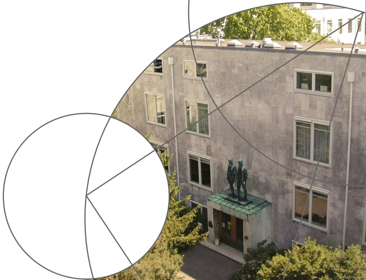
Faculty of Science

Neural Networks

Statistical Methods for Machine Learning

Christian Igel
Department of Computer Science



Outline

Warm-up: Gradient

- *Rate of change* of $f : \mathbb{R}^D \to \mathbb{R}$ at a point $\boldsymbol{x} \in \mathbb{R}^D$ when moving in the direction $\boldsymbol{u} \in \mathbb{R}^D$, $\|\boldsymbol{u}\| = 1$, is defined as:

$$\nabla_{\boldsymbol{u}} f(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{u}) - f((x))}{h}$$

- The *gradient*

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_D} \right)^{\mathsf{T}}$$

points in the direction $\nabla f(\boldsymbol{x}) / \|\nabla f(\boldsymbol{x})\|$ giving maximum rate $\|\nabla f(\boldsymbol{x})\|$ of change.

Warm-up: Chain rule

The *chain rule* for computing the derivative of a composition of two functions,

$$\frac{\partial f(g(x))}{\partial x} = f'(g(x))g'(x)$$

with $f'(x) = \frac{\partial f(x)}{\partial x}$ and $g'(x) = \frac{\partial g(x)}{\partial x}$, can be extended to:

$$\frac{\partial f(g_1(x), g_2(x), \dots, g_n(x))}{\partial x} = \sum_{i=1}^{n} \frac{\partial f(g_1(x), \dots, g_n(x))}{\partial g_i(x)} \frac{\partial g_i(x)}{\partial x}$$

Outline

Outline

What are artificial neural networks?

"There is no universally accepted definition of an NN. But perhaps most people in the field would agree that an NN is a network of many simple processors ("units"), each possibly having a small amount of local memory.The units are connected by communication channels ("connections") which usually carry numeric (as opposed to symbolic) data, encoded by any of various means.The units operate only on their local data and on the inputs they receive via the connections. The restriction to local operations is often relaxed during training. "

(Artificial) Neural Networks FAQ

Computational neuroscience vs. machine learning

Two applications of neural networks:

Computational neuroscience: Modelling biological information processing to gain insights about biological information processing

Machine learning: Deriving learning algorithms (loosely) inspired by neural information processing to solve technical problems better than other methods

Outline

Feed-forward artificial neural networks

Different classes of NNs exist:

- feed-forward NNs $\longleftrightarrow$ recurrent networks
- supervised $\longleftrightarrow$ unsupervised learning

We

- concentrate on feed-forward NNs,
- consider regression and classification,
- just consider supervised learning.

That is, we use data to adapt (train) the parameters (weights) of a mathematical model.

Simple neuron models

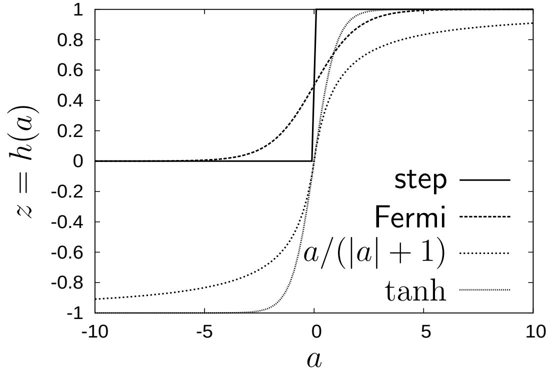
- Let the input be $x_1, \dots, x_D$ collected in the vector $\boldsymbol{x} \in \mathbb{R}^D$.
- Let the output of our neuron i be denoted by $z_i(\boldsymbol{x})$. Ofter we omit writing the dependency on $\boldsymbol{x}$ to keept the notation uncluttered.
- Integration reduces to computing a weighted sum

$$a_i = \sum_{j=1}^{D} w_{ij} x_j + w_{i0}$$

  with bias (threshold, offset) parameter $w_{i0} \in \mathbb{R}$.
- Firing is simulated by a transfer function (activation function) h:

$$z_i = h(a_i) = h\left(\sum_{j=1}^{D} w_{ij} x_j + w_{i0}\right)$$

Activation functions



Step / threshold:

$$h(a) = \Theta(a) = \mathbb{I}\{a > 0\}$$

Fermi / logistic:

$$h(a) = \frac{1}{1 + \exp(-a)}$$

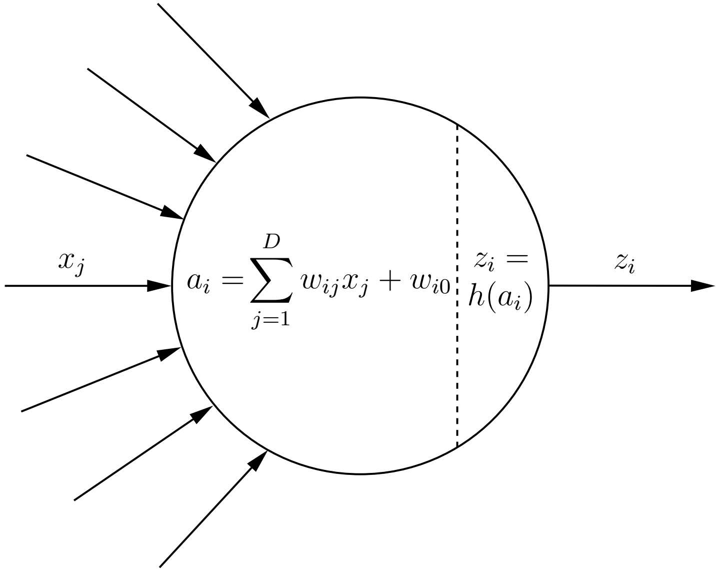
Hyperbolic tangens:

$$h(a) = \tanh(a)$$

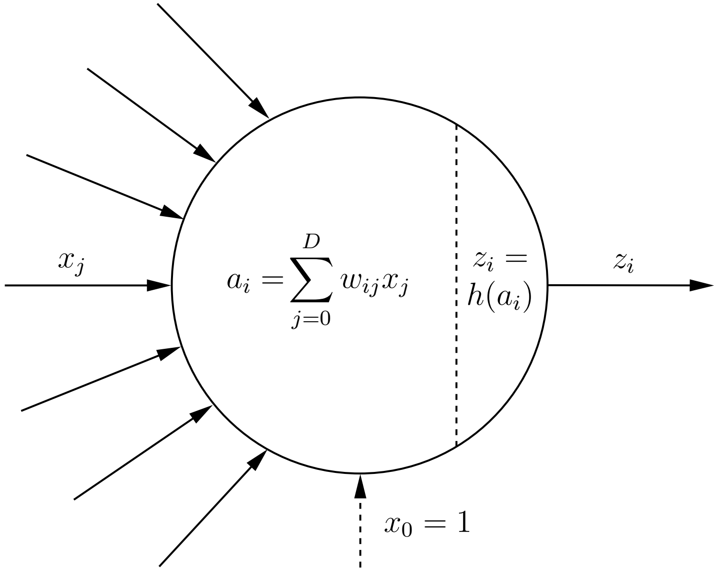
Alternative sigmoid:

$$h(a) = \frac{a}{1 + |a|}$$

Single neuron with bias

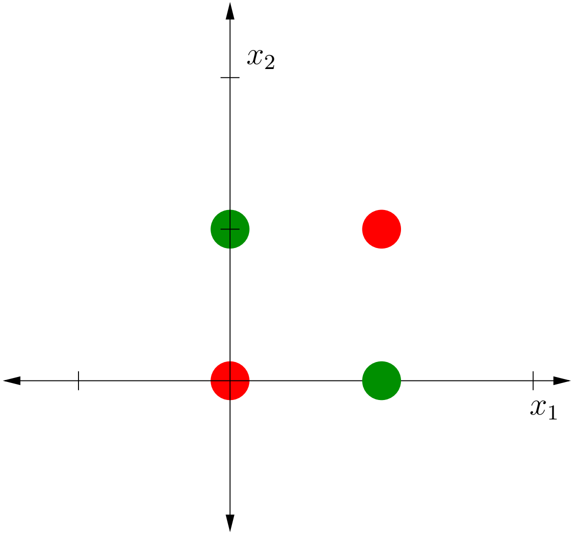


Single neuron with implicit bias



XOR

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0



Simple neural network models

- Neural network (NN): set of connected neurons
- NN can be described by a weighted directed graph
  - Neurons are the nodes
  - Connections between neurons are the edges
  - Strength of connection from neuron j to neuron i is described by weight w_{ij}
  - All weights are collected in weight vector $\boldsymbol{w}$
- Neurons are numbered by integers
- Restriction to feed-forward NNs: we do not allow cycles in the connectivity graph
- NN represents mapping

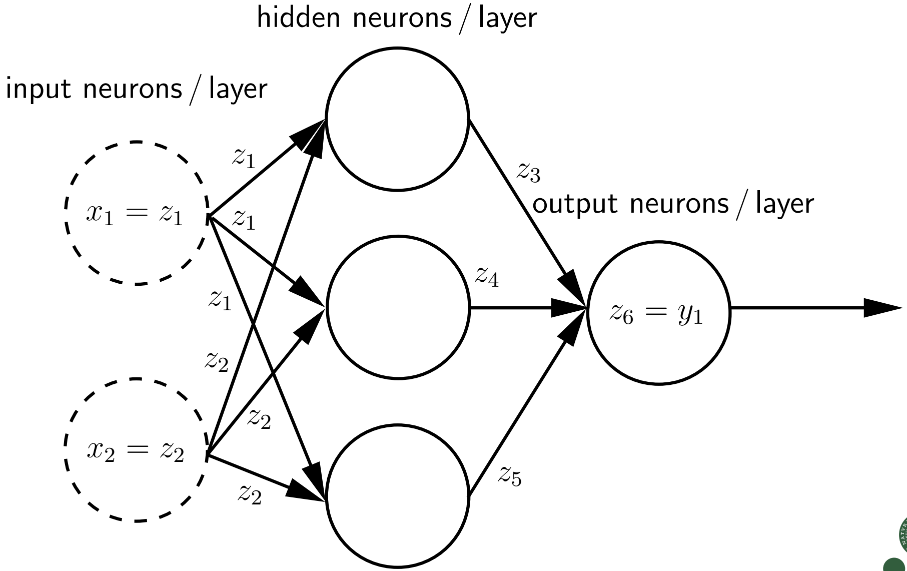
$$f : \mathbb{R}^D \to \mathbb{R}^K$$

parameterized by $\boldsymbol{w}$: $f(\boldsymbol{x}_n; \boldsymbol{w})_i = y_i$

Notation

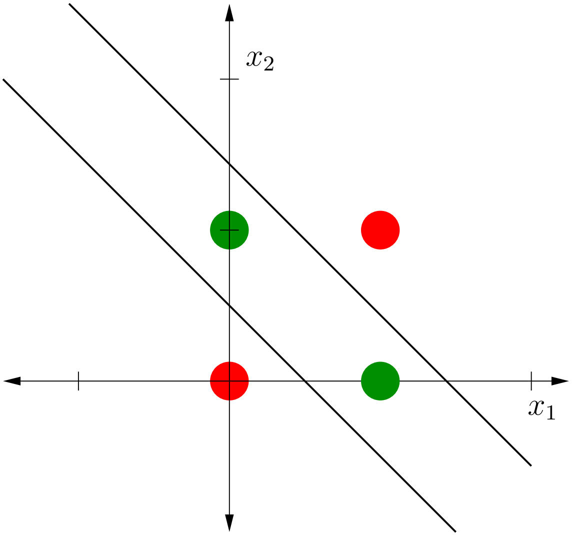
- D input neurons, K output neurons, M *hidden* neurons
- Bishop notation: activation function of hidden neurons is denoted by h, activation function of output neurons is denoted by σ
- Neuron i can get only input from neuron j if $j < i$, this ensures that the graph is acyclic
- Output of neuron i is denoted by z_i
- $z_0(\boldsymbol{x}) = 1$ ($w_{i0}z_0$ is the bias parameter of neuron i)
- $z_1(\boldsymbol{x}) = x_1, \dots, z_D(\boldsymbol{x}) = x_D$ (input neurons)
- $z_i(\boldsymbol{x}) = h\left(\sum_{0\le j<i} w_{ij}z_j\right)$ for $D < i \le D + M$
- $z_i(\boldsymbol{x}) = \sigma\left(\sum_{0\le j<i} w_{ij}z_j\right)$ for $i > D + M$ (output neurons)
- $y_1 = z_{1+M+D}(\boldsymbol{x}), \dots, y_K = z_{K+M+D}(\boldsymbol{x})$

Multi-layer perceptron network

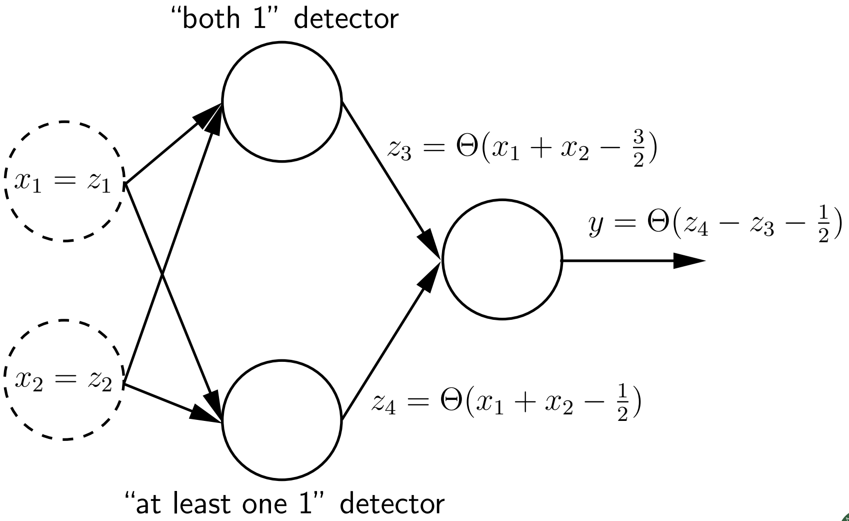


XOR revisited

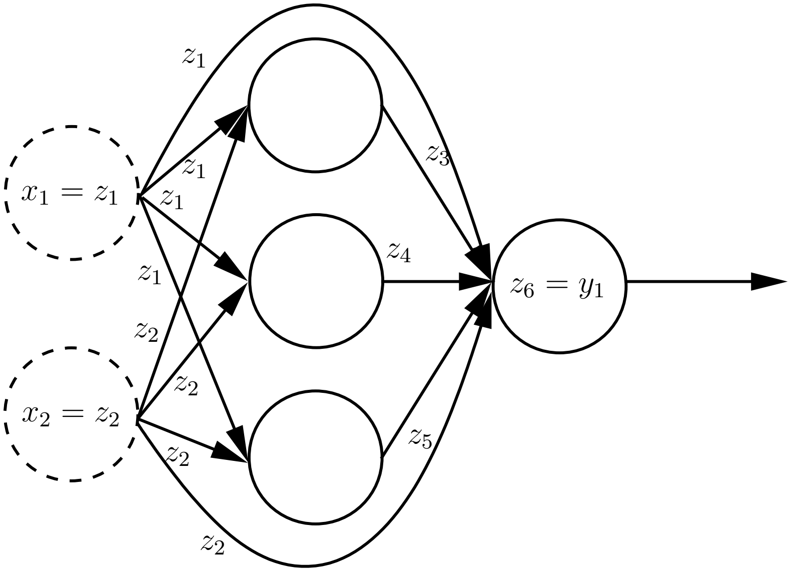
x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0



Multi-layer perceptron solving XOR


"both 1" detector
$x_1 = z_1$
$x_2 = z_2$
$z_3 = \Theta(x_1 + x_2 - \frac{3}{2})$
$y = \Theta(z_4 - z_3 - \frac{1}{2})$
$z_4 = \Theta(x_1 + x_2 - \frac{1}{2})$
"at least one 1" detector

Multi-layer perceptron network with shortcuts



Outline

Regression

- NN shall learn function

$$f : \mathbb{R}^D \to \mathbb{R}^K$$

  $\Rightarrow D$ input neurons, K output neurons

- Training data $S = \{(\boldsymbol{x}_1, \boldsymbol{t}_1), \dots, (\boldsymbol{x}_N, \boldsymbol{t}_N)\}$, $\boldsymbol{x}_i \in \mathbb{R}^D$, $\boldsymbol{t}_i \in \mathbb{R}^K$, $1 \leq i \leq N$
- Sum-of-squares error

$$E = \frac{1}{2} \sum_{n=1}^{N} \| f(\boldsymbol{x}_n; \boldsymbol{w}) - \boldsymbol{t}_n \|^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{i=1}^{K} ([f(\boldsymbol{x}_n; \boldsymbol{w})]_i - [\boldsymbol{t}_n]_i)^2$$

- Usually linear output neurons $\sigma(a) = a$

Sum-of-squares and maximum likelihood

W.l.o.g. $D = 1$, $S = \{(\boldsymbol{x}_1, t_1), \dots, (\boldsymbol{x}_N, t_N)\}$. We assume that the observations t given an input $\boldsymbol{x}$ are normally distributed (with variance s^2) around the model $f(\boldsymbol{x}; \boldsymbol{w})$:

$$p(t|\boldsymbol{x}; \boldsymbol{w}) = \frac{1}{s\sqrt{2\pi}} \exp \frac{-(t - f(\boldsymbol{x}; \boldsymbol{w}))^2}{2s^2}$$

Likelihood and negative log-likelihood:

$$p(S|\boldsymbol{w}) = \prod_{n=1}^{N} \frac{1}{s\sqrt{2\pi}} \exp \frac{-(t_n - f(\boldsymbol{x_n}; \boldsymbol{w}))^2}{2s^2}$$

$$-\ln p(S|\boldsymbol{w}) = \frac{1}{2s^2} \sum_{n=1}^{N} (t_n - f(\boldsymbol{x_n}; \boldsymbol{w}))^2 + N \ln(s\sqrt{2\pi})$$

As blue terms are independent of $\boldsymbol{w}$, minimizing the sum-of-squares error corresponds to maximum likelihood estimation under the Gaussian noise assumption.

Binary classification

For binary classification, assume $\mathcal{Y} = \{0, 1\}$, the output is in $[0, 1]$, and the target follows a Bernoulli distribution:

$$p(t|\boldsymbol{x};\boldsymbol{w}) = f(\boldsymbol{x};\boldsymbol{w})^t[1 - f(\boldsymbol{x};\boldsymbol{w})]^{1-t}$$

Negative logarithm of $p(S|\boldsymbol{w}) = \prod_{n=1}^N p(t_n|\boldsymbol{x}_n;\boldsymbol{w})$ leads to *cross-entropy* error function:

$$-\ln p(S|\boldsymbol{w}) = -\sum_{n=1}^N \{t_n \ln f(\boldsymbol{x}_n;\boldsymbol{w}) + (1-t_n)\ln(1-f(\boldsymbol{x}_n;\boldsymbol{w}))\}$$

Use sigmoid mapping to $[0, 1]$ as output activation function.

Multi-class classification: One-hot

For K classes, use one-hot encoding (1 out of K encoding):

- The jth component of $\boldsymbol{t}_i$ is one, if $\boldsymbol{x}_i$ belongs to the jth class, and zero otherwise.
- Example: If $K = 4$ and $\boldsymbol{x}_i$ belongs to third class, then $\boldsymbol{t}_i = (0, 0, 1, 0)^\mathsf{T}$.

With $\sum_{k=1}^{K} [f(\boldsymbol{x}; \boldsymbol{w})]_k = 1$ and $\forall k : [f(\boldsymbol{x}; \boldsymbol{w})]_k \geq 0$

$$p(\boldsymbol{t}|\boldsymbol{x}; \boldsymbol{w}) = \prod_{k=1}^{K} [f(\boldsymbol{x}; \boldsymbol{w})]_k^{[\boldsymbol{t}]_k}$$

gives negative log likelihood (cross-entropy for multiple classes):

$$-\ln p(S|\boldsymbol{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} [\boldsymbol{t}_n]_k \ln [f(\boldsymbol{x}_n; \boldsymbol{w})]_k$$

Multi-class classification: Soft-max

The *soft-max* activation function

$$[f(\boldsymbol{x};\boldsymbol{w})]_j = \sigma(a_{M+D+j}) = \frac{\exp a_{M+D+j}}{\sum_{k=1}^{K}\exp a_{M+D+k}}$$

naturally extends the logistic function to multiple classes and ensures that $\sum_{j=k}^{K}[f(\boldsymbol{x};\boldsymbol{w})]_k = 1$.

This extends our concept of an output layer activation function, because the output depends on the output of neurons from the same layer (i.e., lateral connections are required). Alternatively, one could consider linear output neurons and consider the normalization to be part of the error function, which the requires extension of our concept of a loss function.

Outline

Gradient descent

- Consider learning by iteratively changing the weights

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \Delta\boldsymbol{w}^{(t)}$$

- Simplest choice is (steepest) gradient descent

$$\Delta\boldsymbol{w}^{(t)} = -\eta \nabla E|_{\boldsymbol{w}^{(t)}}$$

  with learning rate $\eta > 0$

- Often a *momentum term* is added to improve the performance

$$\Delta\boldsymbol{w}^{(t)} = -\eta \nabla E|_{\boldsymbol{w}^{(t)}} + \mu \Delta\boldsymbol{w}^{(t-1)}$$

  with momentum parameter $\mu \geq 0$

Backpropagation I

Let h and σ be differentiable. From

$$z_i = h(a_i) \qquad a_i = \sum_{j<i} w_{ij} z_j$$

$$E = \sum_{n=1}^{N} E^n \qquad \text{e.g.} \qquad \sum_{n=1}^{N} \underbrace{\frac{1}{2}\|\boldsymbol{t}_n - f(\boldsymbol{x}_n \,|\, \boldsymbol{w})\|^2}_{E^n}$$

we get the partial derivatives:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial w_{ij}}$$

In the following, we derive $\frac{\partial E^n}{\partial w_{ij}}$; the index n is omitted to keep the notation uncluttered (i.e., we write E for E^n, $\boldsymbol{x}$ for $\boldsymbol{x}_n$, etc.).

Backpropagation II

We want

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}}$$

and define:

$$\delta_i := \frac{\partial E}{\partial a_i}$$

With

$$\frac{\partial a_i}{\partial w_{ij}} = z_j$$

we get:

$$\frac{\partial E}{\partial w_{ij}} = \delta_i z_j$$

Backpropagation III

For an output unit $M+D < i \leq D+M+K$ we have:

$$\delta_i = \frac{\partial E}{\partial a_i} = \frac{\partial z_i}{\partial a_i}\frac{\partial E}{\partial z_i} = \sigma'(a_i)\frac{\partial E}{\partial z_i} = \sigma'(a_i)\frac{\partial E}{\partial y_{i-M-D}}$$

If $\sigma(a) = a$, i.e., the output is linear and $\sigma'(a) = 1$, and $E = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{y}\|^2$, we get:

$$E = \frac{1}{2}\sum_{i=1}^{K}(y_i - t_i)^2 = \frac{1}{2}\sum_{i=M+D+1}^{D+M+K}(\underbrace{y_{i-M-D}}_{z_i} - t_{i-M-D})^2$$

$$\delta_i = \frac{\partial}{\partial z_i}\frac{1}{2}\sum_{j=M+D+1}^{D+M+K}(z_j - t_{j-M-D})^2 = \frac{\partial}{\partial z_i}\frac{1}{2}(z_i - t_{i-M-D})^2 \Rightarrow$$

$$\delta_i = z_i - t_{i-M-D}$$

Backpropagation IV

To get the δs for a hidden unit $i \in \{D+1, \dots, M+D\}$, we need the chain rule again

$$\delta_i = \frac{\partial E}{\partial a_i} = \sum_{k=i+1}^{M+D+K} \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_i} = \sum_{k=i+1}^{M+D+K} \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial z_i} \frac{\partial z_i}{\partial a_i}$$

and obtain:

$$\delta_i = h'(a_i) \sum_{k=i+1}^{M+D+K} w_{ki} \delta_k$$

Backpropagation V

For each training pattern $(\boldsymbol{x}, \boldsymbol{y})$:

- *Forward pass (determines output of network given $\boldsymbol{x}$):*
  1. Compute $z_D, \ldots, z_{D+M+K}$ in sequential order
  2. $z_{M-K+1}, \ldots, z_M$ define $\boldsymbol{y} = f(\boldsymbol{x} \,|\, \boldsymbol{w})$
- *Backward pass (determines partial derivatives):*
  1. After a forward pass, compute $\delta_D, \ldots, \delta_{D+M+K}$ in reverse order
  2. Compute the partial derivatives according to $\partial E / \partial w_{ij} = \delta_i z_j$

Other error functions

For an output unit $M+D<i\leq D+M+K$ we get

$$\delta_i = z_i - t_{i-M-D}$$

for

- Sum-of-squares error and linear output neurons
- Cross-entropy error and single logistic output neuron
- Cross-entropy error for multiple classes and soft-max output

Online vs. batch learning

Consider training set with N patterns and error function

$$E = \sum_{n=1}^{N} E^n \qquad \text{e.g.} \qquad \sum_{n=1}^{N} \underbrace{\frac{1}{2}(t_n - f(\boldsymbol{x}_n \,|\, \boldsymbol{w}))^2}_{E^n}$$

Batch learning: Compute the gradients over all training samples and do update

$$\Delta \boldsymbol{w}^{(t)} = -\eta \nabla E|_{\boldsymbol{w}^{(t)}}$$

Online learning: Choose a pattern $(\boldsymbol{x}_n, \boldsymbol{t}_n)$, $1 \leq n \leq N$, (e.g., randomly) and do update

$$\Delta \boldsymbol{w}^{(t)} = -\eta \nabla E^n|_{\boldsymbol{w}^{(t)}}$$

with a smaller learning rate η

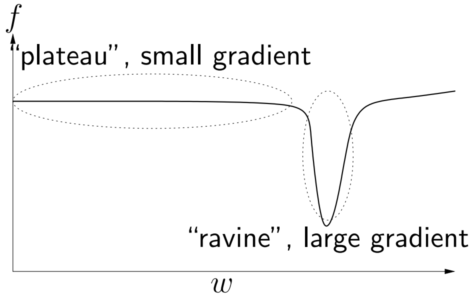
Efficient gradient-based optimization

- Vanilla steepest-descent is usually not the best choice for (batch) gradient-based learning
- Many powerful gradient-based search techniques exist
- Simple & robust & fast method: Resilient Backpropagation (RProp)

Riedmiller: Advanced supervised learning in multi-layer perceptrons – From backpropagation to adaptive learning algorithms. *Computer Standards and Interfaces*, 16(5):265–278, 1994

Igel, Hüsken: Empirical evaluation of the improved Rprop learning algorithm, *Neurocomputing*, 50(C):105–123, 2003

Resilient Backpropagation: Basic ideas



Resilient Backpropagation algorithm

Algorithm 1: Rprop algorithm

```
1 init. $\boldsymbol{w}^{(0)}$; $\forall i, j : \Delta_{ij}^{(1)} > 0$, $g_{ij}^{(0)} = 0$, $\eta^+ = 1.2$; $\eta^- = 0.5$,
  $t \leftarrow 1$
2 while stopping criterion not met do
3     $g_{ij}^{(t)} = \partial f(\boldsymbol{w}^{(t)}) / \partial w_{ij}^{(t)}$
4     foreach $w_{ij}$ do
5         if $g_{ij}^{(t-1)} \cdot g_{ij}^{(t)} > 0$ then $\Delta_{ij}^{(t)} \leftarrow \min\left(\Delta_{ij}^{(t-1)} \cdot \eta^+, \Delta_{\mathsf{max}}\right)$
6         else if $g_{ij}^{(t-1)} \cdot g_{ij}^{(t)} < 0$ then
7             $\Delta_{ij}^{(t)} \leftarrow \max\left(\Delta_{ij}^{(t-1)} \cdot \eta^-, \Delta_{\mathsf{min}}\right)$
8         $w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} - \operatorname{sign}\left(g_{ij}^{(t)}\right) \cdot \Delta_{ij}^{(t)}$
9     $t \leftarrow t + 1$
```

RProp features

- ⊕ Robust w.r.t. hyperparameters
- ⊕ Easy to implement
- ⊕ Fast
- ⊕ Independent of magnitude of partal derivatives
  - well-suited for deep architectures
  - well-suited for “noisy” gradients
- ⊖ Cannot detect correlations
- ⊖ Does not work well for online learning

Outline

Weight-decay

- The smaller the weights, the “more linear” is the neural network function.
- Thus, small $\|\boldsymbol{w}\|$ corresponds to smooth functions.
- Therefore, one can penalize large weights by optimizing

$$E + \gamma \frac{1}{2} \|\boldsymbol{w}\|^2$$

  with regularization hyperparameter $\gamma \geq 0$.
- Note: the weights of linear output neurons should not be considered when computing the norm of the weight vector.

Early stopping

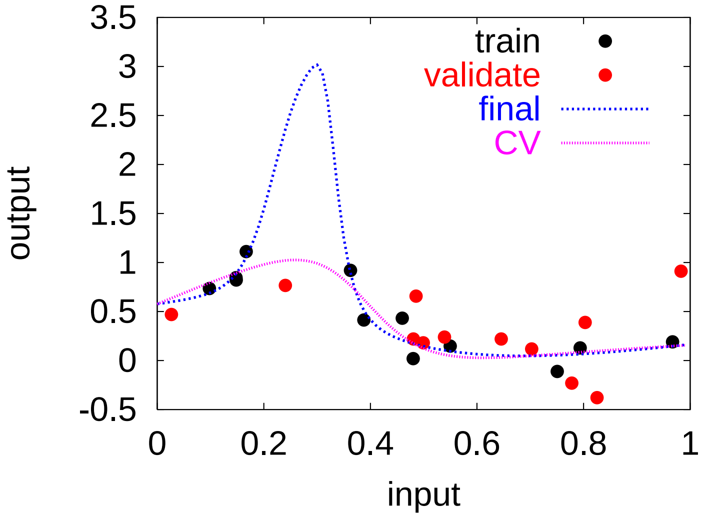
Early-stopping: the learning algorithm

- partitions sample S into training S_{train} and validation S_{val} data
- produces iteratively a sequence of hypotheses

$$h_1, h_2, h_3, \dots$$

  based on S_{train}, ideally corresponding to a nested sequence of hypothesis spaces $\mathcal{H}_1 \subseteq \mathcal{H}_2 \dots$ with $h_i \in \mathcal{H}_i$ and
  - non-decreasing complexity and
  - decreasing empirical risk $\mathcal{R}_{S_{\text{train}}}(h_i) > \mathcal{R}_{S_{\text{train}}}(h_{i+1})$ on S_{train}
- monitors empirical risk $\mathcal{R}_{S_{\text{val}}}(h_i)$ on the validation data
- outputs the hypothesis h_i minimizing $\mathcal{R}_{S_{\text{val}}}(h_i)$.

Early stopping example



Neural network architecture

- Magnitude of the weights is more important for the complexity of the model than number of neurons.
- Depth of network in general increases complexity.
- Training "deep" NNs implementing hierarchical processing is currently an active research field.

The secrets of successful shallow NN training

- Normalize the data component-wise to zero-mean and unit variance (using PCA for removing linear correlations helps)
- Use a single layer with "enough neurons"
- Start with small weights
- Employ early stopping
- Try shortcuts
- Optimization techniques relying on line search are not recommended w/o additional regularization, Rprop and steepest-descent may be preferable