Machine Learning and Pervasive Computing

Stephan Sigg

Georg-August-University Goettingen, Computer Networks

03.06.2015

Overview and Structure

13.04.2015	Organisation
13.04.2015	Introduction
20.04.2015	Rule-based learning
27.04.2015	Decision Trees
04.05.2015	A simple Supervised learning algorithm
11.05.2015	–
18.05.2015	Excursion: Avoiding local optima with random search
25.05.2015	–
01.06.2015	High dimensional data
08.06.2015	Artificial Neural Networks
15.06.2015	k-Nearest Neighbour methods
22.06.2015	Probabilistic models
29.06.2015	Topic models
06.07.2015	Unsupervised learning
13.07.2015	Anomaly detection, Online learning, Recom. systems

Outline

Introduction

Perceptron algorithm

Neural networks
- Introduction
- Definition
- Classification
- Training Neural Networks
- Example: Backpropagation learning

Gradient calculation via backpropagation

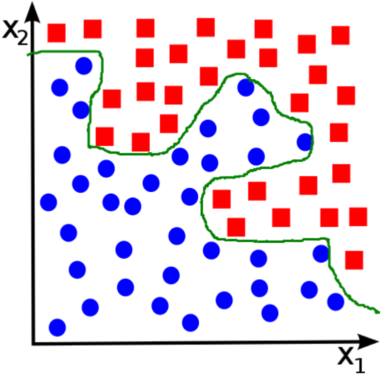
Neural Networks for dimensionality reduction

Motivation

Why yet another classification algorithm ?

Motivation

Why yet another classification algorithm ?

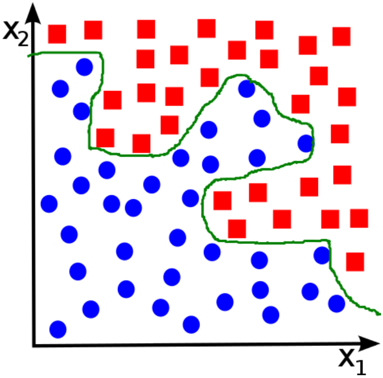


Regression Complex classification requires an enormous number of features

$$w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2x_2 + w_5x_1^3x_2 + w_6x_1x_2^2 + \dots$$

Motivation

Why yet another classification algorithm ?



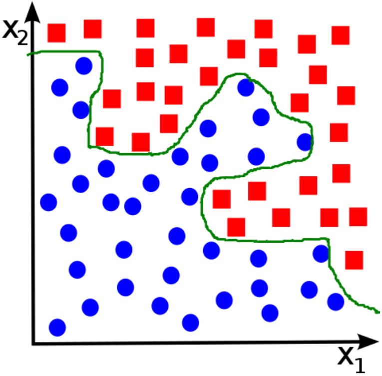
Regression Complex classification requires an enormous number of features

$$w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2x_2 + w_5x_1^3x_2 + w_6x_1x_2^2 + \ldots$$

SVM Finding of optimal kernel yet unsolved problem

Motivation

Why yet another classification algorithm ?



Regression Complex classification requires an enormous number of features

$$w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2x_2 + w_5x_1^3x_2 + w_6x_1x_2^2 + \dots$$

SVM Finding of optimal kernel yet unsolved problem

Artificial Neural Networks are capable of implicitly learning appropriate features also for complex non-linear decision boundaries

Outline

Introduction

Perceptron algorithm

Neural networks
- Introduction
- Definition
- Classification
- Training Neural Networks
- Example: Backpropagation learning

Gradient calculation via backpropagation

Neural Networks for dimensionality reduction

The perceptron algorithm

1962 Frank Rosenblatt introduces the linear discriminant preceptron algorithm

The perceptron algorithm

1962 Frank Rosenblatt introduces the linear discriminant preceptron algorithm

Two-class model ($t \in \{-1, 1\}$) in which the input vector x is first nonlinearly transformed to the feature vector $\phi(x)$:

$$y(x) = f(w^T \phi(x))$$

The perceptron algorithm

1962 Frank Rosenblatt introduces the linear discriminant preceptron algorithm

Two-class model ($t \in \{-1, 1\}$) in which the input vector x is first nonlinearly transformed to the feature vector $\phi(x)$:

$$y(x) = f(w^T \phi(x))$$

nonlinear activation function defined as a step-function:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$

The perceptron algorithm

1962 Frank Rosenblatt introduces the linear discriminant preceptron algorithm

Two-class model ($t \in \{-1, 1\}$) in which the input vector x is first nonlinearly transformed to the feature vector $\phi(x)$:

$$y(x) = f(w^T \phi(x))$$

nonlinear activation function defined as a step-function:

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0. \end{cases}$$

$\phi(x)$ typically includes a BIAS-component $\phi_0(x) = 1$

The perceptron algorithm

Training: We are looking for an Error function for a weight vector $\overrightarrow{w}$ such that

$$\begin{array}{ll} x_i \in C_1 : & \overrightarrow{w}^T \phi(x_i) > 0 \\ x_i \in C_2 : & \overrightarrow{w}^T \phi(x_i) < 0 \end{array}$$

Perceptron criterion

For the set $\mathcal{M}$ of all misclassified patterns, the perceptron criterion is given as

$$E_P(\overrightarrow{w}) = -\sum_{i \in \mathcal{M}} \overrightarrow{w}^T \phi(\overrightarrow{w}_i) t_i$$

The perceptron algorithm

Perceptron criterion

For the set $\mathcal{M}$ of all misclassified patterns, the perceptron criterion is given as

$$E_P(\overrightarrow{w}) = -\sum_{i\in\mathcal{M}} \overrightarrow{w}^T \phi(\overrightarrow{w}_i) t_i$$

The error function is piecewise linear:

linear in regions of $\overrightarrow{w}$-space where pattern is misclassfied

0 in regions where it is classified correctly

The perceptron algorithm

Perceptron criterion

For the set $\mathcal{M}$ of all misclassified patterns, the perceptron criterion is given as

$$E_P(\overrightarrow{w}) = -\sum_{i \in \mathcal{M}} \overrightarrow{w}^T \phi(\overrightarrow{w}_i) t_i$$

The error function is piecewise linear:

linear in regions of $\overrightarrow{w}$-space where pattern is misclassfied

0 in regions where it is classified correctly

Apply stochastic gradient descent to this error function:

$$\overrightarrow{w}^{t+1} = \overrightarrow{w}^t - \delta \nabla E_P(\overrightarrow{w}) = \overrightarrow{w}^t + \delta \phi(\overrightarrow{w_i}) t_i$$

The perceptron algorithm

Interpretation of the learning function

$$\overrightarrow{w}^{t+1} = \overrightarrow{w}^{t} - \delta \nabla E_P(\overrightarrow{w}) = \overrightarrow{w}^{t} + \delta \phi(\overrightarrow{w_i}) t_i$$

The perceptron algorithm

Interpretation of the learning function

$$\overrightarrow{w}^{t+1} = \overrightarrow{w}^{t} - \delta \nabla E_P(\overrightarrow{w}) = \overrightarrow{w}^{t} + \delta \phi(\overrightarrow{w_i}) t_i$$

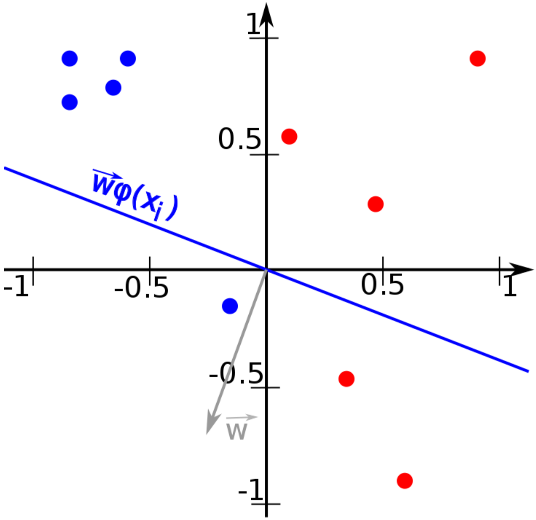
for each x_i:

correct classification: weight vector remains unchanged

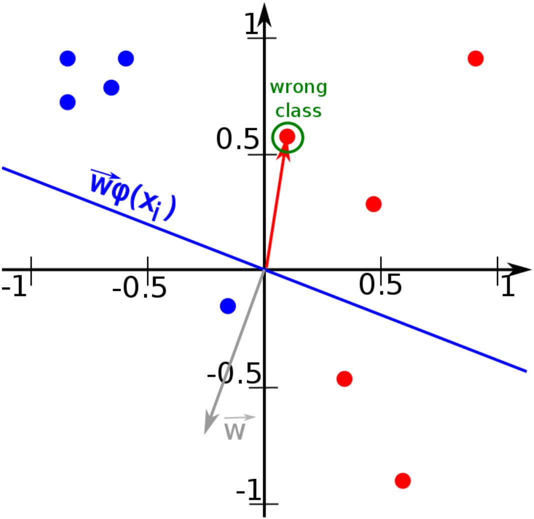
incorrect classification:

$$\begin{aligned} \text{Class}\, C_1 &: \quad \text{add vector } \phi(\overrightarrow{w_i}) \\ \text{Class}\, C_2 &: \quad \text{subtract vector } \phi(\overrightarrow{w_i}) \end{aligned}$$

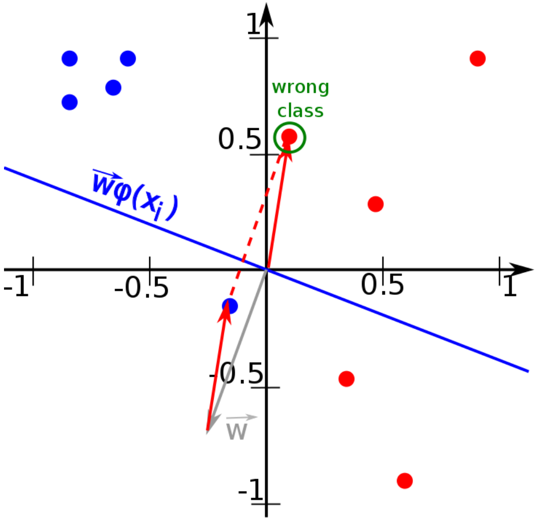
The perceptron algorithm



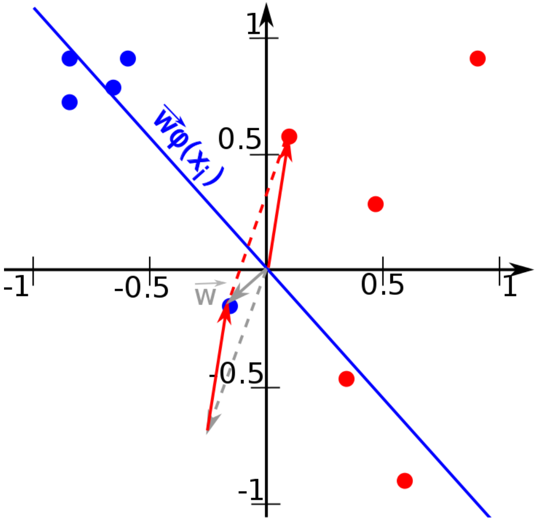
The perceptron algorithm



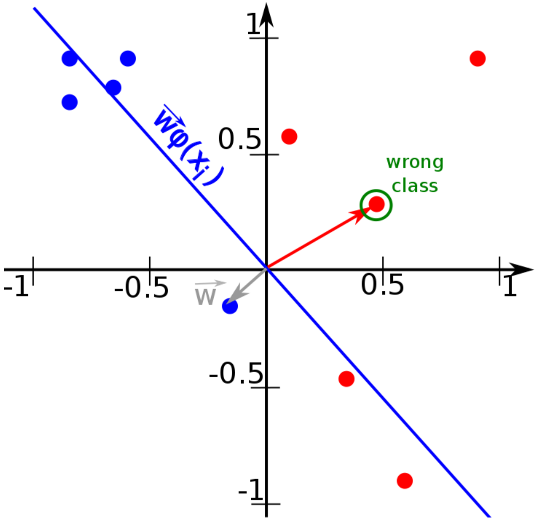
The perceptron algorithm



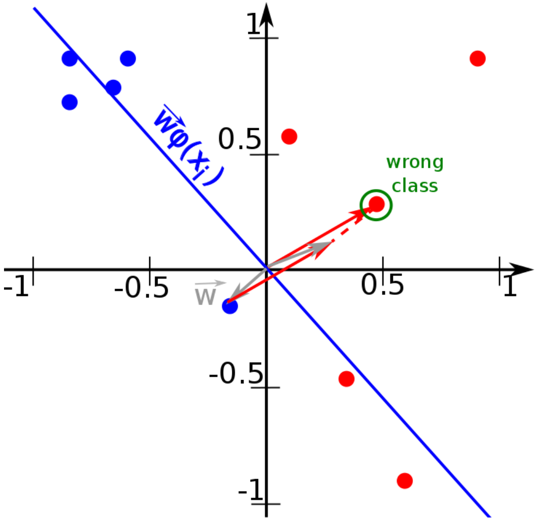
The perceptron algorithm



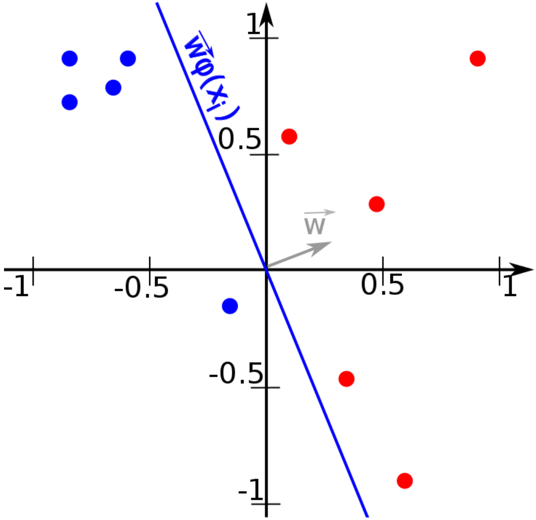
The perceptron algorithm



The perceptron algorithm



The perceptron algorithm



The perceptron algorithm

Perceptron convergence theorem

Iff the training data is linearly separable, then the perceptron learning algorithm will always find an exact solution in finite number of steps.

- $\rightarrow$ Still, number of steps required might be very large
- $\rightarrow$ Until convergence, it is not possible to distinguish separable problem from non-separable
- $\rightarrow$ For on-separable data sets the algorithm will never converge

Outline

Introduction

Perceptron algorithm

Neural networks
- Introduction
- Definition
- Classification
- Training Neural Networks
- Example: Backpropagation learning

Gradient calculation via backpropagation

Neural Networks for dimensionality reduction

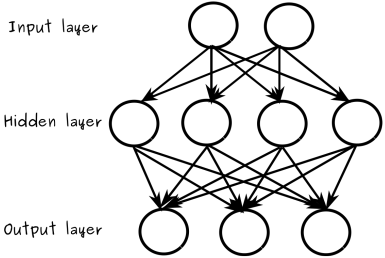
Neural networks

Learn mapping from input to output vector

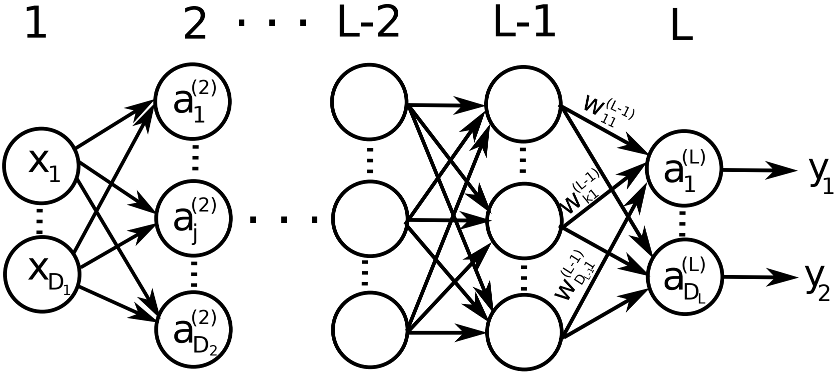
Representation by edge-weighted graph

Distinction between

- Input neurons
- Output neurons
- Hidden nodes

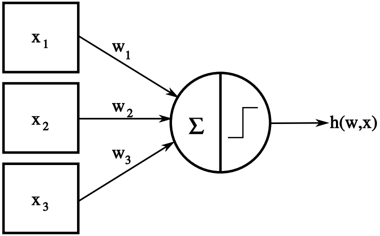


Neural networks



Neural networks

Input neurons are only equipped with outgoing edges



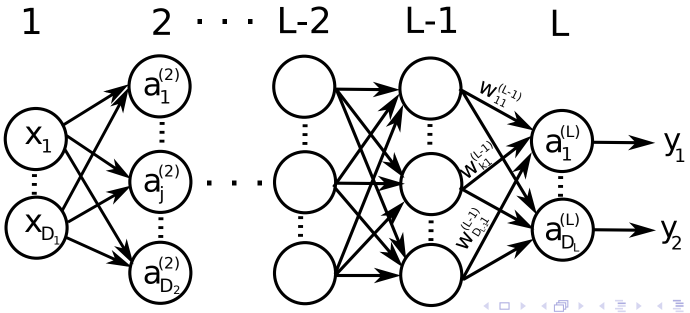
Neural networks

Single hidden layer sufficient to represent arbitrary multi-dimensional functions

Well suited for noisy input data

Implicit clustering of input data possible

Complex to extend network (e.g. add new features)



Neural networks

Neural networks are also known as multilayer perceptrons

Neural networks

Neural networks are also known as multilayer perceptrons

→ However, the model comprises multiple layers of logistic regression models (with continuous nonlinearities) rather than multiple perceptrons (with discontinuous nonlinearities)

(Important, since the model is therefore differentiable which will be required in the learning process)

Neural networks

For the input layer, we construct linear combinations of the input variables $x_1, \ldots, x_{D_1}$ and weights $w_{11}, \ldots, w^{(1)}_{D_1 D_2}$

$$z_j^{(2)} = \sum_{i=1}^{D_1} w_{ij}^{(1)} x_i + w_{0j}^{(1)}$$

Each value $a_j^{(l)}$ in the hidden and output layers $l, l \in \{2, \ldots, L\}$ is computed from $z_j^{(l)}$ using a differentiable, non-linear activation function

$$a_j^{(l)} = f_{\text{act}}^{(l)}\left(z_j^{(l)}\right)$$

Neural networks

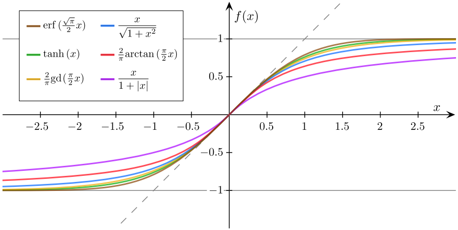
Input layer linear combinations of $x_1, \ldots, x_{D_1}$ and $w_{11}, \ldots, w_{D_1 D_2}$

$$z_j^{(2)} = \sum_{i=1}^{D_1} w_{ij}^{(1)} x_i + w_{0j}^{(1)}$$

Activation function: Differentiable, non-linear

$$a_j^{(2)} = f_{\text{act}}^{(2)}\left(z_j^{(2)}\right)$$

$f_{\text{act}}(\cdot)$ function is usually a sigmoidal function or tanh



Neural networks

Values $a_j^{(2)}$ are then linearly combined in hidden layers:

$$z_k^{(3)} = \sum_{j=1}^{D_2} w_{jk}^{(2)} a_j^{(2)} + w_{0k}^{(2)}$$

with $k = 1, \ldots, D_L$ describing the total number of outputs

Again, these values are transformed using a sufficient transformation function f_{act} to obtain the network outputs

$$f_{\text{act}}^{(3)}(z_k^{(3)})$$

Neural networks

Combine these stages to achieve overall network function:

$$h_k(\vec{x}, \vec{w}) = f_{\text{act}}^{(3)} \left(\sum_{j=1}^{D_2} w_{jk}^{(2)} f_{\text{act}}^{(2)} \left(\sum_{i=1}^{D_1} w_{ij}^{(1)} x_i + w_{0j}^{(1)} \right) + w_{0k}^{(2)} \right)$$

(Multiple hidden layers are added analogously)

Neural networks

Combine these stages to achieve overall network function:

$$h_k(\overrightarrow{x}, \overrightarrow{w}) = f_{\text{act}}^{(3)} \left(\sum_{j=1}^{D_2} w_{jk}^{(2)} f_{\text{act}}^{(2)} \left(\sum_{i=1}^{D_1} w_{ij}^{(1)} x_i + w_{0j}^{(1)} \right) + w_{0k}^{(2)} \right)$$

(Multiple hidden layers are added analogously)

We speak of Forward propagation since the network elements are computed from 'left to right'

Neural networks

Combine these stages to achieve overall network function:

$$h_k(\overrightarrow{x}, \overrightarrow{w}) = f_{\text{act}}^{(3)} \left(\sum_{j=1}^{D_2} w_{jk}^{(2)} f_{\text{act}}^{(2)} \left(\sum_{i=1}^{D_1} w_{ij}^{(1)} x_i + w_{0j}^{(1)} \right) + w_{0k}^{(2)} \right)$$

(Multiple hidden layers are added analogously)

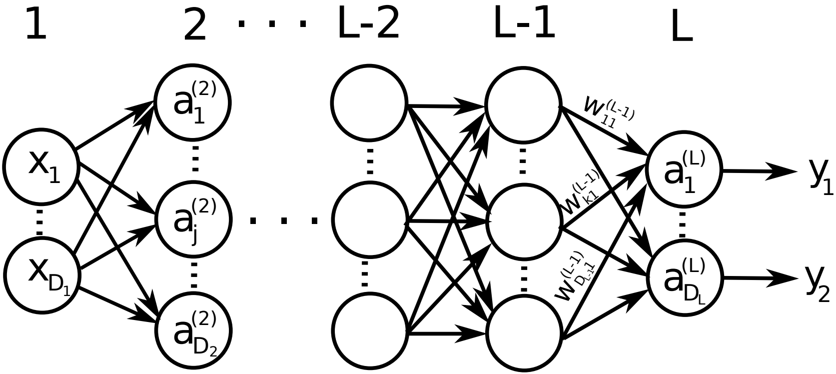
We speak of Forward propagation since the network elements are computed from 'left to right'

This is essentially a logistic regression problem where appropriate features are learned in the first stage of the network

Neural networks

With linear activation functions of hidden units ⇒ Always find equivalent network without hidden units

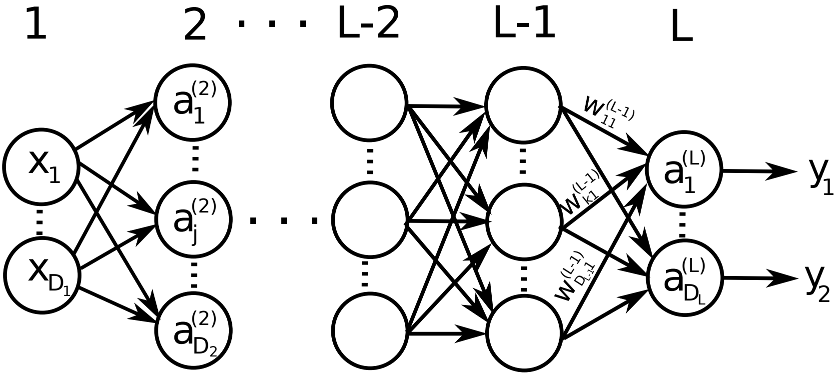
(Composition of successive linear transformations itself linear transformation)



Neural networks

Number of hidden units $<$ number of input or output units $\Rightarrow$ not all linear functions possible

(Information lost in dimensionality reduction at hidden units)



Neural networks

Neural networks are Universal approximators[1] [2] [3] [4] [5] [6] [7] [8]
$\Rightarrow$ 2-layer linear NN can approximate any continuous function

[1] K. Funahashi: On the approximate realisation of continuous mappings by neural networks, Neural Networks, 2(3), 183-192, 1989

[2] G. Cybenko: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2, 304-314, 1989

[3] K. Hornik, M. Sinchcombe, H. White: Multilayer feed-forward networks are universal approximators. Neural Networks, 2(5), 359-366, 1989

[4] N.E. Cotter: The stone-Weierstrass theorem and its application to neural networks. IEEE Transactions on Neural Networks 1(4), 290-295, 1990

[5] Y. Ito: Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks 4(3), 385-394, 1991

[6] K. Hornik: Approximation capabilities of multilayer feed forward networks: Neural Networks, 4(2), 251-257, 1991

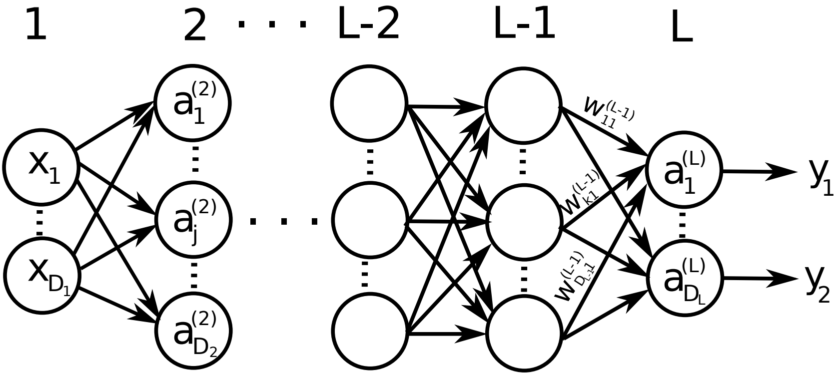
[7] Y.V. Kreinovich: Arbitrary non-linearity is sufficient to represent all functions by neural networks: a theorem. Neural Networks 4(3), 381-383, 1991

[8] B.D. Ripley: Pattern Recognition and Neural Networks. Cambridge University Press, 1996

Neural networks

Remaining issue in neural networks

- Find suitable parameters given a set of training data
- Several learning approaches have been proposed



Neural networks

Simple approach to determine network parameters: Minimise sum-of-squared error function

- Given a training set of samples $\overrightarrow{x_i}$ with $i \in \{1, \ldots, N\}$
- And corresponding targets $\overrightarrow{y_i}$
- Minimise the error function

$$E(\overrightarrow{w}) = \frac{1}{2} \sum_{i=1}^{N} (h(\overrightarrow{x_i}, \overrightarrow{w}) - \overrightarrow{y_i})^2$$

Neural networks – Classification

2 classes $\mathcal{C}_1$ and $\mathcal{C}_2$

- We consider a network with a single output

$$f_{\text{act}}^{(L)}\left(z^{(L)}\right) \equiv \frac{1}{1+e^{-z^{(L)}}}$$

- Output interpreted as conditional probability $\mathcal{P}(\mathcal{C}_1|\overrightarrow{x})$
- Analogously, we have $\mathcal{P}(\mathcal{C}_2|\overrightarrow{x}) = 1 - \mathcal{P}(\mathcal{C}_1|\overrightarrow{x})$

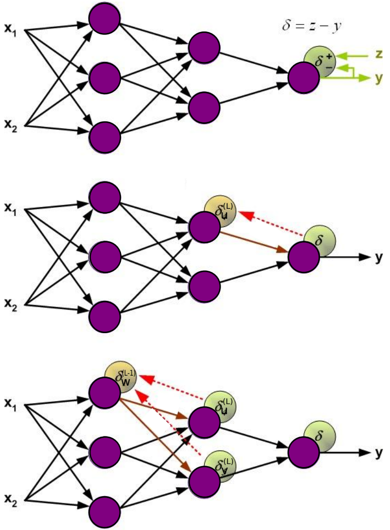
K classes $\mathcal{C}_1, \cdots, \mathcal{C}_K$

- Binary target variables $y_k \in \{0, 1\}$
- Network outputs are interpreted as $h_k(\overrightarrow{x}, \overrightarrow{w}) = \mathcal{P}(y_k = 1|\overrightarrow{x})$

Neural networks – backpropagation (Schematic)

Iterate until the error is sufficiently small

1. Choose training-pair and copy it to the input layer
2. Propagate it through the network
3. Calculate error between computed and expected output
4. Propagate weights back into network to calculate hidden-layer error
5. Adapt weights to the error



Neural networks – Cost function

Cost function for Logistic regression

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i \left(\log h(x_i)\right) + (1 - y_i)\left(\log\left(1 - h(x_i)\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Cost function for Neural networks

Neural networks – Cost function

Cost function for Logistic regression

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m} y_i\left(\log h(x_i)\right) + (1-y_i)\left(\log\left(1-h(x_i)\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

Cost function for Neural networks

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ij}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{D_l}\sum_{j=1}^{D_{l+1}}(w_{ji}^{(l)})^2$$

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Neural networks – Cost function

$$E[W] =$$

$$-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w^{(l)}{}_{vu})^2$$

- m Number of training samples
- C Number of classes (output units)
- L Count of layers
- D_l Number of units at layer l

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}({w^{(l)}}_{vu})^2$$

- m Number of training samples
- C Number of classes (output units)
- L Count of layers
- D_l Number of units at layer l

<u>One cost function for each respective output (class)</u>

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Aim minimise $E[W]$ ($\min_W E[W]$)

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Aim minimise $E[W]$ $(\min_W E[W])$

Required $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Aim minimise $E[W]$ $(\min_{W} E[W])$

Required $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$

Backpropagation (effectively compute $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$)

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Aim minimise $E[W]$ $(\min_{W} E[W])$

Required $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$

Backpropagation (effectively compute $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$)

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{\text{act}}(z^{(l)})$

Neural networks – Cost function

$$E[W] = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{c=1}^{C} y_{ic}\log(h(x_i))_c + (1-y_{ic})\log(1-(h(x_i))_c)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{u=1}^{D_l}\sum_{v=1}^{D_{l+1}}(w_{vu}^{(l)})^2$$

Aim minimise $E[W]$ $(\min_W E[W])$

Required $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$

Backpropagation (effectively compute $\frac{\partial}{\partial w_{vu}^{(l)}} E[W]$)

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{\text{act}}(z^{(l)})$

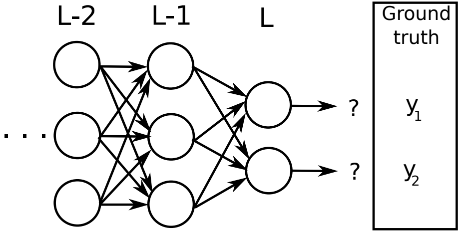
($\circ$ $\rightarrow$ Hadamard product (Element-wise multiplication))

(f'_{act} $\rightarrow$ Derivative of the activation function)

Element-wise multiplication

Hadamard product

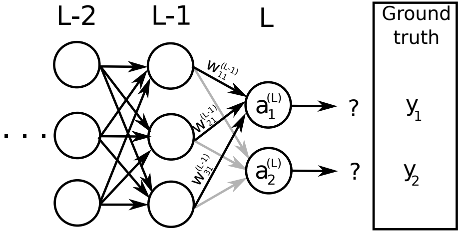
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11}\,b_{11} & a_{12}\,b_{12} & a_{13}\,b_{13} \\ a_{21}\,b_{21} & a_{22}\,b_{22} & a_{23}\,b_{23} \\ a_{31}\,b_{31} & a_{32}\,b_{32} & a_{33}\,b_{33} \end{pmatrix}$$



Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

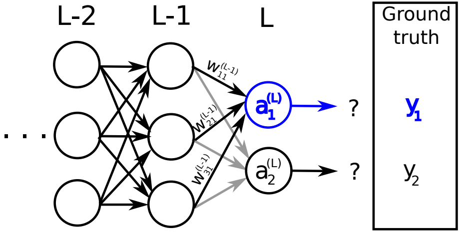
Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$



Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

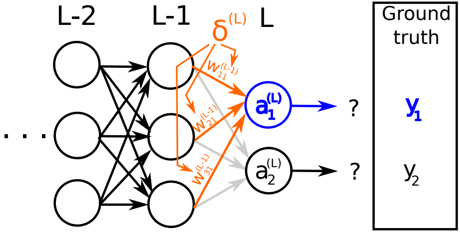
Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$



Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

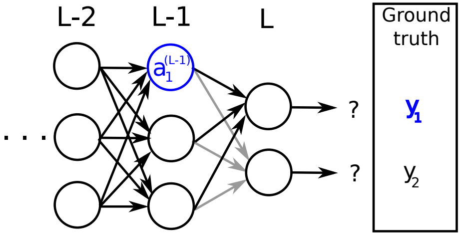
Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$



Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

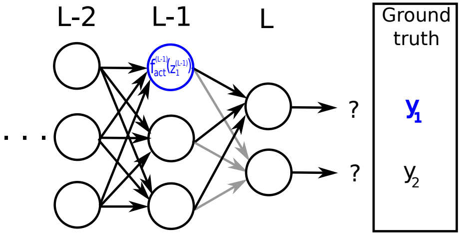


Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{\text{act}}(z^{(l)})$

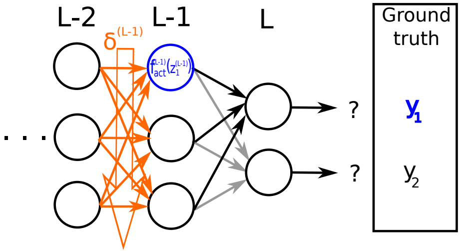


Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{act}(z^{(l)})$

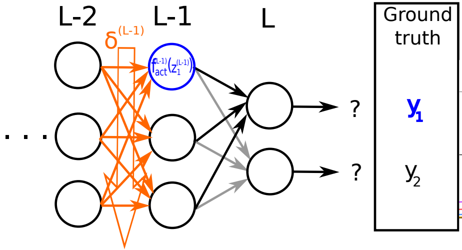


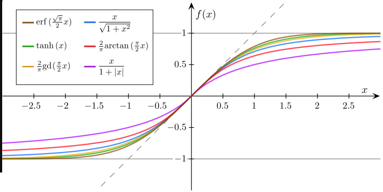
Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{act}(z^{(l)})$



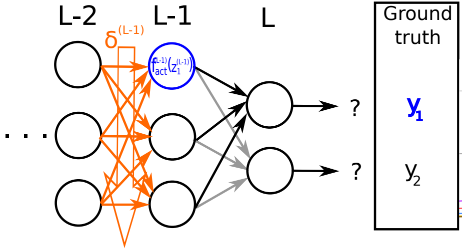


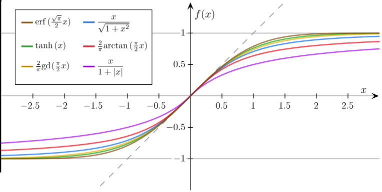
Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

Layer l $\delta^{(l)} = \left(W^{(l)}\right)^T \delta^{(l+1)} \circ f'_{\text{act}}(z^{(l)})$





Backpropagation

$\delta_u^{(l)}$ Error of node j in layer l

Layer L $\delta_u^{(L)} = a_u^{(L)} - y_u \rightarrow \delta^{(L)} = a^{(L)} - y$

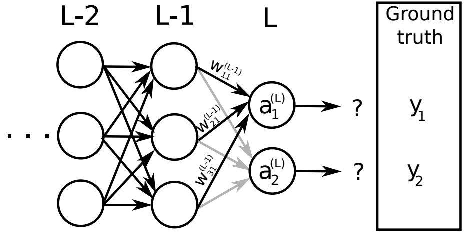
Layer l $\delta^{(l)} = \underbrace{\left(W^{(l)}\right)^T \delta^{(l+1)}}_{\text{direction } \rightarrow (a-y)} \circ \underbrace{f'_{act}(z^{(l)})}_{\text{speed}}$

Remarks

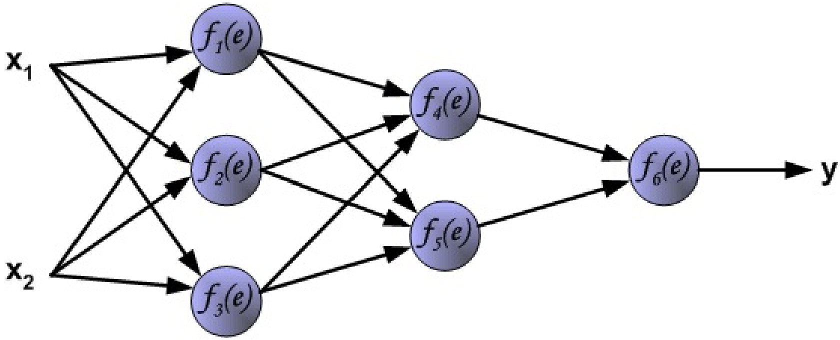
Initialisation of weights

w_{ij} have to be initialised randomly !

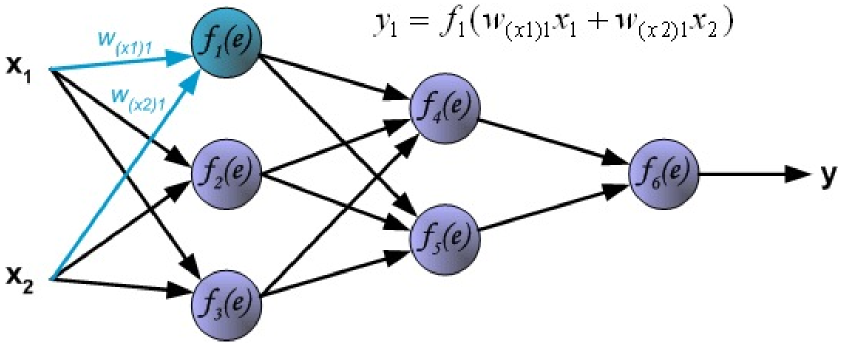
$w_{ij} = 0 || w_{ij} = w_{kl} \forall i, j, j, l \Rightarrow$ $\delta_u^{(l)}$ will be identical $\forall\ u$



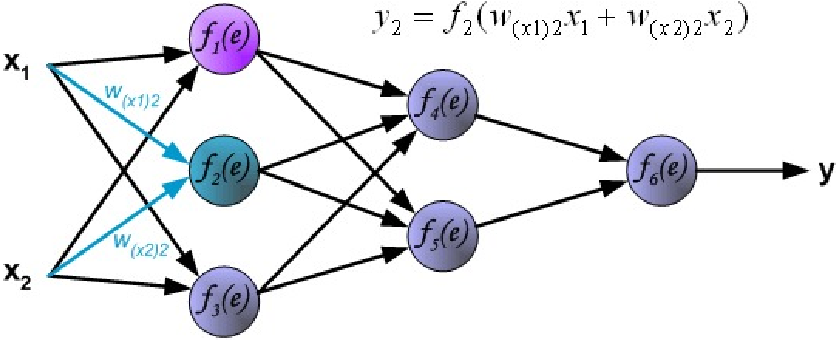
Example: Backpropagation learning



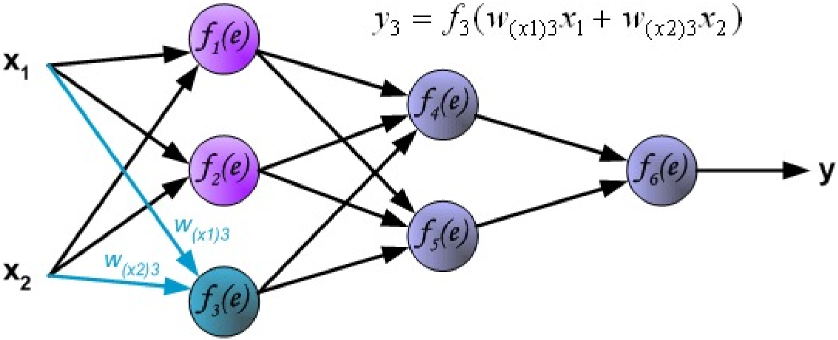
Example: Backpropagation learning



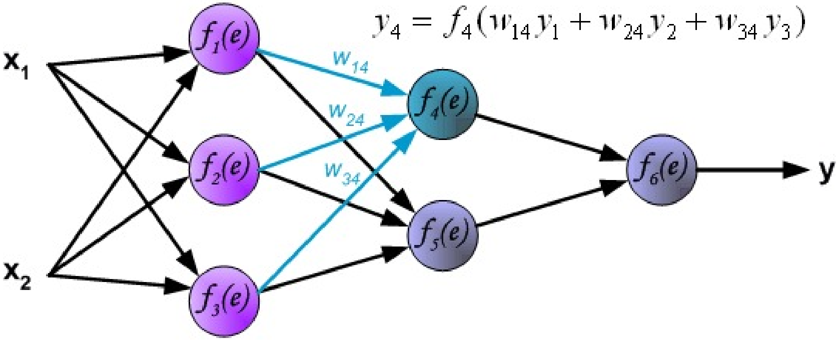
Example: Backpropagation learning



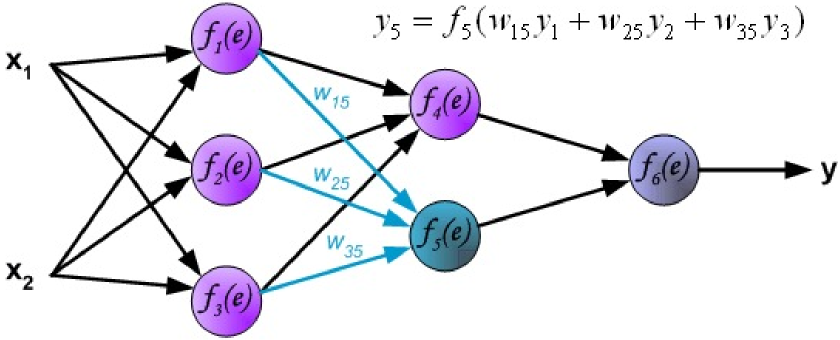
Example: Backpropagation learning



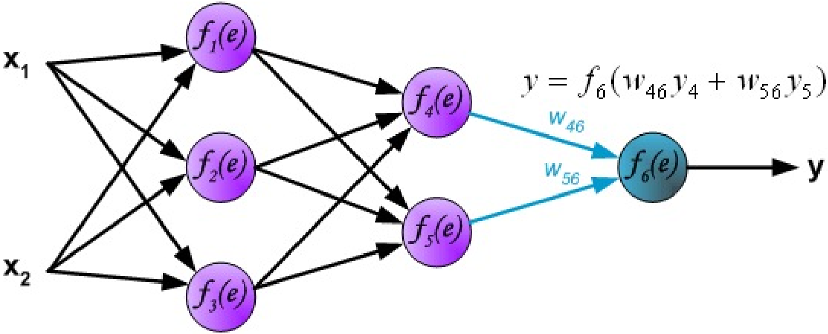
Example: Backpropagation learning



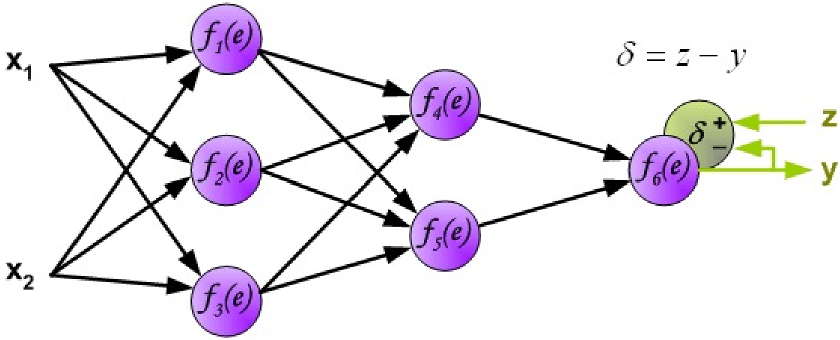
Example: Backpropagation learning



Example: Backpropagation learning

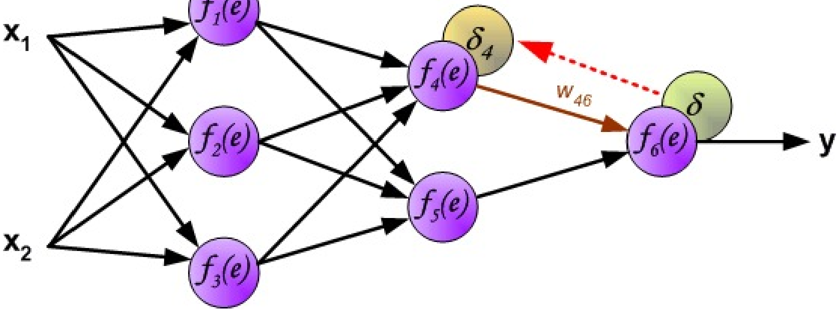


Example: Backpropagation learning

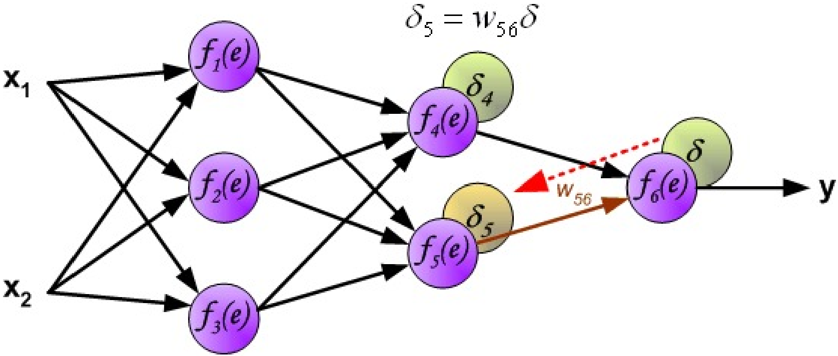


Example: Backpropagation learning

$$\delta_4 = w_{46}\delta$$

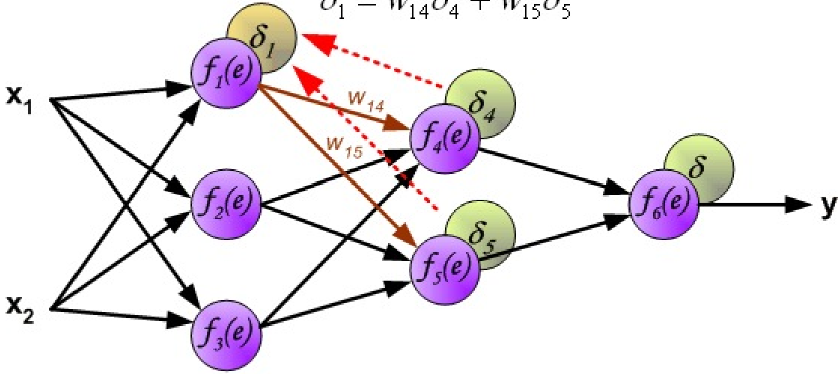


Example: Backpropagation learning

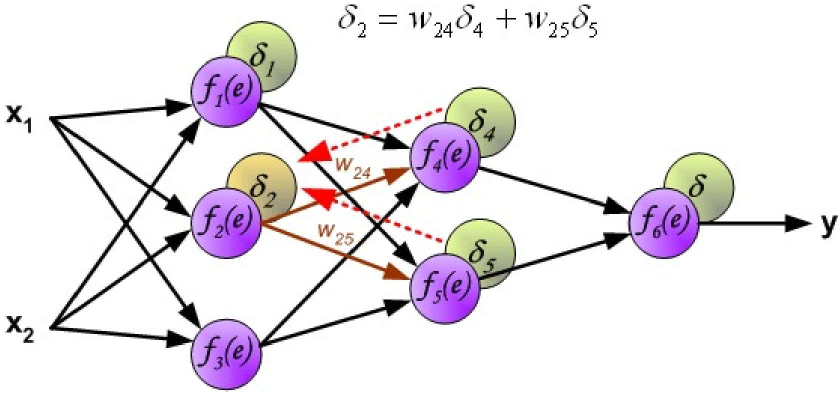


Example: Backpropagation learning

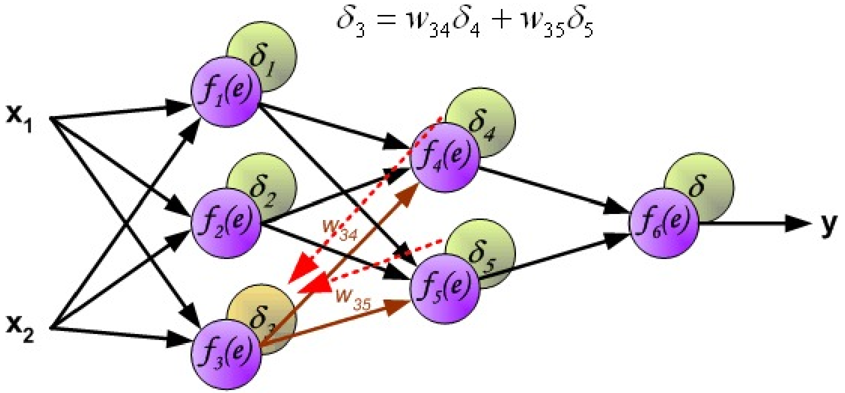
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



Example: Backpropagation learning



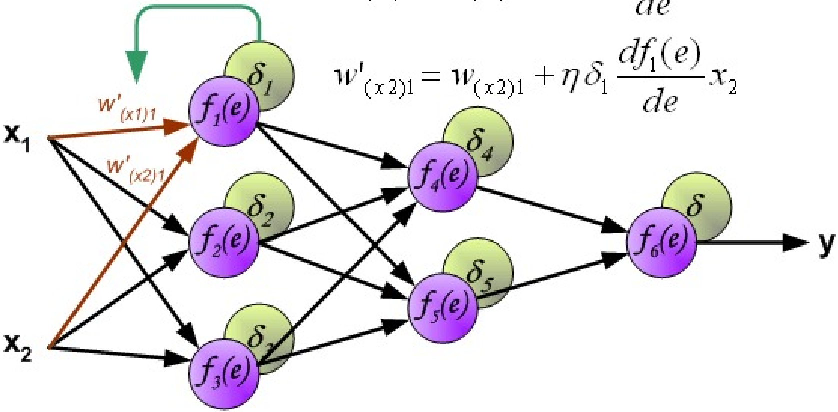
Example: Backpropagation learning



Example: Backpropagation learning

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

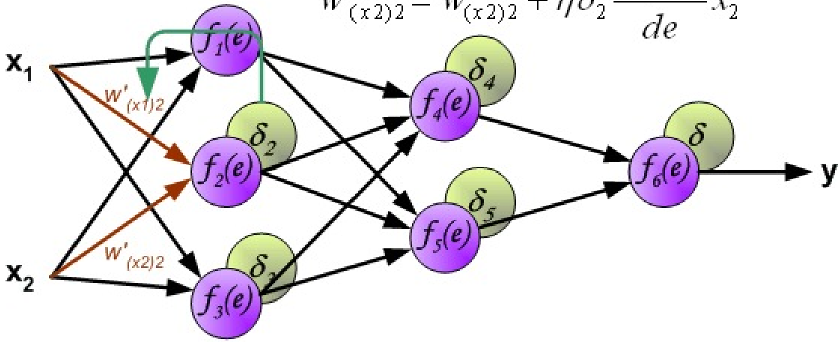
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Example: Backpropagation learning

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

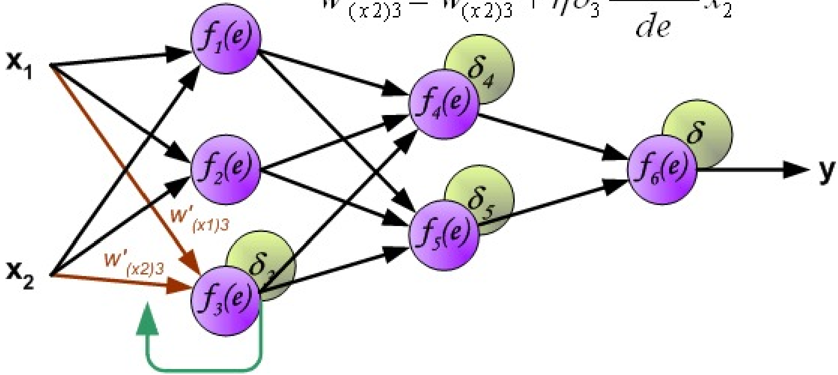
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Example: Backpropagation learning

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

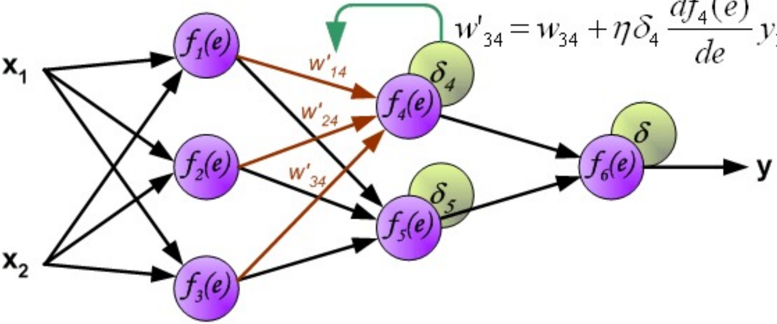


Example: Backpropagation learning

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

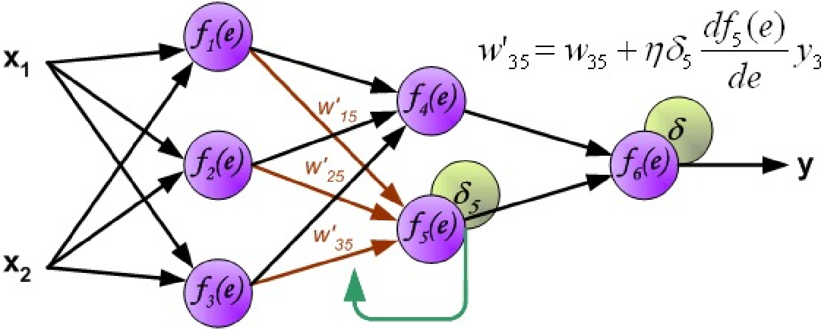


Example: Backpropagation learning

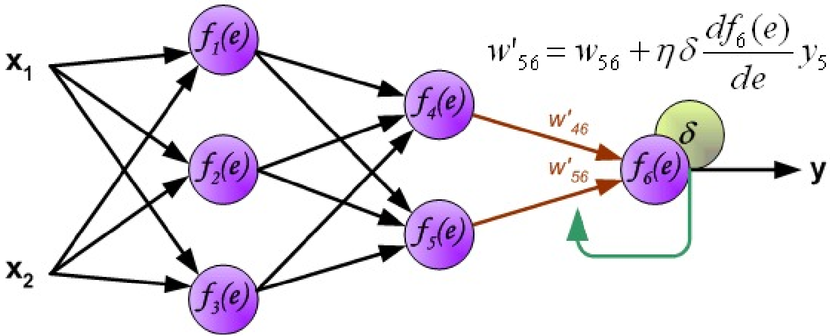
$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



Example: Backpropagation learning



$$w'_{46} = w_{46} + \eta\delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta\delta \frac{df_6(e)}{de} y_5$$

Backpropagation is an effective way of calculating the gradient of an ANN error function.

ANN error function

The ANN error function is composed from the sum of the error functions for the individual inputs:

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w})$$

$$E_n = \frac{1}{2}\sum_{k}(y_{ik} - t_{ik})^2$$

Backpropagation is an effective way of calculating the gradient of an ANN error function.

ANN error function

The ANN error function is composed from the sum of the error functions for the individual inputs:

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w})$$

$$E_n = \frac{1}{2} \sum_k (y_{ik} - t_{ik})^2$$

$\rightarrow$ In particular, it computes the gradient for each unit:

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i \rightarrow (z_i \text{ could be an input and } z_j \text{ an output})$$

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w}); \; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

In each unit, the ANN cost function computes

$z_j = h(a_j)$; with $a_j = \sum_i w_{ji} z_i \rightarrow$ (z_i could be an input and z_j an output)

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w}); \;\; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w}); \; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

Compute the derivative of E_n:

- E_n depends on w_{ij} only via the summed input a_j $\rightarrow$ chain-rule:

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w}); \;\; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

Compute the derivative of E_n:

- E_n depends on w_{ij} only via the summed input a_j $\rightarrow$ chain-rule:

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

$$\delta_j \equiv \frac{\partial E_n}{\partial w_{ji}}$$

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w}); \;\; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

Compute the derivative of E_n:

- E_n depends on w_{ij} only via the summed input a_j → chain-rule:

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

$$\delta_j \equiv \frac{\partial E_n}{\partial w_{ji}} \qquad\qquad \frac{\partial a_j}{\partial w_{ji}} = z_i$$

$$E(\overrightarrow{w}) = \sum_{i=1}^{N} E_n(\overrightarrow{w});\ \ E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j);\ \text{with}\ a_j = \sum_i w_{ji} z_i$$

Compute the derivative of E_n:

- E_n depends on w_{ij} only via the summed input $a_j \rightarrow$ chain-rule:

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \qquad \frac{\partial a_j}{\partial w_{ji}} = z_i \qquad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

$$E(\vec{w}) = \sum_{i=1}^{N} E_n(\vec{w}); \;\; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}; \quad \delta_j \equiv \frac{\partial E_n}{\partial a_j}; \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

For the output units, we have

$$\delta_k = y_k - t_k$$

$$E(\vec{w}) = \sum_{i=1}^{N} E_n(\vec{w}); \;\; E_n = \frac{1}{2}\sum_k (y_{ik} - t_{ik})^2$$

$$z_j = h(a_j); \text{ with } a_j = \sum_i w_{ji} z_i$$

$$\frac{\partial E_n}{\partial w_{ij}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}; \quad \delta_j \equiv \frac{\partial E_n}{\partial a_j}; \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

For the hidden units, use the chain rule again:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \qquad \rightarrow \delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$

Outline

Introduction

Perceptron algorithm

Neural networks
- Introduction
- Definition
- Classification
- Training Neural Networks
- Example: Backpropagation learning

Gradient calculation via backpropagation

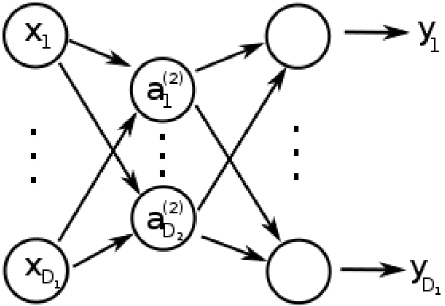
Neural Networks for dimensionality reduction

Neural Networks for dimensionality reduction

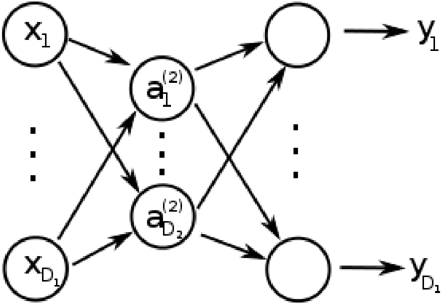
Dimensionality reduction can be achieved with a multilayer perceptron with

→ Same number $D_1 = D_L$ of inputs as outputs

→ A single hidden layer with $D_2 < D_1$ nodes



Neural Networks for dimensionality reduction

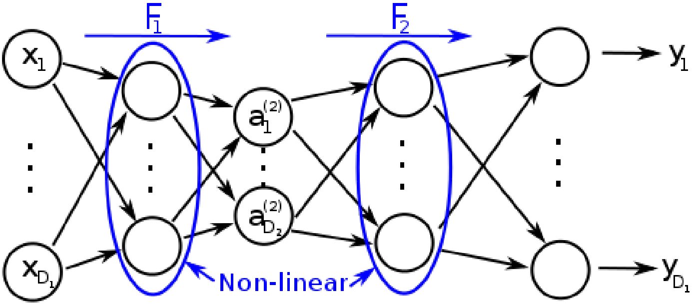


For linear activation functions, it can be shown that the error function has a global minimum

Furthermore, at this minimum, the network projects the input vectors onto the D_2-dimensional sub-space spanned by the first D_2 principal components

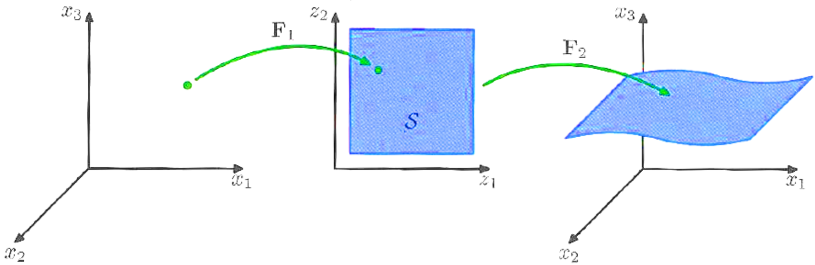
→ Linear dimensionality reduction (Same as for PCA)

Neural Networks for dimensionality reduction



With more than 2 layers and non-linear activation functions, also non-linear dimensionality reduction is possible

Neural Networks for dimensionality reduction



With more than 2 layers and non-linear activation functions, also non-linear dimensionality reduction is possible

Outline

Introduction

Perceptron algorithm

Neural networks

- Introduction
- Definition
- Classification
- Training Neural Networks
- Example: Backpropagation learning

Gradient calculation via backpropagation

Neural Networks for dimensionality reduction

Questions?

Stephan Sigg
stephan.sigg@cs.uni-goettingen.de

Literature

- C.M. Bishop: Pattern recognition and machine learning, Springer, 2007.
- R.O. Duda, P.E. Hart, D.G. Stork: Pattern Classification, Wiley, 2001.