Thresholded Single-Layer Networks

- Perceptron Learning

$$\Delta\vec{w} = \eta e\vec{x} \qquad \text{where } e = t - y$$

- Delta Rule

$$\Delta\vec{w} = \eta e\vec{x} \qquad \text{where } e = t - \vec{w}^T\vec{x}$$

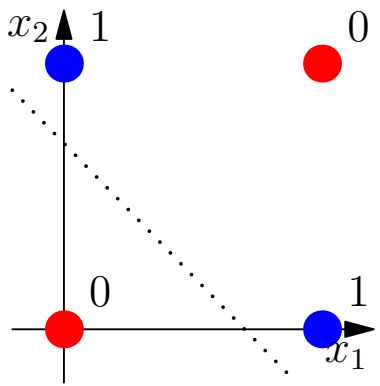
Linear Separation

Can all groups of patterns be separated?

Linear Separation

Can all groups of patterns be separated?
Classical counter-example: Exclusive OR (XOR)

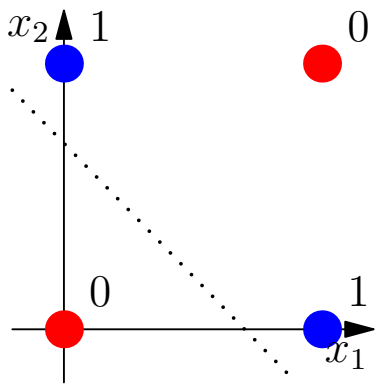
$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow 0 \qquad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow 1 \qquad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow 1 \qquad \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow 0$$



Linear Separation

Can all groups of patterns be separated?
Classical counter-example: Exclusive OR (XOR)

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} \rightarrow 0 \qquad \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow 1 \qquad \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow 1 \qquad \begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow 0$$



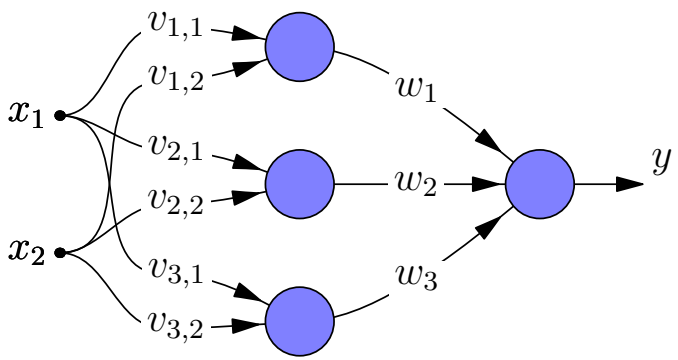
Not linearly separable!

Multi Layer Feed-Forward Networks

What can a thresholded two layer network compute?

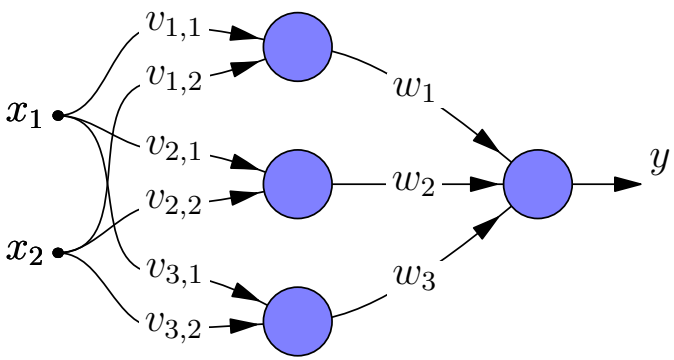
Multi Layer Feed-Forward Networks

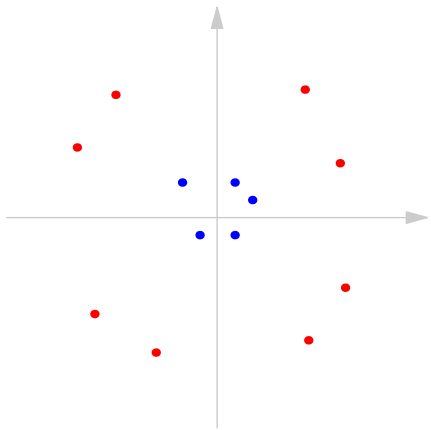
What can a thresholded two layer network compute?



Multi Layer Feed-Forward Networks

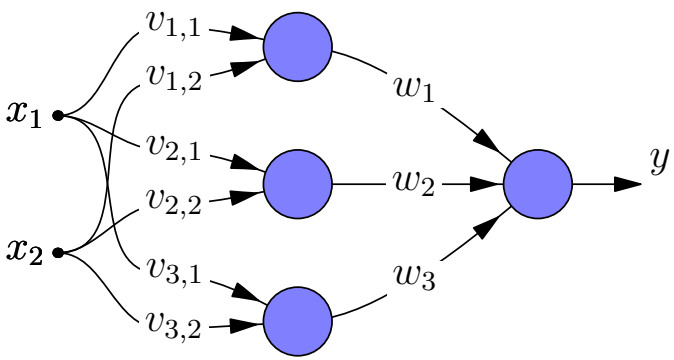
What can a thresholded two layer network compute?

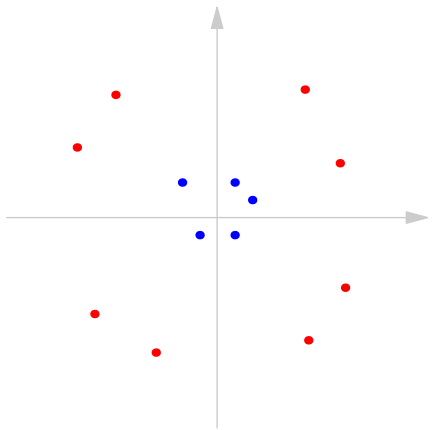


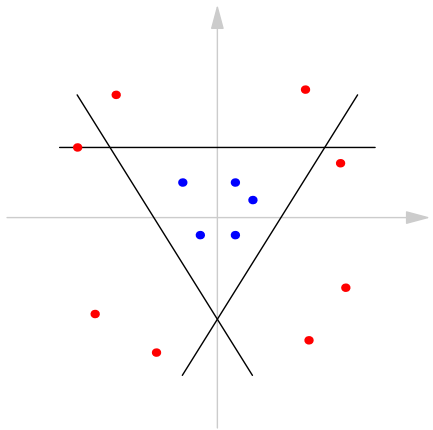


Multi Layer Feed-Forward Networks

What can a thresholded two layer network compute?

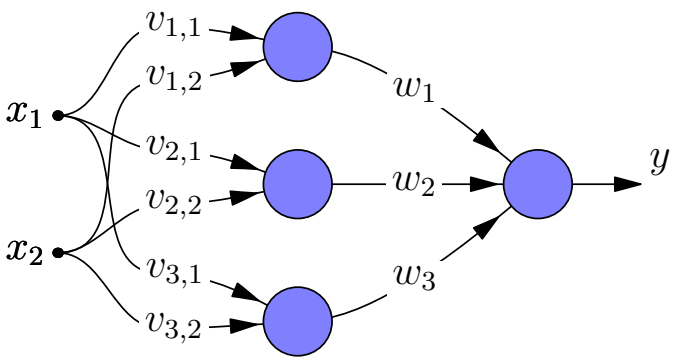


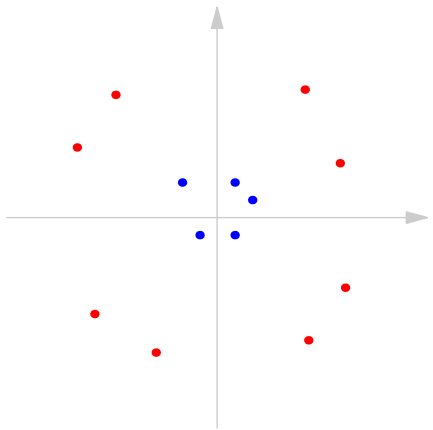


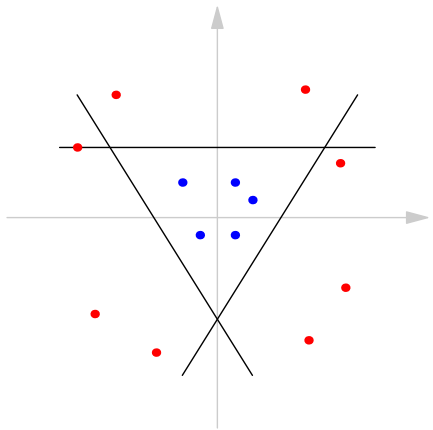


Multi Layer Feed-Forward Networks

What can a thresholded two layer network compute?







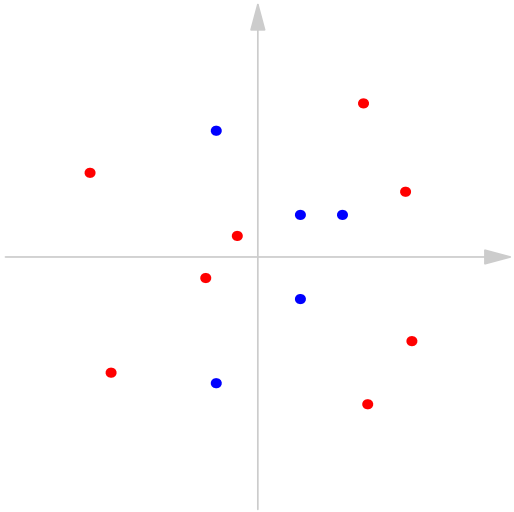
With $w_1 = w_2 = w_3 = 1$ and $\theta = 2.5$ the second layer operates as an AND-gate.

Multi Layer Feed-Forward Networks

What happens if the area is not convex?

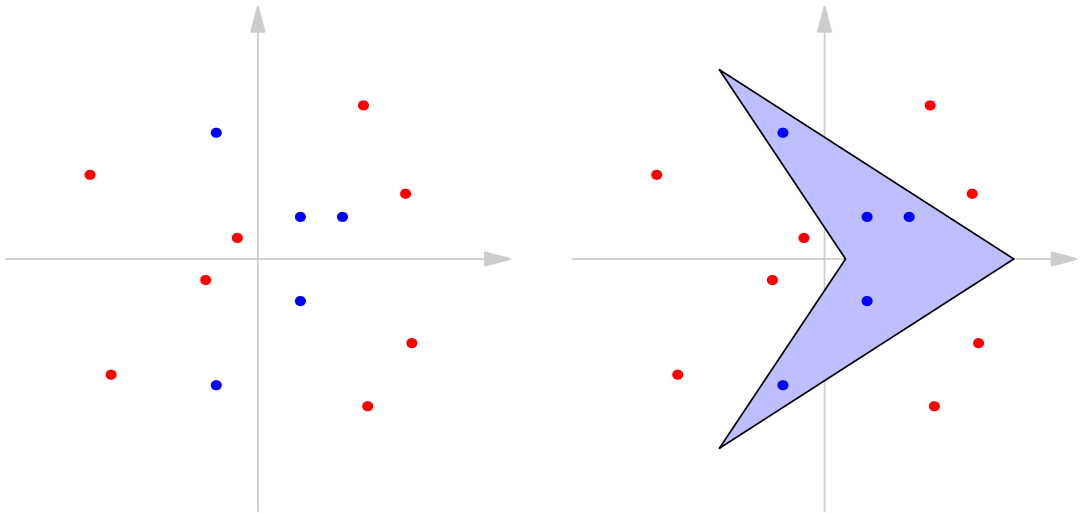
Multi Layer Feed-Forward Networks

What happens if the area is not convex?



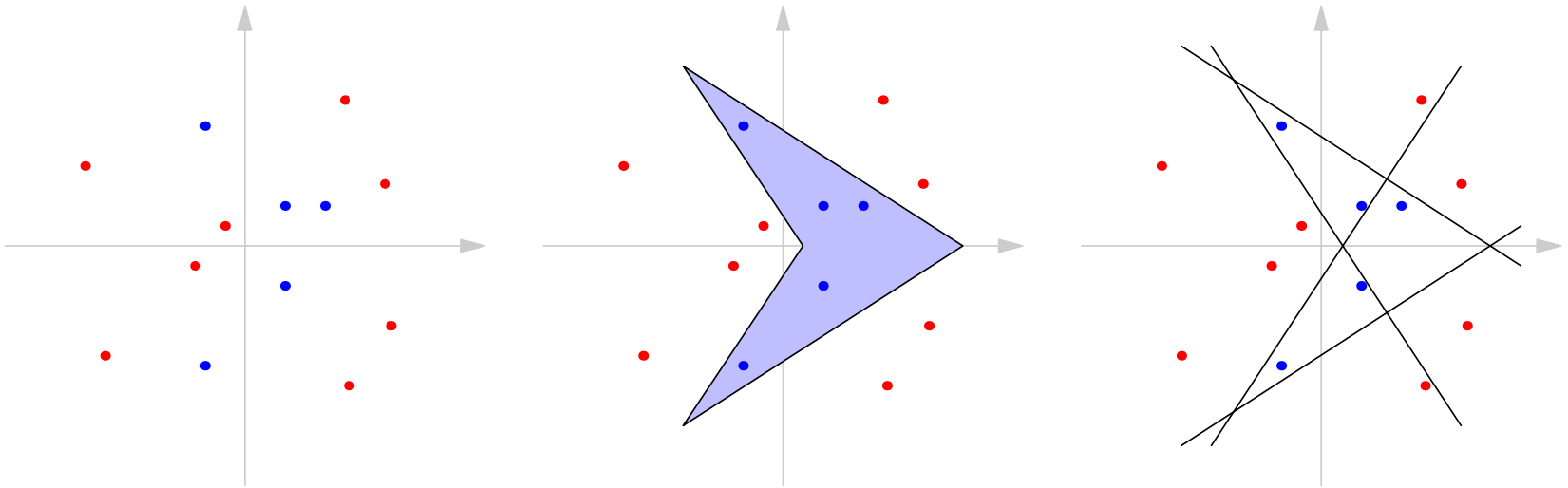
Multi Layer Feed-Forward Networks

What happens if the area is not convex?



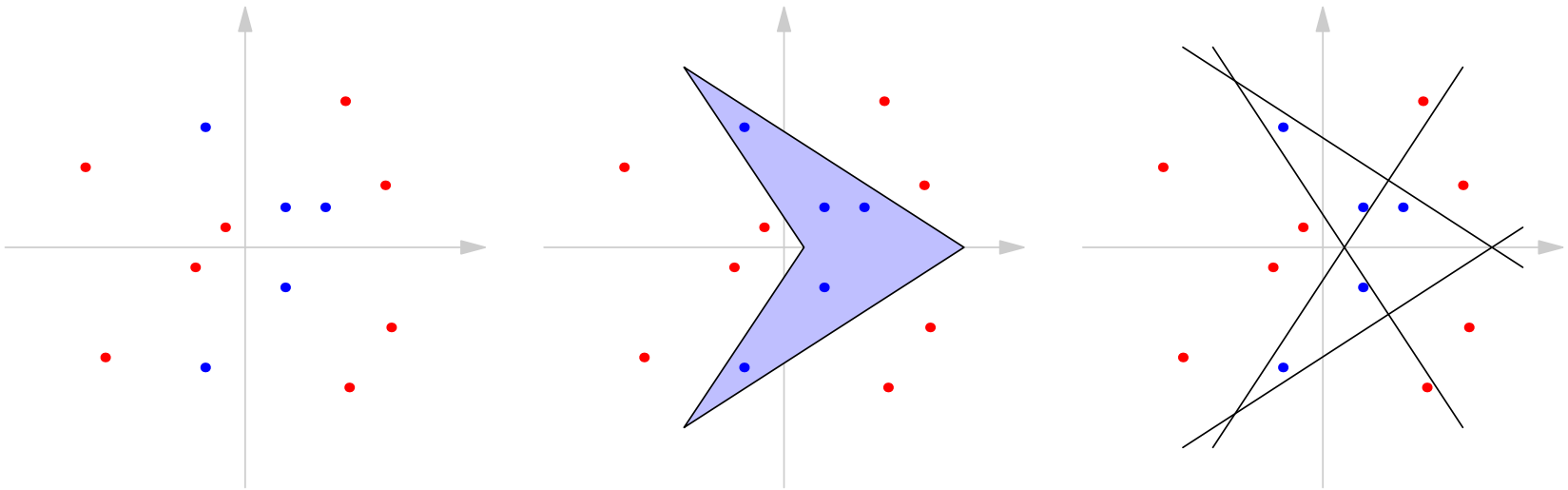
Multi Layer Feed-Forward Networks

What happens if the area is not convex?



Multi Layer Feed-Forward Networks

What happens if the area is not convex?



Arbitrarily complex areas can be extracted provided there are enough hidden units

Learning

How can we train a multi layer network?

- Perceptron Learning
- Delta Rule

Learning

How can we train a multi layer network?

- Perceptron Learning
  Requires that we know in which direction the weights should be changed to come nearer the solution.

- Delta Rule

Learning

How can we train a multi layer network?

- Perceptron Learning
  Requires that we know in which direction the weights should be changed to come nearer the solution.
  Does not work!
- Delta Rule

Learning

How can we train a multi layer network?

- Perceptron Learning
  Requires that we know in which direction the weights should be changed to come nearer the solution.
  Does not work!
- Delta Rule
  Requires that we can measure the error before thresholding, but this only works for the last layer.

Learning

How can we train a multi layer network?

- Perceptron Learning
  Requires that we know in which direction the weights should be changed to come nearer the solution.
  Does not work!
- Delta Rule
  Requires that we can measure the error before thresholding, but this only works for the last layer.
  Does not work!

Learning

Dilemma:

- Thresholding destroys information needed for learning
- Without thresholding we loose the advantage of multiple layers

Learning

Dilemma:

- Thresholding destroys information needed for learning
- Without thresholding we loose the advantage of multiple layers

Solution

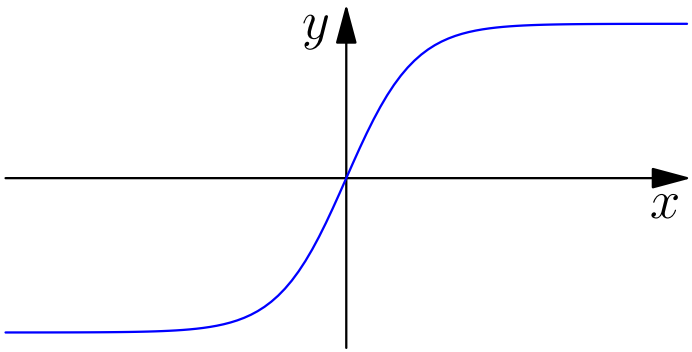
Use threshold-like but differentiable transfer functions

Learning

Two commonly used transfer functions $\varphi(\sum)$

Learning

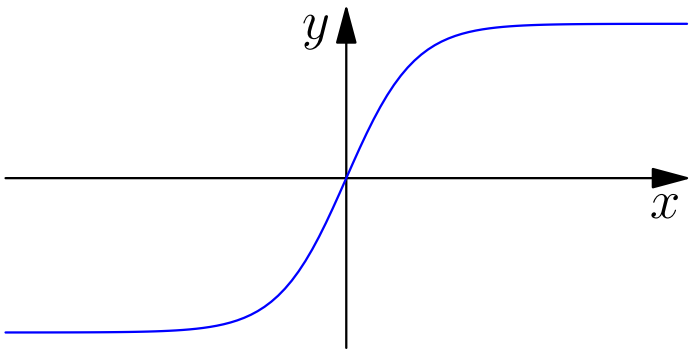
Two commonly used transfer functions $\varphi(\sum)$

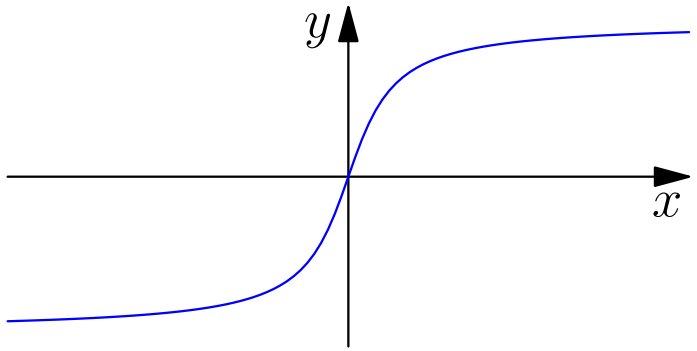


$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Learning

Two commonly used transfer functions $\varphi(\sum)$





$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

$$\varphi(x) = \arctan(x)$$

Generalization of the Delta Rule:

Generalization of the Delta Rule:

1. Choose a cost function ε

2. Minimize it using Steepest Decent

Generalization of the Delta Rule:

1. Choose a cost function ε

$$\varepsilon = \frac{1}{2}||\vec{t} - \vec{y}||^2 = \frac{1}{2}\sum_k (t_k - y_k)^2$$

2. Minimize it using Steepest Decent

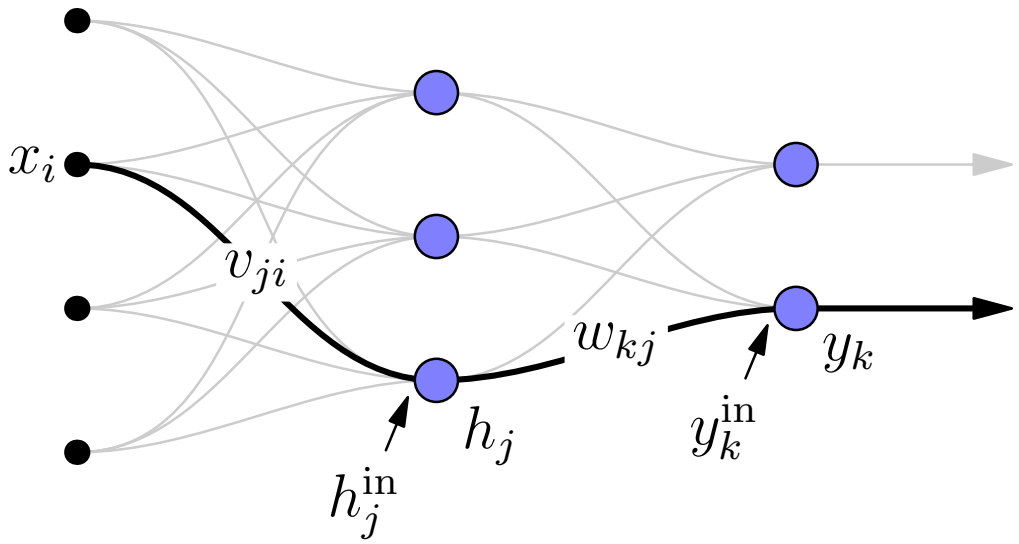
Generalization of the Delta Rule:

1. Choose a cost function ε

$$\varepsilon = \frac{1}{2}||\vec{t} - \vec{y}||^2 = \frac{1}{2}\sum_k (t_k - y_k)^2$$

2. Minimize it using Steepest Decent
   Compute the gradient, i.e.

$$\frac{\partial \varepsilon}{\partial v_{ji}} \quad \text{and} \quad \frac{\partial \varepsilon}{\partial w_{kj}}$$


x_i
v_{ji}
w_{kj}
y_k
h_j
h_j^{in}
y_k^{in}

First case: derivative w.r.t. a weight w_{kj} in the second layer

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial w_{kj}} &= \frac{\partial \varepsilon}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \frac{\partial \varphi(y_k^{\text{in}})}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot \frac{\partial y_k^{\text{in}}}{\partial w_{kj}} \\
&= -(t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot h_j \\
&= -\delta_k h_j
\end{aligned}$$

Here we have introduced $\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}})$

Second case: derivative w.r.t. a weight v_{kj} in the first layer

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}} &= \sum_k \frac{\partial \varepsilon}{\partial y_k} \cdot \frac{\partial y_k}{\partial v_{ji}} \\
&= -\sum_k (t_k - y_k) \cdot \frac{\partial y_k}{\partial v_{ji}} \\
&= -\sum_k (t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \cdot \frac{\partial y_k^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot \frac{\partial y_k^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \frac{\partial h_j}{\partial v_{ji}}
\end{aligned}$$

We continue...

$$\begin{aligned}
\frac{\partial \varepsilon}{\partial v_{ji}} &= -\sum_k \delta_k \cdot w_{kj} \cdot \frac{\partial h_j}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}}) \cdot \frac{\partial h_j^{\text{in}}}{\partial v_{ji}} \\
&= -\sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}}) \cdot x_i \\
&= -\delta_j x_i
\end{aligned}$$

Here we have introduced $\delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$

Summary

$$\frac{\partial \varepsilon}{\partial w_{kj}} = -\delta_k h_j \qquad \text{where } \delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}})$$

$$\frac{\partial \varepsilon}{\partial v_{ji}} = -\delta_j x_i \qquad \text{where } \delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$$

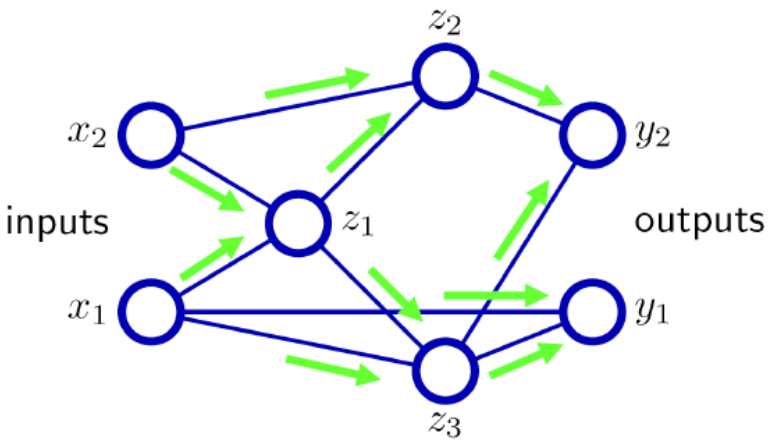
Gradient Decent

$$\Delta w_{kj} = \eta \delta_k h_j$$

$$\Delta v_{ji} = \eta \delta_j x_i$$

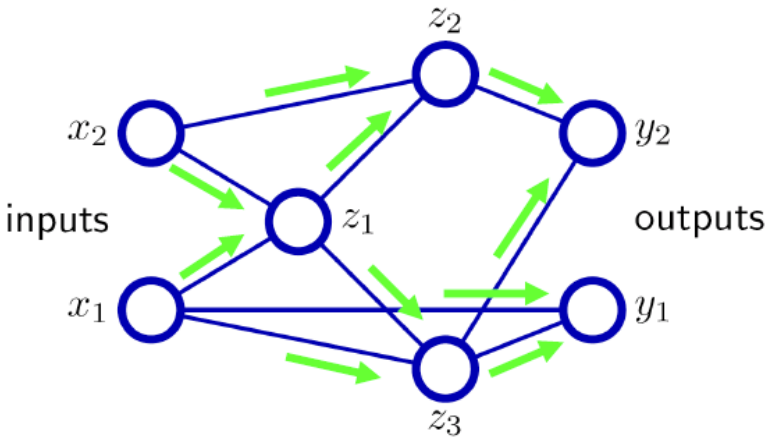
Backprop concept - summary

Forward pass

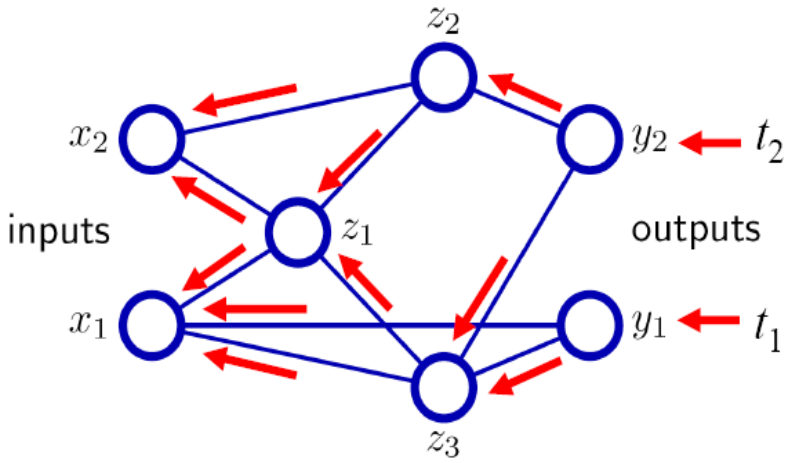


Backprop concept - summary

Forward pass



Backward pass



Error Back-Propagation

1. **Forward Pass**: Compute all h_j and y_k

$$h_j = \varphi(\sum_i v_{ji} x_i) \qquad y_k = \varphi(\sum_j w_{kj} h_j)$$

2. **Backward Pass**: Compute all δ_k and δ_j

$$\delta_k = (t_k - y_k) \cdot \varphi'(y_k^{\text{in}}) \qquad \delta_j = \sum_k \delta_k \cdot w_{kj} \cdot \varphi'(h_j^{\text{in}})$$

3. **Weight Updating**:

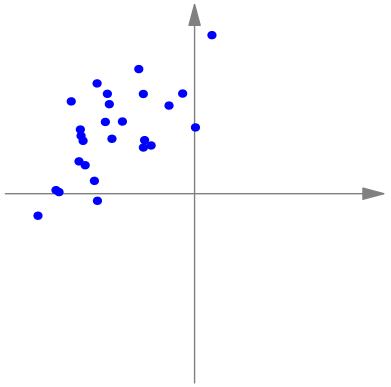
$$\Delta w_{kj} = \eta \delta_k h_j \qquad \Delta v_{ji} = \eta \delta_j x_i$$

Problems with BackProp

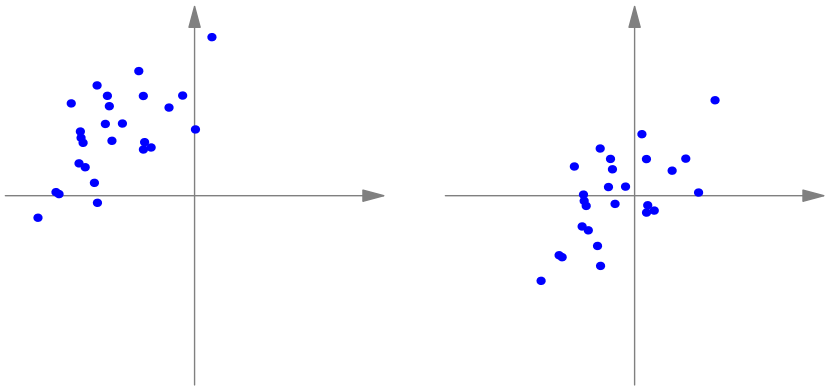
- Does not always converge (gets stuck in local minima)
- Slow convergence
- Many parameters need to be tuned
- Bad scaling behavior for large problems
- Biologically unrealistic
  - Backward propagating signal
  - Requires known target values

Tips when using BackProp

- Use an antisymmetric $\varphi(x)$
- Put the target values $\vec{t}$ inside the domain interval of φ
- Order or weigh the training examples so the hard examples dominate
- Choose smart initial weights
- Introduce momentum in the weight updating
- Add random noise to the weights during training
- Remove the squashing-function ($\varphi(x)$) for the output units



Preprocessing of input patterns

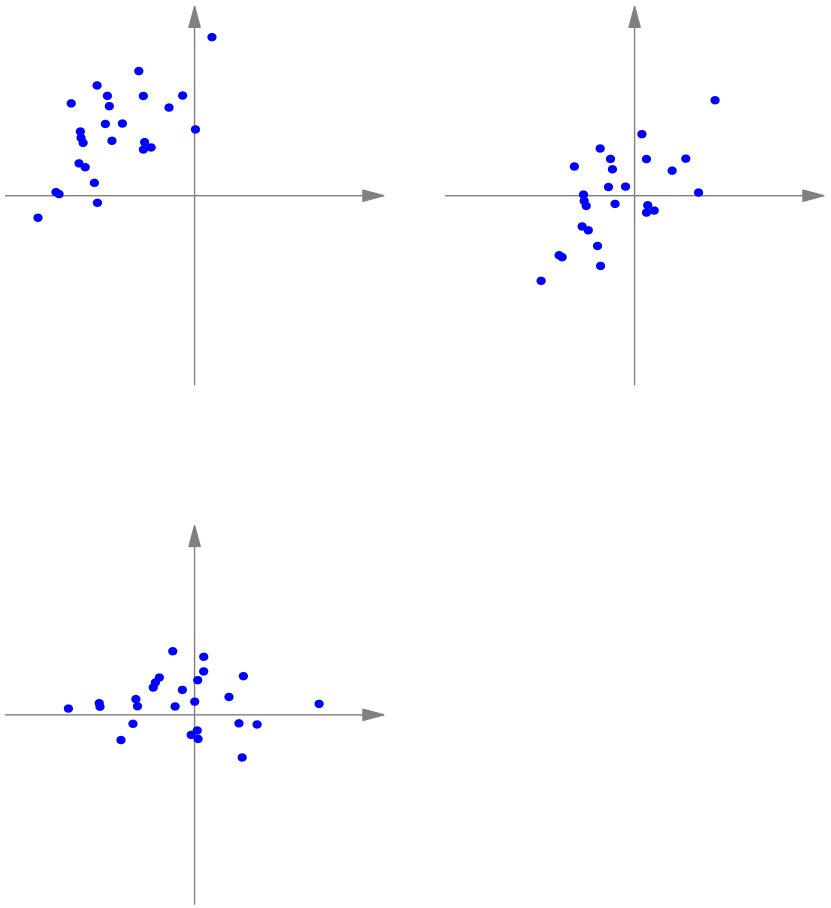


Preprocessing of input patterns

1. Subtract the average

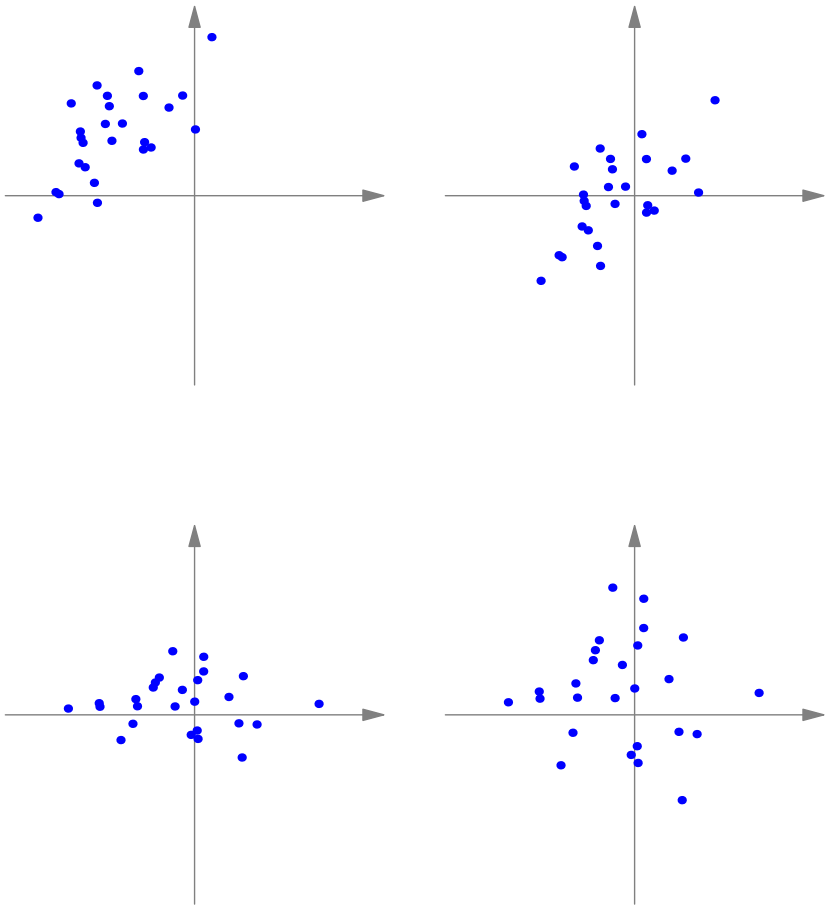
Preprocessing of input patterns

1. Subtract the average
2. Decorrelate



Preprocessing of input patterns

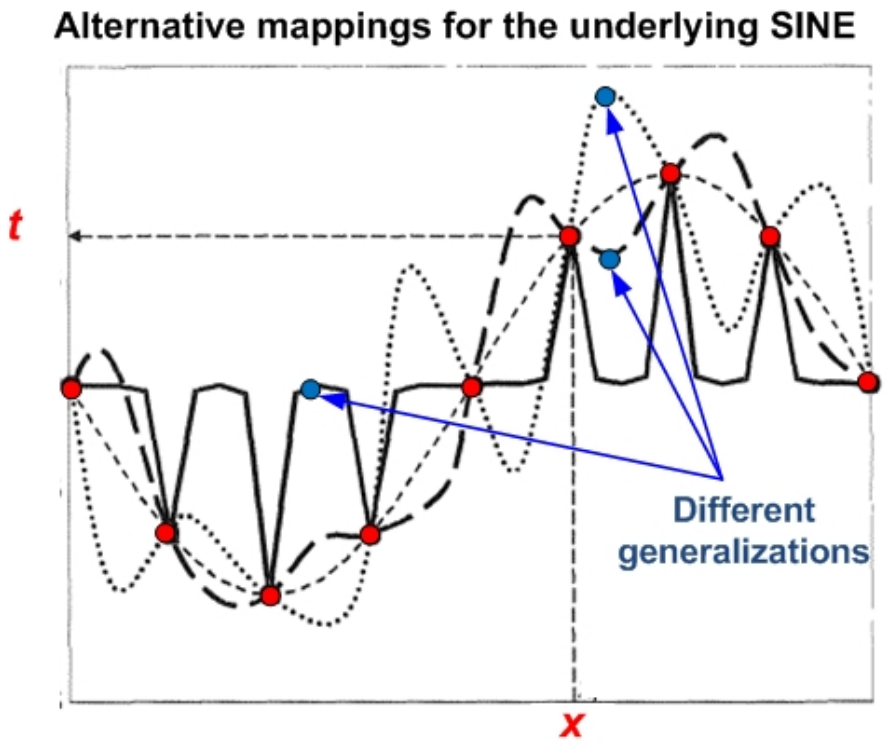
1. Subtract the average
2. Decorrelate
3. Normalize the variance



What is generalisation?

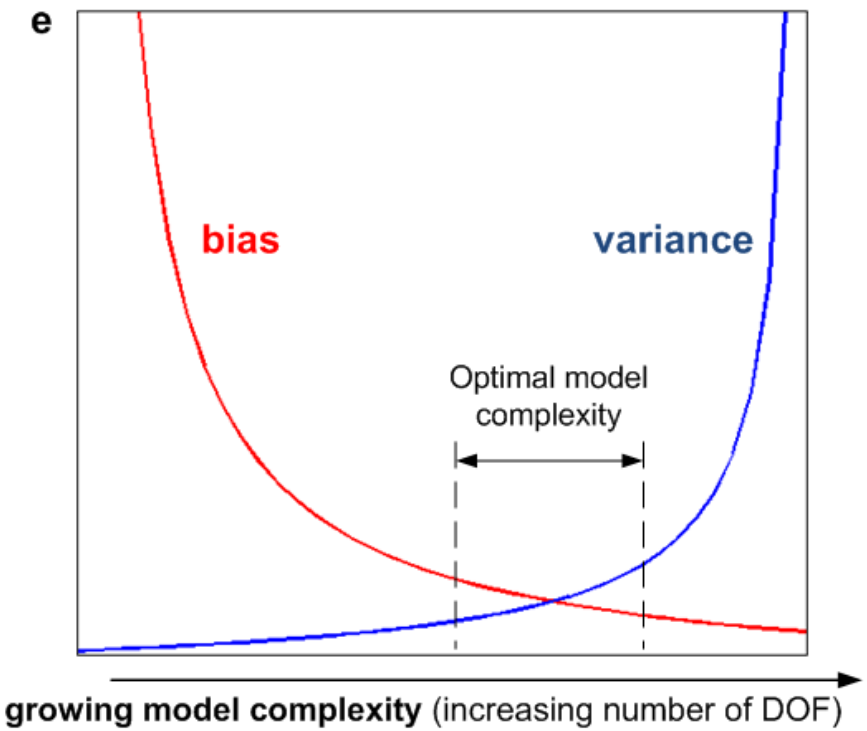
- **Generalisation** is the capacity to apply learned knwoledge to new situations
  - the capability of a learning machine to perform well on *unseen data*
  - a tacit assumption that the test set is drawn from the same distribution as the training one
- Generalisation is affected by the training data size, complexity of the learning machine (e.g. NN) and the physical complexity of the underlying problem
- Analogy to curve-fitting problem - nonlinear mapping

Overfitting phenomenon



- network memorises the training data (noise fitting)
- instead it should learn the underlying function
- on the other hand, the danger of underfitting/undertraining

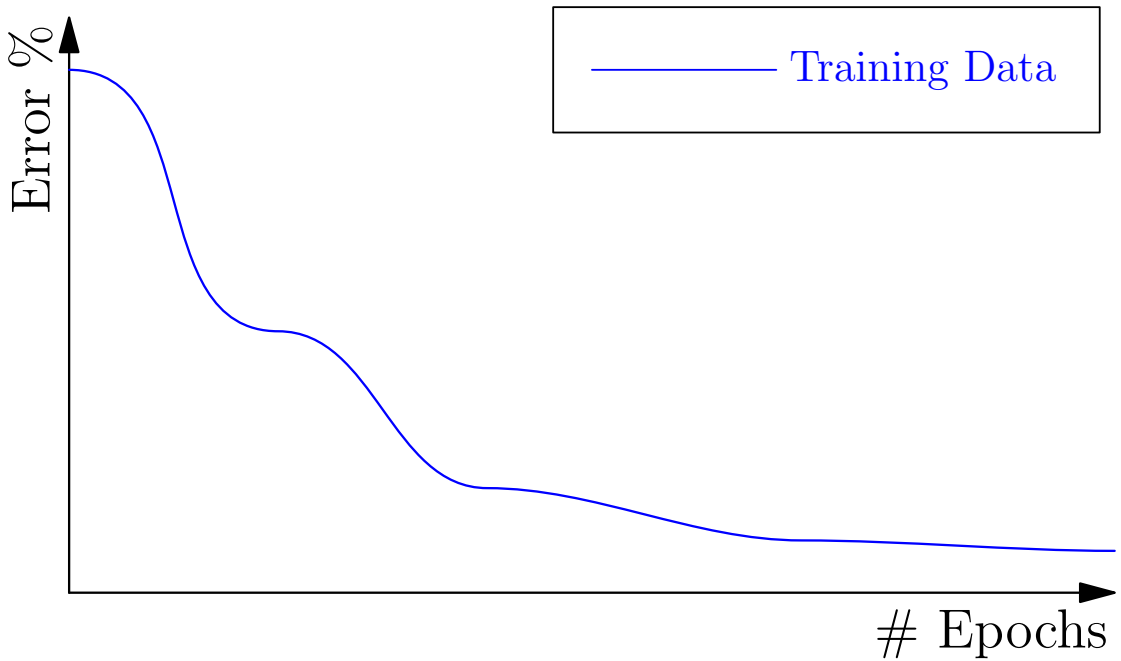
Bias-variance illustration



- the problem can be alleviated by increasing training data size
- otherwise, the complexity of the model has to be restricted
- for NNs, identification of the optimal network architecture

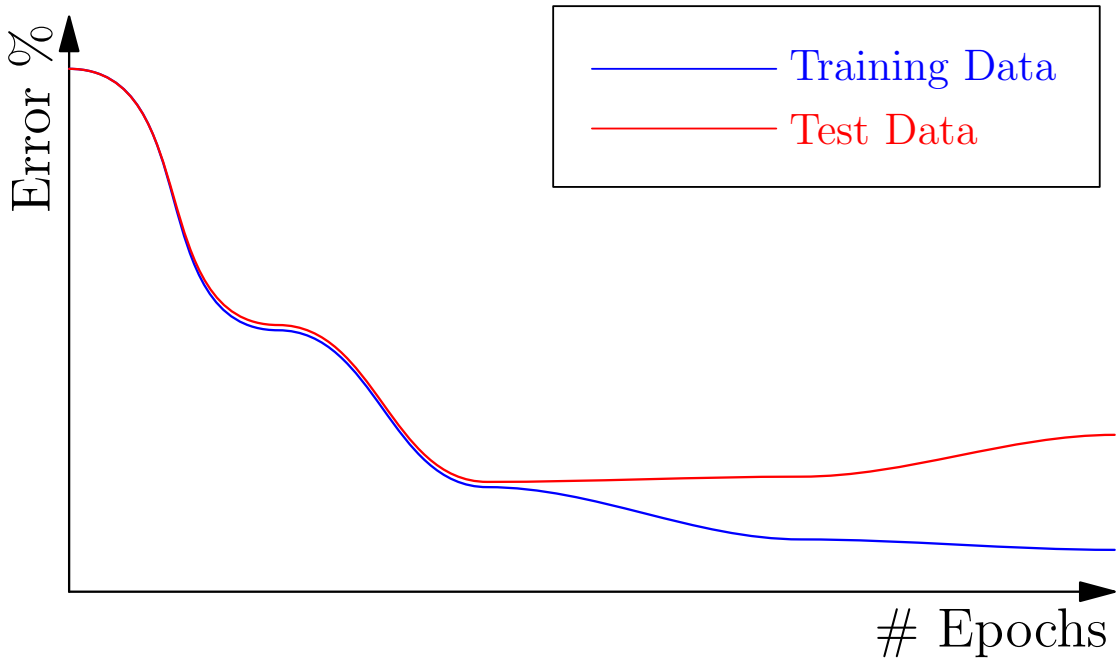
Overfitting

Typical learning curve



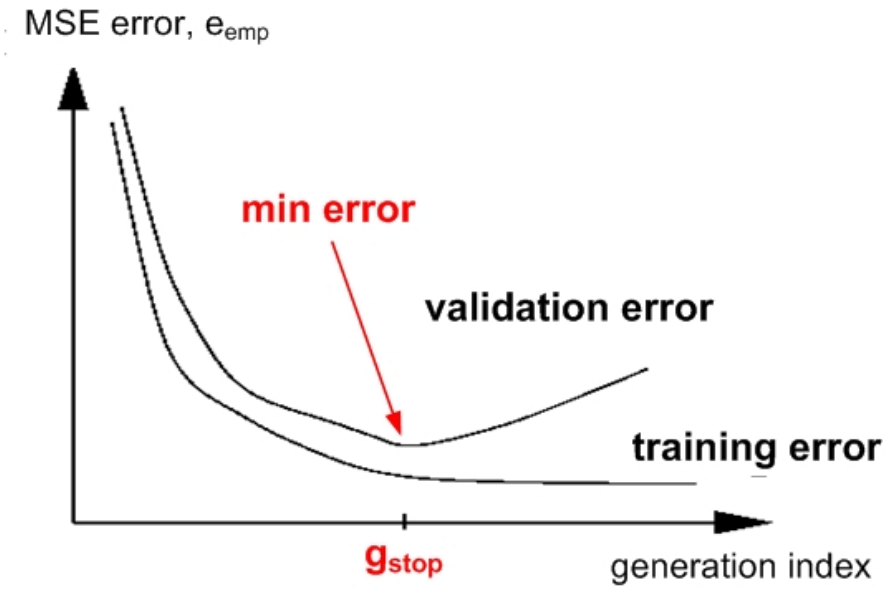
Overfitting

Typical learning curve



Early stopping

- An additional data set is required - **validation set** (split of original data)
- The network is trained with BP until the error monitored on the validation set reaches minimum (further on only noise fitting)



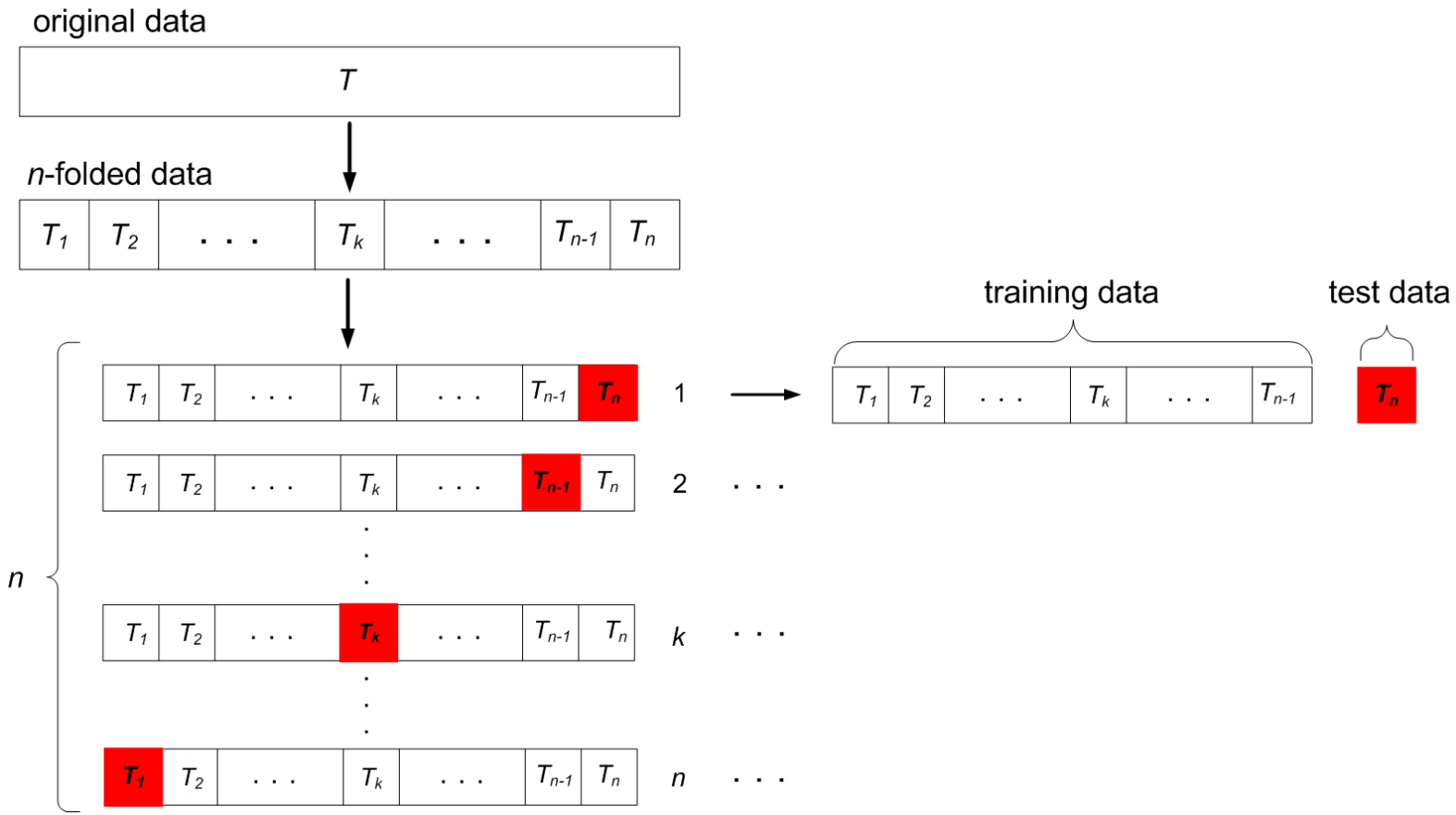
- For quadratic error, it corresponds to learning with weight decay

Model selection and verification

- Empirical assessment of generalisation capabilities
  - allows for model verification, comparison and thus selection
  - separate training and test sets - the simplest approach

- Basic ***hold-out*** (also referred to as *cross-validation*) method often relies on 3 sets and is commonly combined with early stopping
- The cost of sacrificing original data for testing (>10%) can be too high for small data sets

n-fold cross-validation



$$E_{\text{true}}^{\text{CV}} = \frac{1}{n}\sum E_{\text{emp}}^{T \setminus T_k \to T_k}$$