Introduction to Neural Networks

David Stutz
david.stutz@rwth-aachen.de

Seminar Selected Topics in WS 2013/2014 – February 10, 2014

Human Language Technology and Pattern Recognition
Lehrstuhl für Informatik 6
Computer Science Department
RWTH Aachen University, Germany

Outline

1. Literature

[Bishop 06] **Pattern Recognition and Machine Learning.** ***2006*.**

- **Chapter 5 gives a short introduction to neural networks in pattern recognition.**

[Bishop 95] **Neural Networks for Pattern Recognition.** ***1995*.**

[Haykin 05] **Neural Networks A Comprehensive Foundation.** ***2005***

[Duda & Hart$^+$ 01] **Pattern Classification.** ***2001*.**

- **Chapter 6 covers mainly the same aspects as Bishop.**

[Rumelhart & Hinton$^+$ 86] **Learning Representations by Back-Propagating Errors.** ***1986***

- **Error backpropagation algorithm.**

[Rosenblatt 58] **The Perceptron: A Probabilistic Model of Information Storage and Organization in the Brain.** ***1958***

2. Motivation

Theoretically, a state-of-the-art computer is a lot faster than the human brain – comparing the number of operations per second.

Nevertheless, we consider the human brain somewhat smarter than a computer. Why?

- **Learning – The human brain learns from experience and prior knowledge to perform new tasks.**

How to specify "learning" with respect to computers?

- **Let g be an unknown *target function*.**
- **Let $T := \{(x_n, t_n \approx g(x_n)) : 1 \leq n \leq N\}$ be a set of (noisy) training data.**
- **Task: learn a good approximation of g.**

Artificial neural networks, simply *neural networks*, try to solve this problem by modeling the structure of the human brain ...

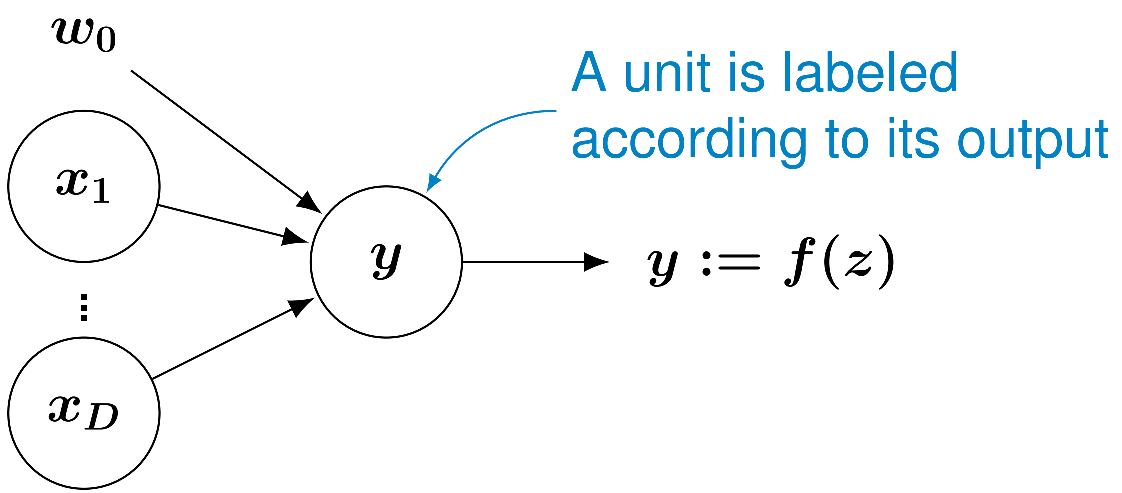
See

- **[Haykin 05] for details on how artificial neural networks model the human brain.**

3. Artificial Neural Networks – Processing Units

Core component of a neural network: *processing unit* $=$ neuron of the human brain.

A processing unit maps multiple input values onto one output value y:



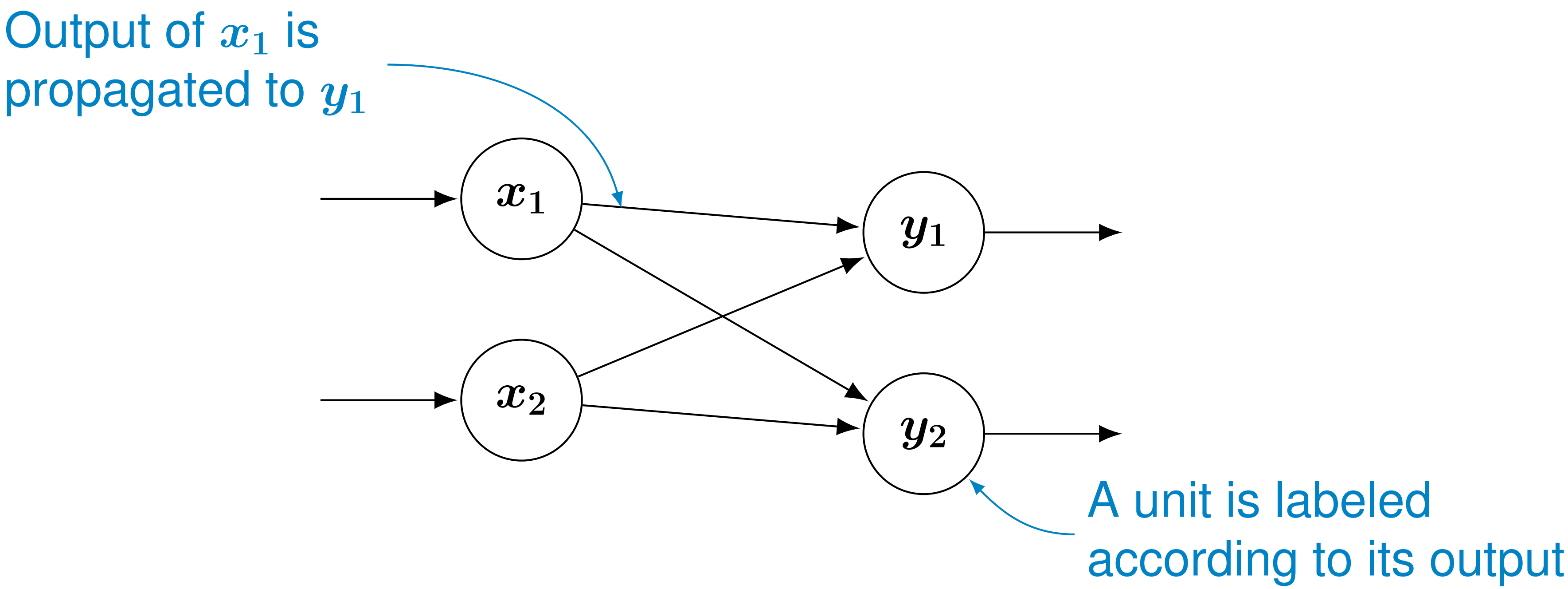
- $x_1, \ldots, x_D$ **are inputs, e.g. from other processing units within the network.**
- w_0 **is an external input called *bias*.**
- **The *propagation rule* maps all input values onto the actual input z.**
- **The *activation function* is applied to obtain $y = f(z)$.**

3. Artificial Neural Networks – Network Graphs

A neural network is a set of interconnected processing units.

We visualize a neural network by means of a *network graph*:

- **Nodes represent the processing units.**
- **Processing units are interconnected by directed edges.**



3. The Perceptron

Introduced by Rosenblatt in [Rosenblatt 58].

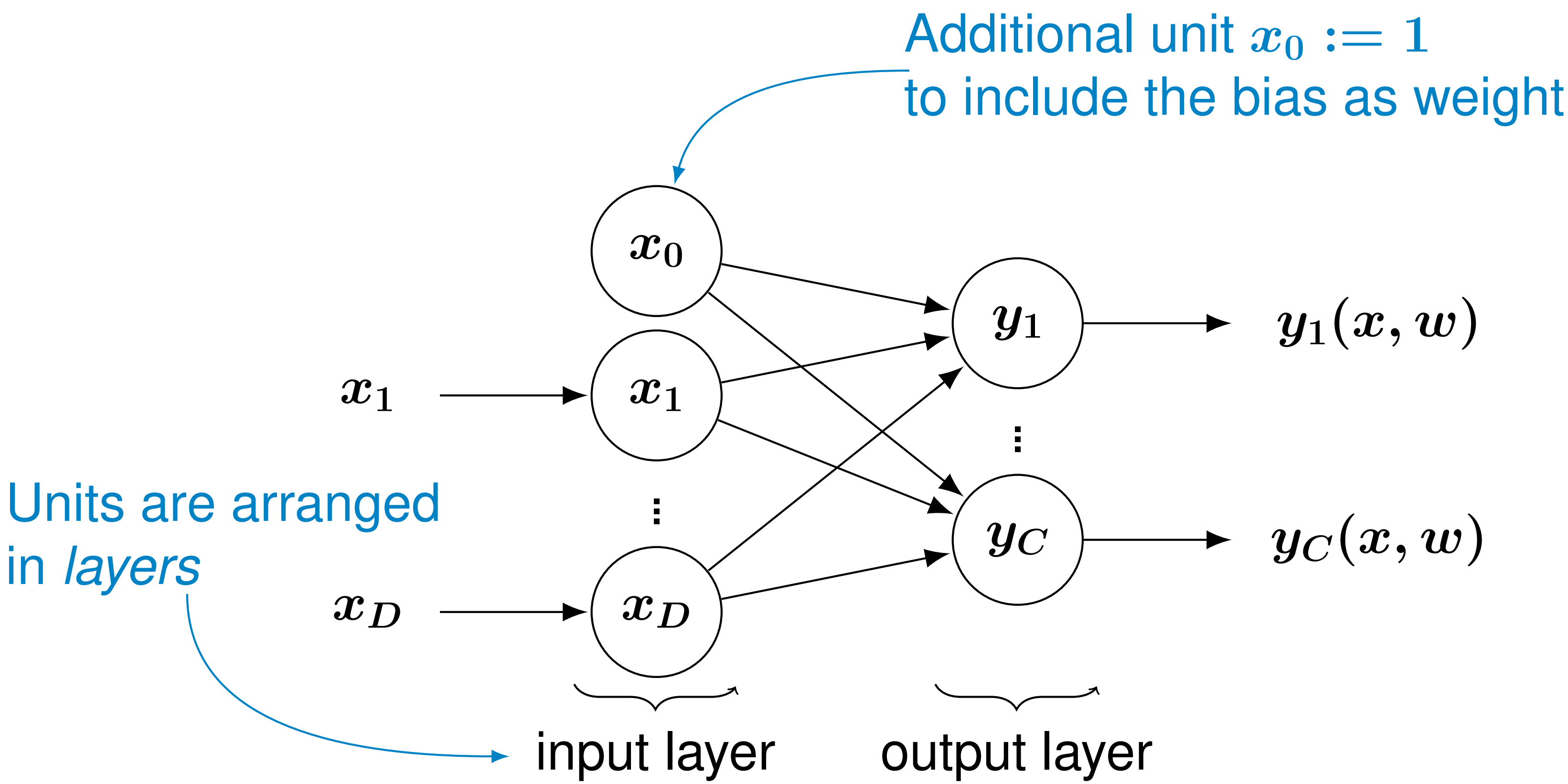
The (single-layer) *perceptron* consists of D input units and C output units.

- **Propagation rule: weighted sum over inputs x_i with weights w_{ij}.**
- **Input unit i: single input value $z = x_i$ and identity activation function.**
- **Output unit j calculates the output**

$$y_j(x, w) = f(z_j) = f\left(\sum_{k=1}^{D} w_{jk}x_k + w_{j0}\right) \stackrel{x_0:=1}{=} f\left(\sum_{k=0}^{D} w_{jk}x_k\right). \quad (1)$$

propagation rule with additional bias w_{j0}

3. The Perceptron – Network Graph



3. The Perceptron – Activation Functions

Used propagation rule: weighted sum over all inputs.

How to choose the activation function $f(z)$?

- **Heaviside function $h(z)$ models the electrical impulse of neurons in the human brain:**

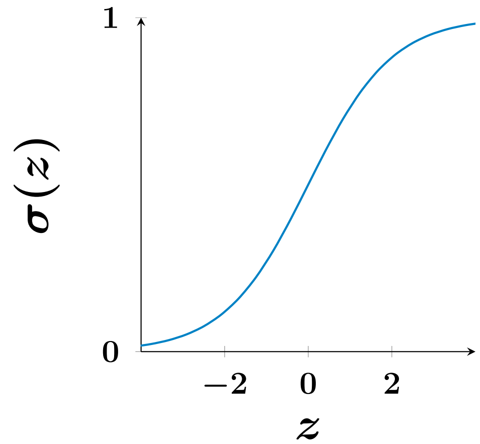
$$h(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}. \quad (2)$$

3. The Perceptron – Activation Functions

In general we prefer monotonic, differentiable activation functions.

- **Logistic sigmoid $\sigma(z)$ as differentiable version of the Heaviside function:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



- **Or its extension for multiple output units, the softmax activation function:**

$$\sigma(z, i) = \frac{\exp(z_i)}{\sum_{k=1}^{C} \exp(z_k)}. \quad (3)$$

See

- **[Bishop 95] or [Duda & Hart$^+$ 01] for more on activation functions and their properties.**

3. Multilayer Perceptrons

Idea: Add additional $L > 0$ *hidden* layers in between the input and output layer.

- **$m^{(l)}$ hidden units in layer (l) with $m^{(0)} := D$ and $m^{(L+1)} := C$.**
- **Hidden unit i in layer l calculates the output**

layer

$$y_i^{(l)} = f\left(\sum_{k=0}^{m^{(l-1)}} w_{ik} y_k^{(l-1)}\right). \quad (4)$$

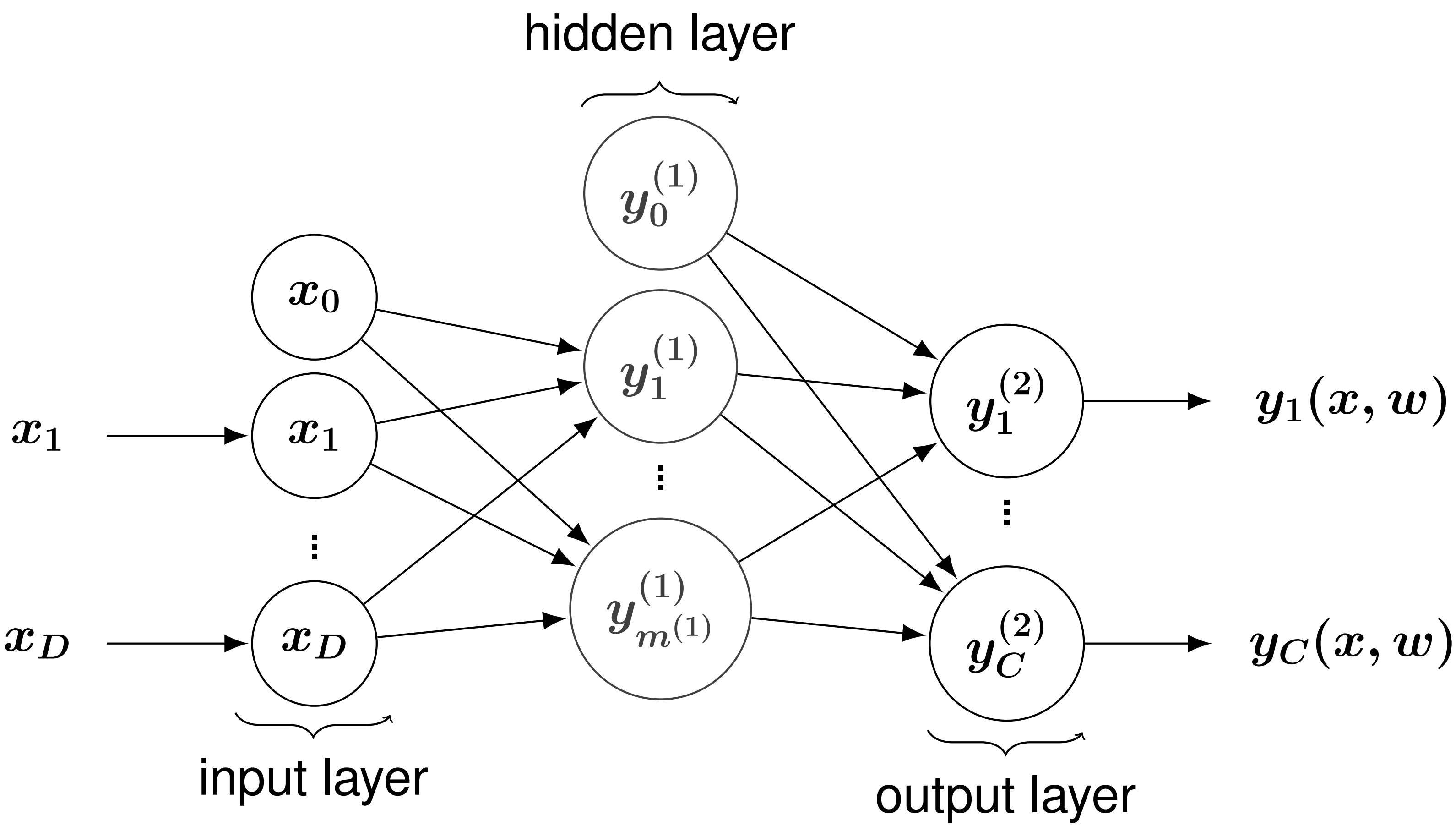
unit

A *multilayer perceptron* models a function

$$y(\cdot, w) : \mathbb{R}^D \mapsto \mathbb{R}^C, x \mapsto y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix} = \begin{pmatrix} y_1^{(L+1)} \\ \vdots \\ y_C^{(L+1)} \end{pmatrix} \quad (5)$$

where $y_i^{(L+1)}$ is the output of the i-th output unit.

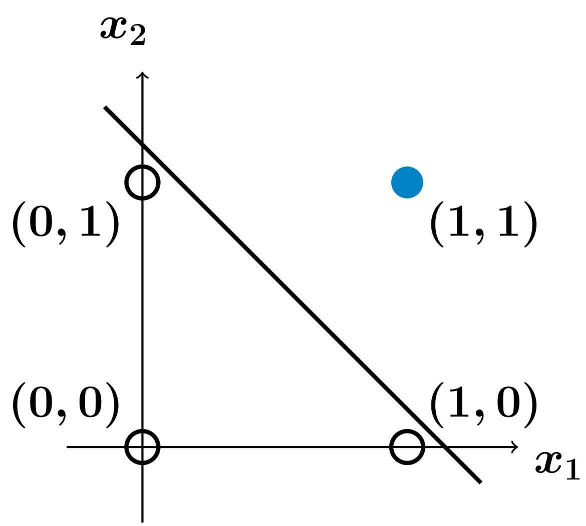
3. Two-Layer Perceptron – Network Graph



3. Expressive Power – Boolean AND

Which target functions can be modeled using a single-layer perceptron?

- **A single-layer perceptron represents a hyperplane in multidimensional space.**

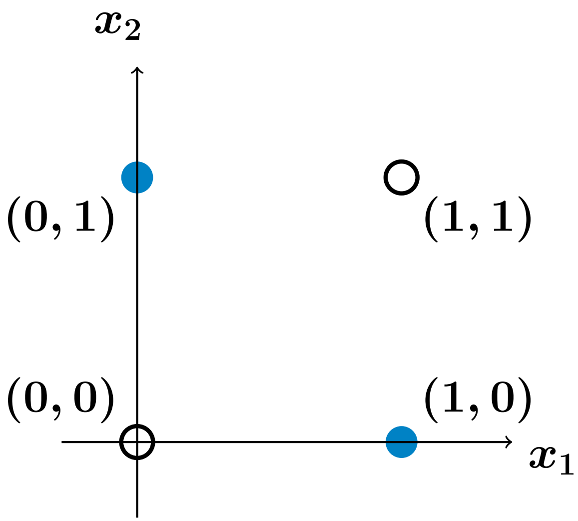


Modeling boolean AND with target function $g(x_1, x_2) \in \{0, 1\}$.

3. Expressive Power – XOR Problem

Problem: How to model boolean exclusive OR (XOR) using a line in two-dimensional space?

- **Boolean XOR cannot be modeled using a single-layer perceptron.**



Boolean exclusive OR target function.

3. Expressive Power – Conclusion

Do additional hidden layers help?

- **Yes. A multilayer perceptron with $L > 0$ additional hidden layers is a universal approximator.**

See

- **[Hornik & Stinchcombe$^+$ 89] for details on multilayer perceptrons as universal approximators.**
- **[Duda & Hart$^+$ 01] for a detailed discussion of the XOR Problem.**

4. Network Training

Training a neural network means adjusting the weights to get a good approximation of the target function.

***How* does a neural network learn?**

- ***Supervised learning*: *Training set* T provides both input values and the corresponding target values:**

input value – *pattern*

$$T := \{(x_n, t_n) : 1 \leq n \leq N\}. \tag{6}$$

target value

- **Approximation performance of the neural network can be evaluated using a distance measure between approximation and target function.**

4. Network Training – Error Measures

Sum-of-squared error function:

$$E(w) = \sum_{n=1}^{N} E_n(w) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{C}(y_k(x_n, w) - t_{nk})^2. \quad (7)$$

weight vector

k-th component of modeled function y

k-th entry of t_n

Cross-entropy error function:

$$E(w) = \sum_{n=1}^{N} E_n(w) = -\sum_{n=1}^{N}\sum_{k=1}^{C} t_{nk} \log y_k(x_n, w). \quad (8)$$

See

- **[Bishop 95] for a more detailed discussion of error measures for network training.**

4. Network Training – Training Approaches

Idea: Adjust the weights such that the error is minimized.

Stochastic training Randomly choose an input value x_n and update the weights based on the error $E_n(w)$.

Mini-batch training Process a subset $M \subseteq \{1, \ldots, N\}$ of all input values and update the weights based on the error $\sum_{n \in M} E_n(w)$.

Batch training Process all input values x_n, $1 \leq n \leq N$ and update the weights based on the overall error $E(w) = \sum_{n=1}^{N} E_n(w)$.

4. Parameter Optimization

How to minimize the error $E(w)$?

Problem: $E(w)$ can be nonlinear and may have multiple local minima.

Iterative optimization algorithms:

- **Let $w[0]$ be a starting vector for the weights.**
- **$w[t]$ is the weight vector in the t-th iteration of the optimization algorithm.**
- **In iteration $[t+1]$ choose a *weight update* $\Delta w[t]$ and set**

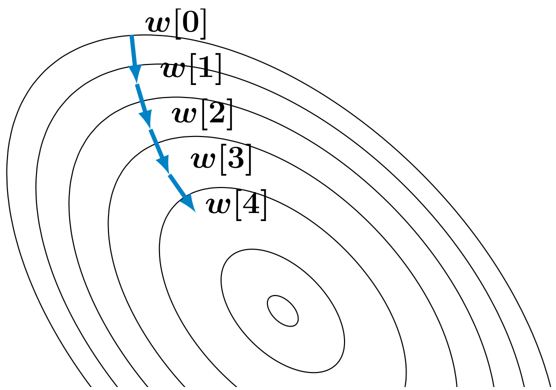
$$w[t+1] = w[t] + \Delta w[t]. \tag{9}$$

- **Different optimization algorithms choose different weight updates.**

4. Parameter Optimization – Gradient Descent

Idea: In each iteration take a step in the direction of the negative gradient.

- **The direction of the steepest descent.**



- **Weight update $\Delta w[t]$ is given by**

$$\Delta w[t] = -\gamma \frac{\partial E}{\partial w[t]}. \tag{10}$$

learning rate – *step size*

4. Parameter Optimization – Second Order Methods

Gradient descent is a simple and efficient optimization algorithm.

- **Uses first-order information of the error function E.**
- **But: often slow convergence and can get stuck in local minima.**

Second-order methods offer faster convergence:

- **Conjugate gradients,**
- **Newton's method,**
- **Quasi-Newton methods.**

See

- **[Becker & LeCun 88] for more on accelerating network training with second-order methods.**
- **[Bishop 95] for more details on parameter optimization for network training.**
- **[Gill & Murray$^+$ 81] for a general discussion of optimization.**

4. Error Backpropagation – Motivation

Summary: We want to minimize the error $E(w)$ on the training set T to get a good approximation of the target function.

Using gradient descent and stochastic learning, the weight update in iteration $[t+1]$ is given by

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}. \tag{11}$$

How to evaluate the gradient $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ of the error function with respect to the current weight vector?

Using the chain rule we can write:

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \underbrace{\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}}_{= y_j^{(l-1)}}. \tag{12}$$

4. Error Backpropagation – Step 1

Error backpropagation allows to evaluate $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ for each weight in $\mathcal{O}(W)$ where W is the total number of weights:

(1) Calculate the *errors* $\delta_i^{(L+1)}$ for the output layer:

$$\delta_i^{(L+1)} := \frac{\partial E_n}{\partial z_i^{(L+1)}} = \frac{\partial E_n}{\partial y_i^{(L+1)}} f'\left(z_i^{(L+1)}\right). \tag{13}$$

- **The output errors are often easy to calculate.**
  - **For example using the sum-of-squared error function and the identity as output activation function:**

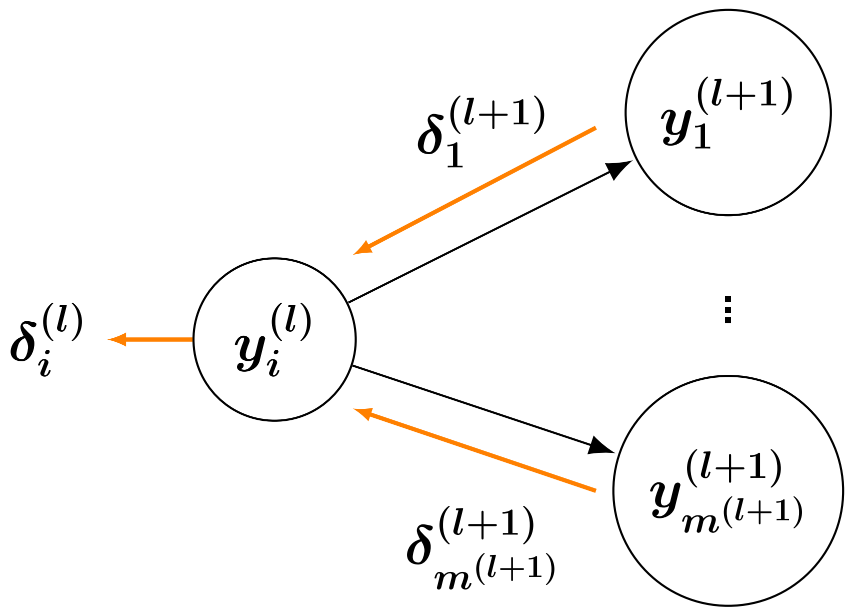
$$\delta_i^{(L+1)} = \frac{\partial \left[\frac{1}{2}\sum_{k=1}^{C}(y_k^{(L+1)} - t_{nk})^2\right]}{\partial y_i^{(L+1)}} \cdot 1 = y_i(x_n, w) - t_{ni}. \tag{14}$$

4. Error Backpropagation – Step 2

(2) Backpropagate the errors $\delta_i^{(L+1)}$ through the network using

$$\delta_i^{(l)} := \frac{\partial E_n}{\partial z_i^{(l)}} = f'\left(z_i^{(l)}\right) \sum_{k=1}^{m^{(l+1)}} w_{ik}^{(l+1)} \delta_k^{(l+1)}. \quad (15)$$

- **This can be evaluated recursively for each layer after determining the errors $\delta_i^{(L+1)}$ for the output layer.**



4. Error Backpropagation – Step 3

(3) Determine the needed derivatives using

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)}. \quad (16)$$

Now use the derivatives $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ to update the weights in each iteration.

- **In iteration step $[t+1]$ set**

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}. \quad (17)$$

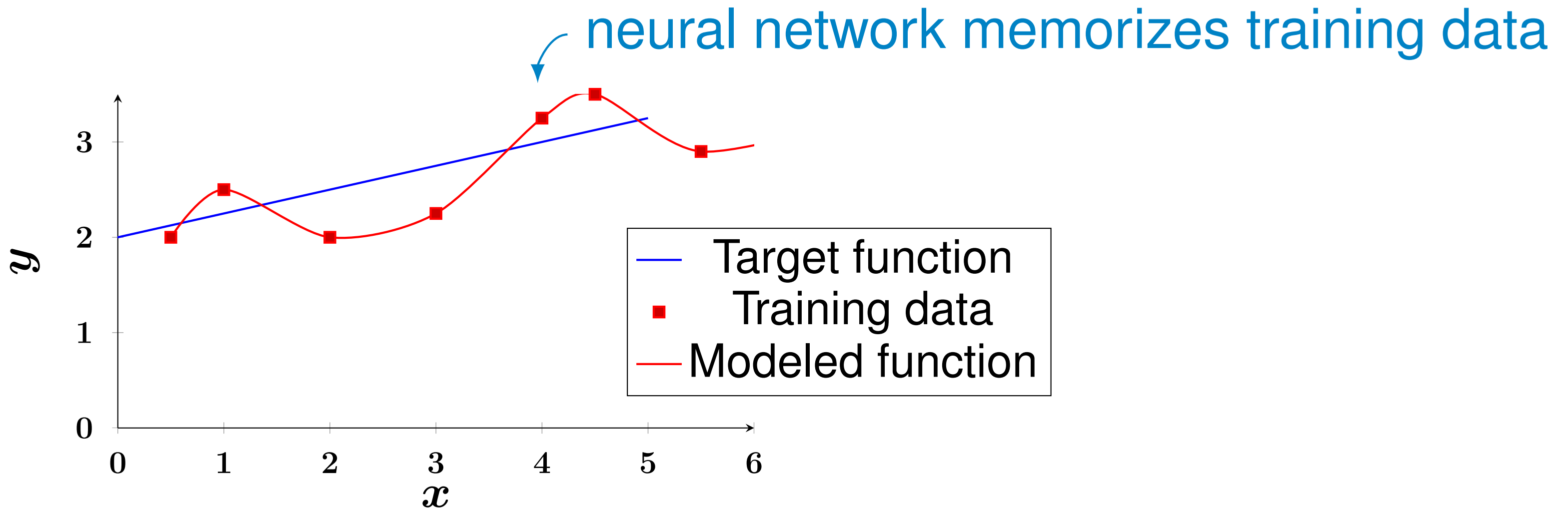
See

- **[Rumelhart & Hinton+ 86], [Duda & Hart+ 01] or [Bishop 95] for the derivation of the error backpropagation algorithm.**
- **[Bishop 92] for a similar algorithm to evaluate the Hessian of the error function.**

5. Regularization – Motivation

Recap: a multilayer perceptron is a universal approximator.

- **Given enough degrees of freedom, the network is able to memorize the training data.**
- **Memorizing the training data is also referred to as *over-fitting* and usually leads to a poor generalization performance.**



How to measure the generalization performance?

- **A network has good generalization capabilities if the trained approximation works well for unseen data – the *validation set*.**

5. Regularization

Regularization tries to avoid over-fitting.

- **Control the complexity of the neural network to avoid memorization of the training data.**

How do we control the complexity of the neural network?

- **Add a *regularizer* to the error function to influence the complexity during training:**

$$\hat{E}(w) = E(w) + \eta P(w). \tag{18}$$

See

- **[Bishop 06], [Bishop 95] or [Duda & Hart$^+$ 01] for more details on regularization.**

5. Regularization – L_2-Regularization

Observation: Large weights within the network tend to result in an approximation with poor generalization capabilities.

- **Penalize large weights using a regularizer of the form**

$$P(w) = w^T w = \|w\|_2^2. \tag{19}$$

- **Then, the weights tend exponentially to zero – therefore also called *weight decay*.**

6. Pattern Classification

Problem (Classification): Given a D-dimensional input vector x assign it to one of C discrete classes.

- **The target values t_n of the training set T can be encoded according to the 1-of-C encoding scheme:**

$$t_{nk} = 1 \quad \Leftrightarrow \quad x_n \text{ belongs to class } k. \tag{20}$$

We interpret the pattern x and the class c as random variables:

- $p(x)$ **– probability of observing the pattern x;**
- $p(c)$ **– probability of observing a pattern belonging to class c;**
- $p(c|x)$ **– *posterior probability* for class c after observing pattern x.**

the probability we are interested in

6. Pattern Classification – Bayes' Decision Rule

Assume we observed pattern x.

Assume we know the true posterior probabilities $p(c|x)$ for all $1 \leq c \leq C$.

Which class should the pattern be assigned to?

- ***Bayes' decision rule*** **minimizes the number of misclassifications:**

$$c : \mathbb{R}^D \rightarrow \{1, \ldots, C\}, x \mapsto \underset{1 \leq c \leq C}{\arg\max} \{p(c|x)\} . \quad (21)$$

assign pattern x to class c with the highest posterior probability $p(c|x)$

6. Pattern Classification – Model Distribution

Problem: The true posterior probability distribution $p(c|x)$ is unknown.

Possible solution: model the posterior probability distribution by $q_\theta(c|x)$.

Model distribution **depending on some parameters θ**
– for example the network weights $\theta = w$

- **Apply the model-based decision rule which is given by**

$$c : \mathbb{R}^D \to \{1, \ldots, C\}, x \mapsto \underset{1 \leq c \leq C}{\arg\max} \{q_\theta(c|x)\}. \quad (22)$$

6. Pattern Classification – Network Output

Idea: model the posterior probabilities $p(c|x)$ by means of the network output.

- **For example using appropriate output activation functions:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \quad \text{for two classes with one output unit such that } y(x,w) = q_\theta(c = 1|x) \text{ and } 1 - y(x,w) = q_\theta(c = 2|x); \tag{23}$$

$$\sigma(z,i) = \frac{\exp(z_i)}{\sum_{k=1}^{C} \exp(z_k)} \quad \text{for } C > 2 \text{ classes with } C \text{ output units and } y_i(x,w) = q_\theta(c = i|x). \tag{24}$$

Then: Use the training set and maximum likelihood estimation to derive error measures to train the network.

7. Conclusion

- **Artificial neural networks try to learn a specific (unknown) target function using a set of (noisy) training data.**
- **In a multilayer perceptron the processing units are arranged in layers and use the weighted sum propagation rule and arbitrary activation functions.**
- **A multilayer perceptron with at least one hidden layer is a universal approximator.**

7. Conclusion – Cont'd

- **A multilayer perceptron is trained by adjusting its weights to minimize a chosen error function on the given training data.**
  - **The error backpropagation algorithm allows to use first-order optimization algorithms.**
- **Regularization tries to avoid over-fitting to give a better generalization performance.**
  - **The generalization performance can be measured using a set of unseen data – the validation set.**
- **Pattern classification tasks can be solved by modeling the posterior probabilities by means of the network output.**
  - **Then, we can apply the model-based decision rule to classify new observations.**

Thank you for your attention

David Stutz

david.stutz@rwth-aachen.de

http://www-i6.informatik.rwth-aachen.de/

References

[Becker & LeCun 88] S. Becker, Y. LeCun: Improving the Convergence of Back-Propagation Learning with Second Order Methods. Technical report, University of Toronto, Toronto, 1988. 21

[Bishop 92] C.M. Bishop: Exact Calculation of the Hessian Matrix for the Multi-layer Perceptron. *Neural Computation*, Vol. 4, 1992. 25

[Bishop 95] C.M. Bishop: *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995. 3, 10, 17, 21, 25, 27

[Bishop 06] C.M. Bishop: *Pattern Recognition and Machine Learning*. Springer Verlag, New York, 2006. 3, 27

[Duda & Hart$^+$ 01] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification*. Wiley-Interscience Publication, New York, 2001. 3, 10, 15, 25, 27

[Gill & Murray$^+$ 81] P.E. Gill, W. Murray, M.H. Wright: *Practical optimization*. Academic Press, London, 1981. 21

[Haykin 05] S. Haykin: *Neural Networks A Comprehensive Foundation*. Pearson Education, New Delhi, 2005. 3, 4

[Hornik & Stinchcombe$^+$ 89] K. Hornik, M. Stinchcombe, H. White: Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, Vol. 2, 1989. 15

[Rosenblatt 58] F. Rosenblatt: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, Vol. 65, 1958. 3, 7

[Rumelhart & Hinton$^+$ 86] D.E. Rumelhart, G.E. Hinton, R.J. Williams: Learning Representations by Back-Propagating Errors. *Nature*, Vol. 323, 1986. 3, 25