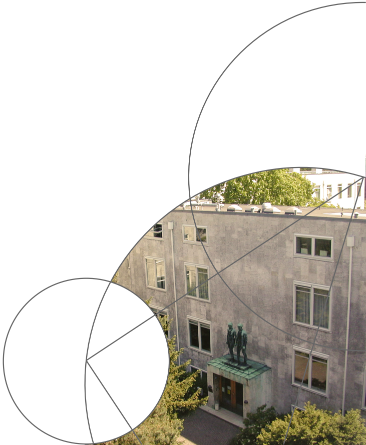
Faculty of Science

Neural Networks

Machine Learning

Christian Igel
Department of Computer Science



Outline

Warm-up: Gradient

The *gradient*

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

points in the direction of maximum rate of change.

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{e}_i) - f(\boldsymbol{x})}{\epsilon}$$

with $\boldsymbol{e}_i = (0, \dots, 0, \underbrace{1}_{i\text{-th component}}, 0, \dots, 0)^{\mathsf{T}}$

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

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

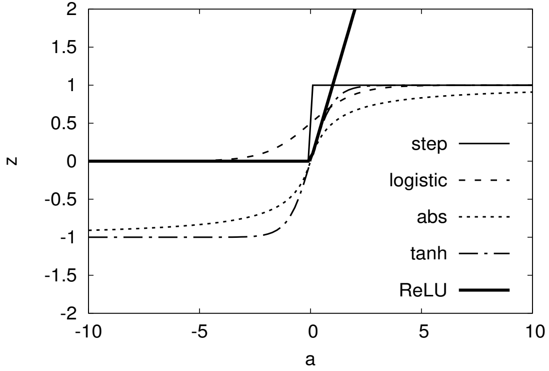
- Let the input be $x_1, \dots, x_d$ collected in the vector $\boldsymbol{x} \in \mathbb{R}^d$.
- Let the output of neuron i be denoted by $z_i(\boldsymbol{x})$. Often we omit writing the dependency on $\boldsymbol{x}$ to keept the notation uncluttered.
- "Integration": Computing weighted sum

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

  with bias (threshold, offset) parameter $w_{i0} \in \mathbb{R}$
- "Firing": Applying 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:

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

Hyperbolic tangens:

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

Rectified linear unit:

$$h(a) = \max(0, a)$$

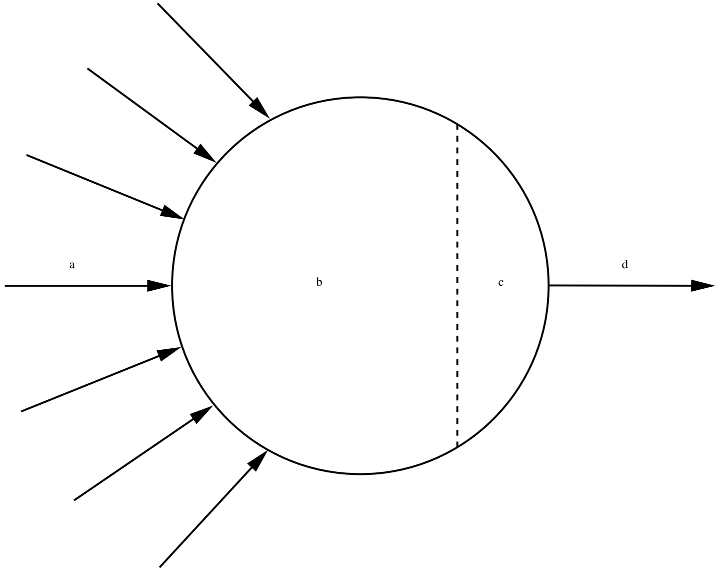
Fermi / logistic:

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

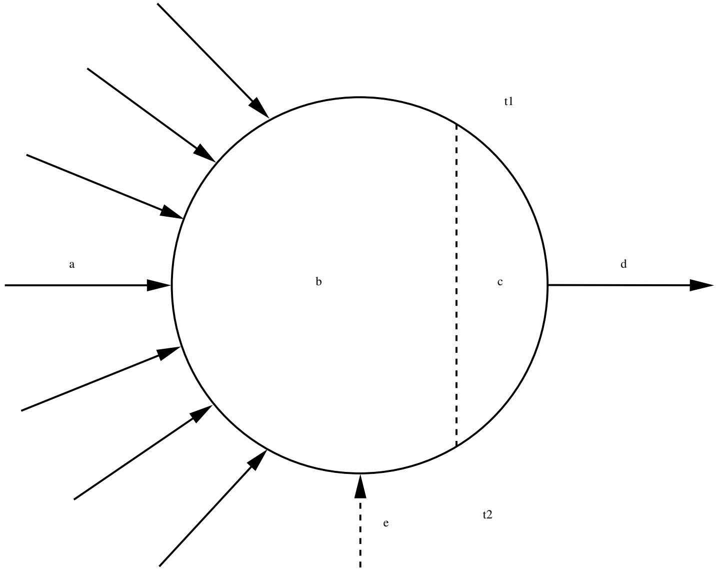
Alternative sigmoid:

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

Single neuron with bias

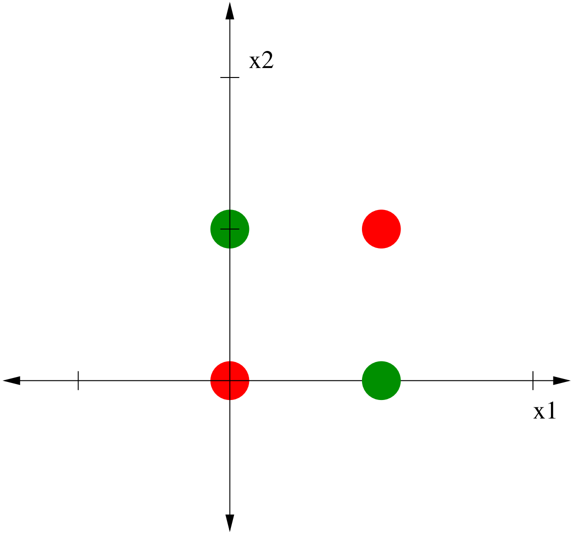


Single neuron with implicit bias



XOR

x_1	x_2	y
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 $\boldsymbol{w}$: $f(\boldsymbol{x}_n; \boldsymbol{w})_i = y_i$

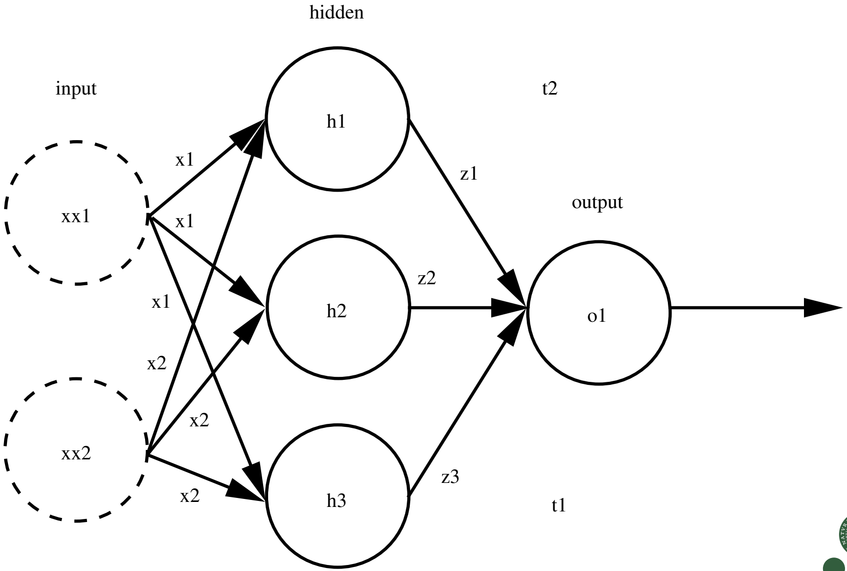
Notation I

- 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

Notation II

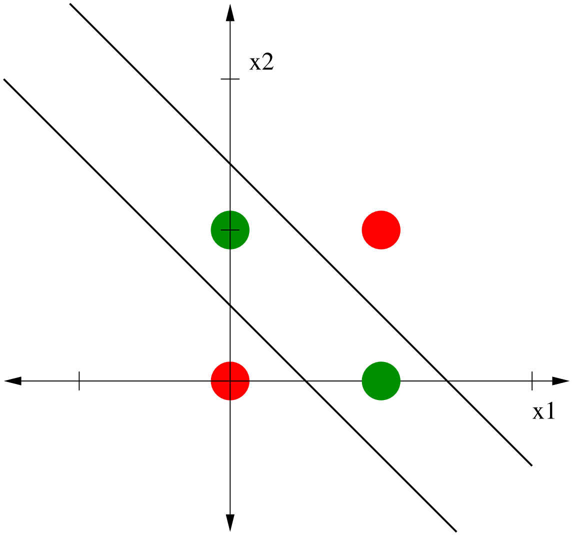
- Output of neuron i is denoted by z_i
- $z_0(\boldsymbol{x}) = 1$ ($w_{i0}z_0$ is the bias parameter of neuron i)
- $z_1(\boldsymbol{x}) = x_1, \dots, z_d(\boldsymbol{x}) = x_d$ (input neurons)
- $z_i(\boldsymbol{x}) = h\left(\sum_{0 \leq j < i} w_{ij} z_j\right)$ for $d < i \leq d + M$
- $z_i(\boldsymbol{x}) = \sigma\left(\sum_{0 \leq j < i} w_{ij} z_j\right)$ for $i > d + M$ (output neurons)
- $\hat{y}_1 = z_{1+M+d}(\boldsymbol{x}), \dots, \hat{y}_K = z_{K+M+d}(\boldsymbol{x})$

Multi-layer perceptron network

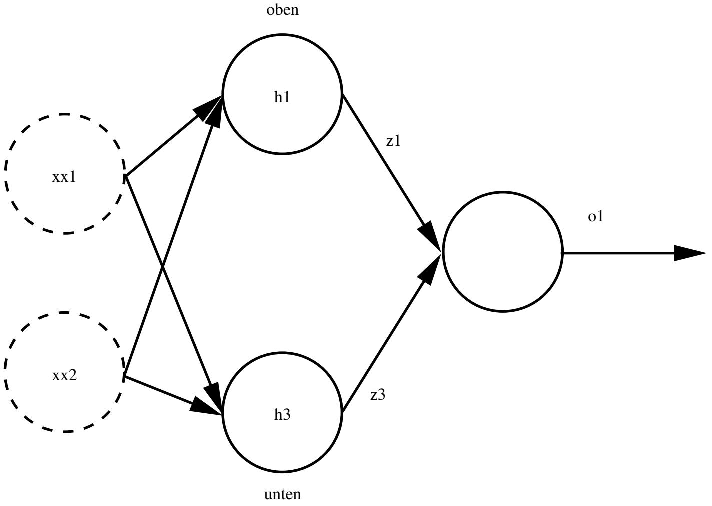


XOR revisited

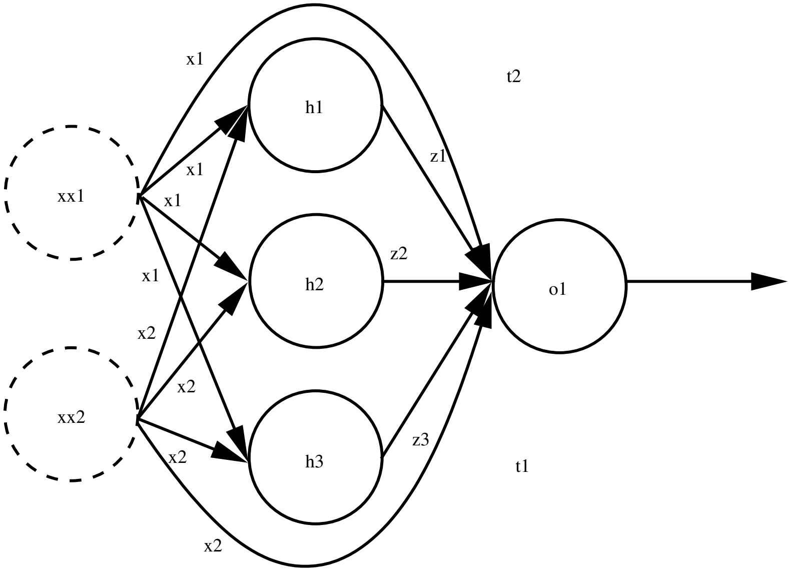
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Multi-layer perceptron solving XOR



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 $S = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, $\boldsymbol{y}_i \in \mathbb{R}^K$, $1 \le i \le N$
- Sum-of-squares error

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

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

Sum-of-squares and maximum likelihood

W.l.o.g. $d = 1$, $S = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$. We assume that the observations t given an input $\boldsymbol{x}$ are normally distributed (with variance s^2) around the model $f(\boldsymbol{x}; \boldsymbol{w})$:

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

Likelihood and negative log-likelihood:

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

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

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:

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

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

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

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 $\boldsymbol{y}_i$ 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 $\boldsymbol{y}_i = (0, 0, 1, 0)^\mathsf{T}$.

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

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

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

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

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

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

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

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:

$$\delta_i = \frac{\partial E}{\partial a_i} = \frac{\partial z_i}{\partial a_i} \frac{\partial E}{\partial z_i} = \sigma'(a_i) \frac{\partial E}{\partial z_i} = \sigma'(a_i) \frac{\partial E}{\partial \hat{y}_{i-M-d}}$$

If $\sigma(a) = a$, i.e., the output is linear and $\sigma'(a) = 1$, and $E = \frac{1}{2}\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2$, we get:

$$E = \frac{1}{2}\sum_{i=1}^{K}(\hat{y}_i - y_i)^2 = \frac{1}{2}\sum_{i=M+d+1}^{d+M+K}(\underbrace{\hat{y}_{i-M-d}}_{z_i} - y_{i-M-d})^2$$

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

$$\delta_i = z_i - y_{i-M-d}$$

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 $z_{d+1}, \ldots, z_{d+M+K}$ in sequential order
  2. $z_{M-K+1}, \ldots, z_M$ define $\hat{\boldsymbol{y}} = f(\boldsymbol{x} \,|\, \boldsymbol{w})$
- *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

$$\delta_i = z_i - y_{i-M-d}$$

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 iteration

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

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

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

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

with a smaller learning rate η

Mini-batch learning: Choose a subset $S_m = \{(\boldsymbol{x}_{i_1}, \boldsymbol{y}_{i_1}), \dots, (\boldsymbol{x}_{i_B}, \boldsymbol{y}_{i_B})\}$, $1 \leq i_1 \leq \dots \leq i_B \leq N$, and update

$$\Delta \boldsymbol{w}^{(t)} = -\eta \sum_{(\boldsymbol{x}_n, \boldsymbol{y}_n) \in S_B} \nabla E^n|_{\boldsymbol{w}^{(t)}}$$

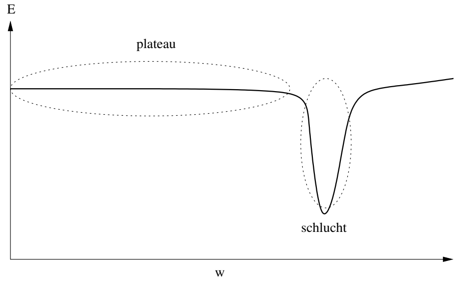
Efficient gradient-based optimization

- 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 for batch learning: Resilient Backpropagation (RProp)
- Recent method for online/min-batch learning: Adam (from "adaptive moments")

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     foreach $w_{ij}$ do
4         $g_{ij}^{(t)} = \partial f(\boldsymbol{w}^{(t)})/\partial w_{ij}^{(t)}$
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

Adam algorithm

Algorithm 2: Rprop algorithm

```
1  init. $\boldsymbol{w}^{(0)}, \beta_1, \beta_2, \alpha, \epsilon$; $t \leftarrow 1$; $\boldsymbol{v}^{(t)}, \hat{\boldsymbol{v}}^{(t)}, \boldsymbol{m}^{(t)}, \hat{\boldsymbol{m}}^{(t)} \leftarrow \mathbf{0}$
2  while stopping criterion not met do
3      foreach $w_{ij}$ do
4          $g_{ij}^{(t)} \leftarrow \partial f(\boldsymbol{w}^{(t)}) / \partial w_{ij}^{(t)}$
5          $m_{ij}^{(t+1)} \leftarrow \beta_1 \cdot m_{ij}^{(t)} + (1 - \beta_1) \cdot g_{ij}^{(t)}$
6          $v_{ij}^{(t+1)} \leftarrow \beta_2 \cdot v_{ij}^{(t)} + (1 - \beta_2) \cdot \left(g_{ij}^{(t)}\right)^2$
7          $\hat{m}_{ij}^{(t+1)} \leftarrow m_{ij}^{(t+1)} / (1 - \beta_1^t)$
8          $\hat{v}_{ij}^{(t+1)} \leftarrow v_{ij}^{(t+1)} / (1 - \beta_2^t)$
9          $w_{ij}^{(t+1)} \leftarrow w_{ij}^{(t)} - \alpha \cdot \hat{m}_{ij}^{(t+1)} / \left(\sqrt{\hat{v}_{ij}^{(t+1)}} + \epsilon\right)$
10     $t \leftarrow t + 1$
```

^{t}: power of t; $^{(t)}$: iteration step t

Adam default values

parameter	range	default	comment
β_1	$[0,1[$	0.9	first moment learning rate
β_2	$[0,1[$	0.999	second raw moment learning rate
ϵ	$\mathbb{R}^+$	10^{-8}	avoid devision by zero
α	$\mathbb{R}^+$	0.001	learning rate
			upper bound on update

- β_1 controls learning the gradient's geometric mean
- β_2 controls learning the gradient components' second raw moments
- $(1-\beta_1^t)$ and $(1-\beta_2^t)$ compensate for initialization bias
- ϵ avoids devision by zero (let's ignore it in the following)

Kingma, Lei Ba. Adam: A method for Stochastic Optimization. *ICLR*, 2015

Adam: Effects

- Update by $\alpha \cdot \hat{m}_{ij}^{(t+1)} / \sqrt{\hat{v}_{ij}^{(t+1)}}$
- Note $\sqrt{\mathbb{E}\{\left(g_{ij}^{(t)}\right)^2\}} \geq \mathbb{E}\{g_{ij}^{(t)}\}$
- α plays the role of an upper bound on the steps (can be increased by $(1-\beta_1)/\sqrt{1-\beta_2}$)
- If for all t the $g_{ij}^{(t)}$ have the same sign (i.e., the steps for that weight go in the same direction), $\sqrt{\mathbb{E}\{\left(g_{ij}^{(t)}\right)^2\}} = \mathbb{E}\{g_{ij}^{(t)}\}$
- If for a weight the $g_{ij}^{(t)}$ often change sign, $\hat{m}_{ij}^{(t+1)} / \sqrt{\hat{v}_{ij}^{(t+1)}}$ gets small

Adam: Initialization bias correction

- We have:

$$v_{ij}^{(t+1)} = (1-\beta_2)\sum_{i=1}^{t}\beta_2^{t-i}\big(g_{ij}^{(i)}\big)^2 = (1-\beta_2)\sum_{i=0}^{t-1}\beta_2^{t-i-1}\big(g_{ij}^{(i+1)}\big)^2$$

Assume $\mathbb{E}\{\big(g_{ij}^{(t)}\big)^2\}$ to be stationary and recall from geometric series that $\sum_{i=0}^{t-1}\alpha^i = (1-\alpha^t)/(1-\alpha)$:

$$\mathbb{E}\{v_{ij}^{(t+1)}\} = \mathbb{E}\Big\{(1-\beta_2)\sum_{i=0}^{t-1}\beta_2^{t-i-1}\big(g_{ij}^{(i+1)}\big)^2\Big\}$$

$$= \mathbb{E}\{\big(g_{ij}\big)^2\}(1-\beta_2)\sum_{i=0}^{t-1}\beta_2^{t-i-1}$$

$$= \mathbb{E}\{\big(g_{ij}\big)^2\}(1-\beta_2)\sum_{i=0}^{t-1}\beta_2^{i} = \mathbb{E}\{\big(g_{ij}\big)^2\}(1-\beta_2^t)$$

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

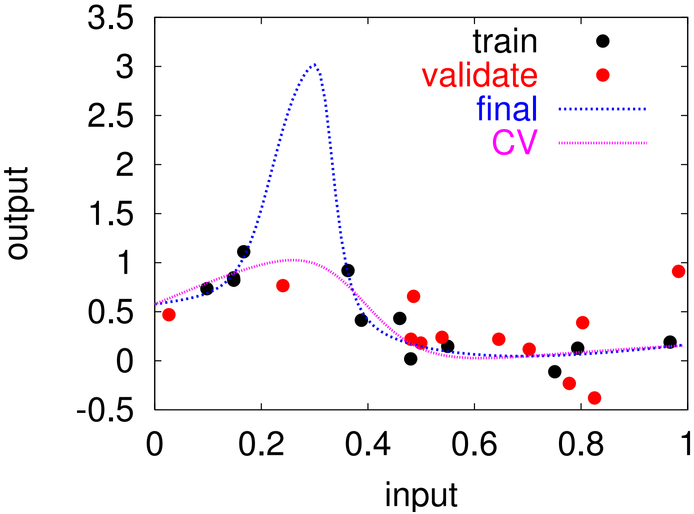
Early-stopping: the learning algorithm

- 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

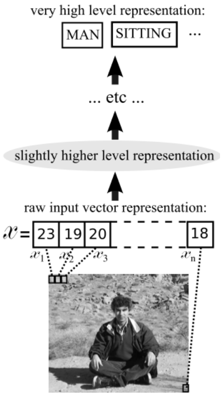
- Normalize the data component-wise to zero-mean and unit variance (using PCA for removing linear correlations helps)
- Use a single layer with “enough neurons”
- Start with small weights
- Employ early stopping
- Try shortcuts
- Optimization techniques relying on line search are not recommended w/o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

Outline

Deep learning

- Deep learning refers to ML architectures composed of multiple levels of non-linear transformations.
- Idea: Extracting more and more abstract features from input data, learning more abstract representations.
- Representations can be learned in a supervised and/or unsupervised manner.
- Example: Convolutional neural networks (CNNs) are popular deep learning architectures.

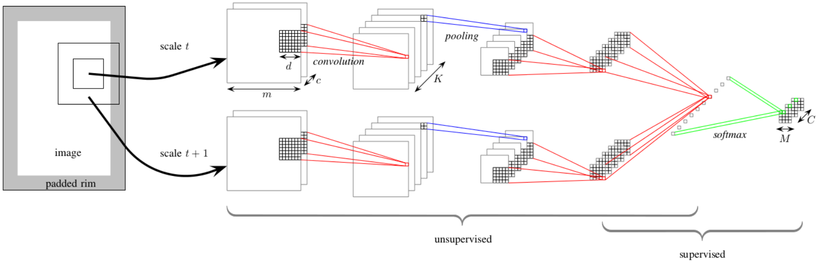
  LeCun, Bottou, Bengio, Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998


very high level representation:
MAN
SITTING
...
... etc ...
slightly higher level representation
raw input vector representation:
x = 23 19 20 18
x_1 x_2 x_3 x_n

Bengio. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* 2(1): 1–127, 2009

Convolutional neural network (CNN)

Example of a special CNN architecture we use:



A canonical CNN consists of

- convolutional layers, each of which producing several *feature maps*; interleaved with
- pooling layers; and
- a standard neural network on top.

Recall: Convolution I

Convolution with filter kernel w:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)\mathsf{d}a$$

Convolution of a discrete image I with filter kernel K:

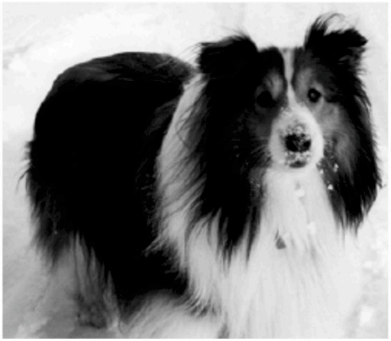
$$\begin{aligned} S(i,j) = (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m,j-n) \\ &= \sum_m \sum_n I(i-m,j-n)K(m,n) \end{aligned}$$

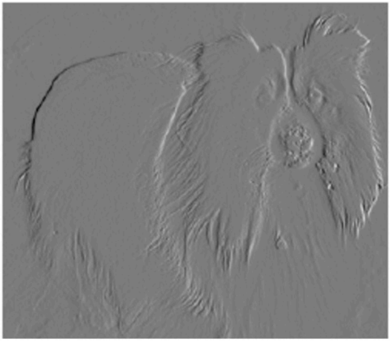
Similar to (and often mistaken with) cross-correlation:

$$S(i,j) = \sum_m \sum_n I(i+m,j+n)K(m,n)$$

Example: Convolution

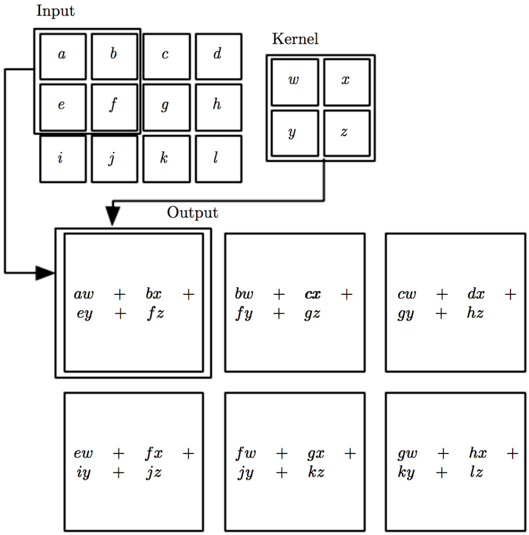
Example of applying a horizontal derivative filter $K = (1, -1)$:





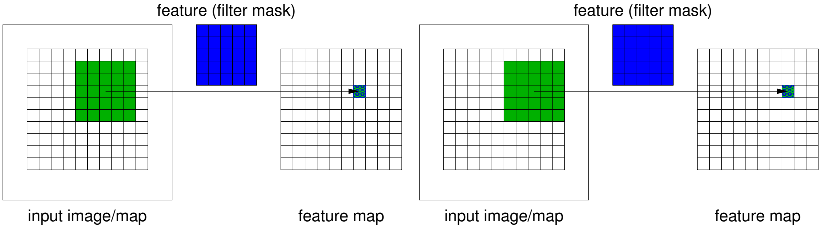
Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press

Example: Cross-correlation



Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press

Convolutional layer



- A convolutional layer computes a feature map by convolution of the input with a linear filter.
- The coefficients of each convolution kernel are learned as weights in a neural network.
- A non-linear function is applied to the feature map elements, which can be viewed as neurons with shared weights.

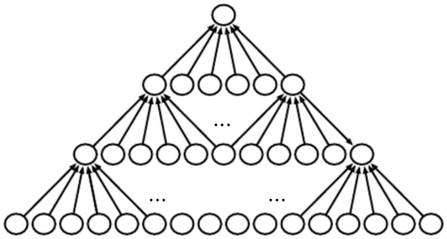
Convolutional layer: 3 ideas

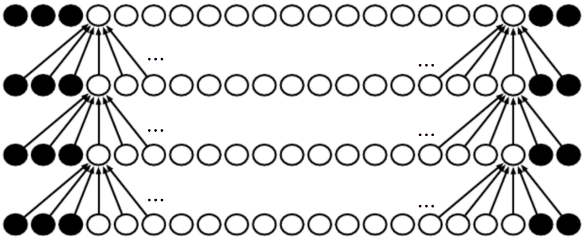
Sparse interactions: Kernel is typically smaller than input, i.e., each neuron is not fully connected; increased memory and statistical efficiency

Parameter sharing: Neurons in a layer have same weights; increased memory and statistical efficiency

Equivariance: If the input is translated (an object in an image or an event in a time series is shifted), the output is translated in the same way

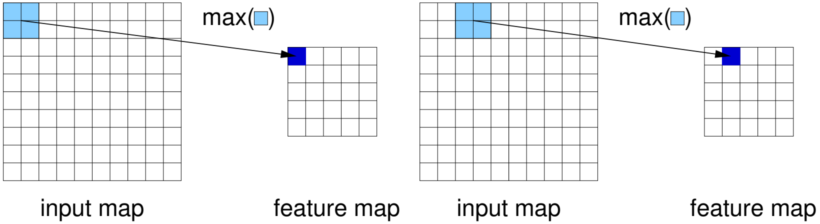
Zero padding: Dealing with the boundary





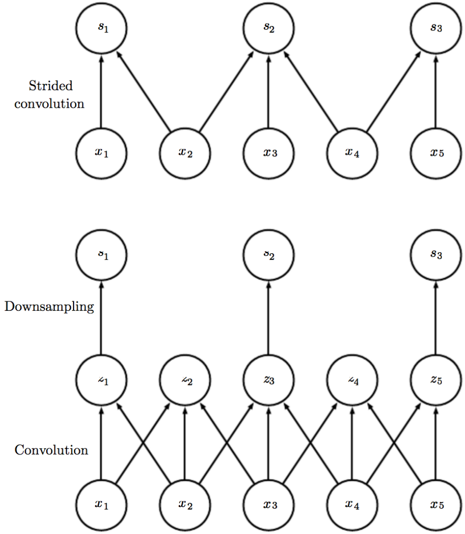
Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press

Pooling layer



- Pooling layers compute the maximum or average over a region of a feature map.
- Used to reduce the dimensionality, increase the scale, and to support translation invariance.

Convolution with stride



Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press

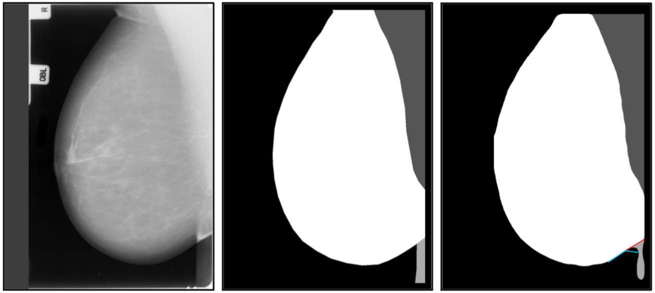
Example: Breast cancer

- Breast cancer is a frequent cause of death among women
- Screening programs halve the risk of death[1] and reduce mortality by 28-36 %[2]
- Still: 33 % of cancers are missed,[3] 70 % of referrals are false positives,[4] and 25 % of cancers could have been detected earlier[5]
- Goal: More accurate image-based biomarkers allowing personalized breast cancer screening

[1] Otto et al. *Cancer Epidemiol Biomarkers Prev* 21, 2012
[2] Broeders et al. *J Med Screen* 19, 2012
[3] Karssemeijer et al. *Radiology* 227, 2003
[4] Yankaskas et al. *Am J Roentgenol* 177, 2001
[5] Timmers et al. *Eur J Public Health*, 2012

Breast segmentation

Breast segmentation is the first step in the analysis.



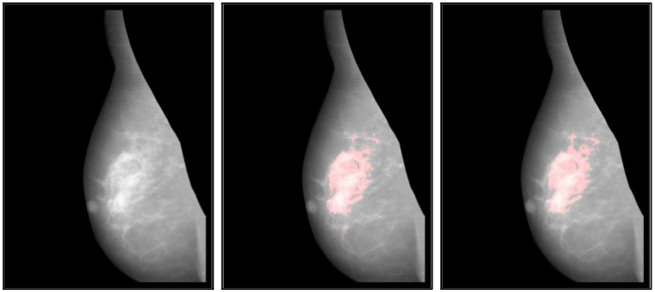
input manual CNN

Dataset	Background	Pectoral muscle	Breast tissue
Nijmegen ($N = 495$)	0.99 ± 0.01	0.95 ± 0.08	0.98 ± 0.01
Mayo ($N = 717$)	0.99 ± 0.01	0.93 ± 0.11	0.97 ± 0.02

Dice's coefficients

Breast density scoring

Breast density is related to breast cancer risk, the risk of missing breast cancer, and the risk of false positive referral.



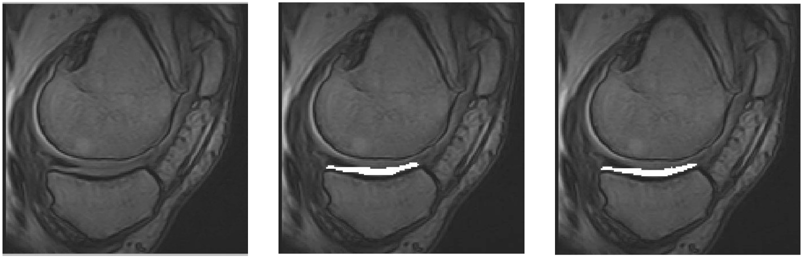
input manual CNN

Current work: Better biomarkers using CNN density scores and CNN texture scores

Kallenberg, Petersen, Nielsen, Ng, Diao, Igel, Vachon, Holland, Winkel, Karssemeijer, and Lillholm. Unsupervised deep learning applied to breast density segmentation and mammographic risk scoring. *IEEE Transactions on Medical Imaging*, 2016

Example: Knee cartilage segmentation

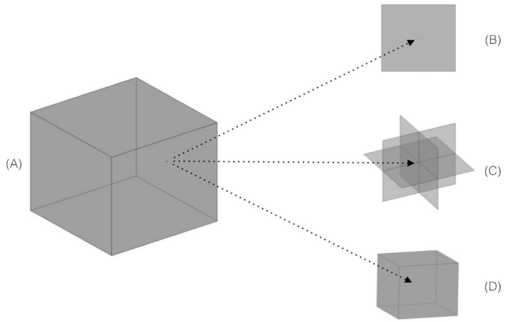
- Cartilage segmentation in knee MRI is the method of choice for quantifying cartilage deterioration.
- Cartilage deterioration implies osteoarthritis, one of the major reasons for work disability in the western world.



Prasoon, Petersen, Igel, Lauze, Dam, Nielsen. Deep Feature Learning for Knee Cartilage Segmentation Using a Triplanar Convolutional Neural Network. *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, LNCS 8150, 2013

Tri-planar CNN

Processing 3D images (A) by 2D (B), tri-planar (C), 3D (D) convolutions:



Tri-planar CNNs are computationally and statistically efficient.

Prasoon, Petersen, Igel, Lauze, Dam, Nielsen. Deep Feature Learning for Knee Cartilage Segmentation Using a Triplanar Convolutional Neural Network. *Medical Image Computing and Computer Assisted Intervention (MICCAI)*, LNCS 8150, 2013

Pai, Teng, Blair, Kallenberg, Dam, Sommer, Igel, Nielsen. Characterisation of errors in deep learning-based brain MRI segmentation. *Deep Learning for Medical Image Analysis*, Elsevier, in press

Ensemble methods

- Using an ensemble of models can avoid overfitting.
- Typically, these models are trained on different data sets
- Even trained on the same data, CNNs can be quite different because of different random initializations, different mini-batches, different hyperparameters, etc.
- Ensemble can “average out” prediction errors if these are independent of each other.
- CNN ensembles have been very successful.

Ensemble methods example

- k regression models
- For each example, model i makes error ϵ_i normally distributed around 0 with variance $\mathbb{E}[\epsilon_i^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$
- We have:

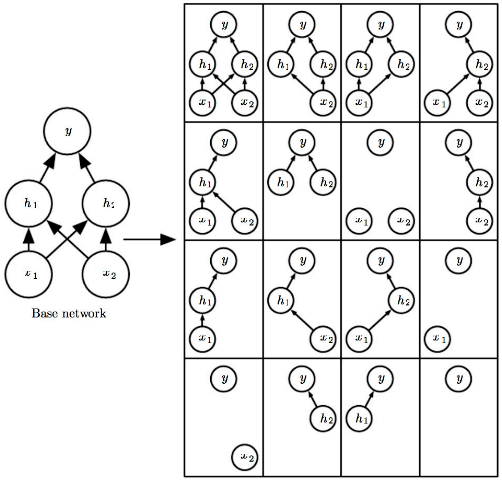
$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{j\neq i}\epsilon_i\epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

Dropout idea

- Dropout is a heuristic to avoid overfitting.
- Idea: View subnetworks of a deep neural network as members of an ensemble
- A binary mask $\boldsymbol{\mu}$ determines whether a node n is used ($\mu_n = 1$) or not ($\mu_n = 0$).

Srivastava et al. Dropout: A simple way to prevent neural networks from overtting. *JMLR*, 2014

Dropout subnetworks



Ensemble of Sub-Networks

Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press

Dropout outline

- Mini-batch training
- Mask $\boldsymbol{\mu}$ resampled after every batch
- Probability of hidden neuron to be include typically 0.5, for an input 0.8
- Learning in each step as normal (ignoring weights associated with nodes having mask value 0)

Dropout ensemble inference

- Output of subnetwork defined by mask $\boldsymbol{\mu}$ given input $\boldsymbol{x}$ is by $p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\mu})$.
- For d units that may be dropped, the full ensemble gives:

$$2^{-d}\sum_{\boldsymbol{\mu}} p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\mu})$$

- In practice, the sum above is approximated by sampling some masks $\boldsymbol{\mu}$.

Dropout weight scaling

- Weight scaling inference rule is an efficient alternative to dropout ensemble inference.
- Only the full model is evaluated, however, all weights are multiplied with the probability of the corresponding weight being included in the ensemble (i.e., typically divided by 2).
- This heuristic is exact, for example, if
  - the networks are combined by the geometric (instead of arithmetic) mean, and
  - there are no non-linear hidden units.

Hinton et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv:1207.0580*

Some thoughts on CNNs

- CNNs are well suited for image and sound processing; they have won several competitions recently.
  The best performance is usually achieved when the CNN is combined with careful image preprocessing.
- CNNs exploit "spatial" structure in the input; in particular, they incorporate translation invariance.
- Unsupervised and supervised learning can be combined.
- Training CNNs typically scales linearly in training set size.
- CNNs profit from massively parallel computing (e.g., GPUs)

Some more thoughts

- $\oplus/\ominus$ Deep networks are highly non-linear models.
- $\ominus$ Deep networks typically have many hyperparameters.
- $\ominus$ Tuning the architecture of a deep learning system can be very difficult.
- $\ominus$ Deep learning is badly understood theoretically.

The revival of deep neural networks is partly due to the availability of fast (parallel) hardware and larger training data sets (which makes overfitting less of a problem).

Some deep learning resources

Goodfellow, Bengio, Courville. *Deep Learning*. MIT Press, to appear, `http://www.deeplearningbook.org/`

Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural Networks* 61:85–117, 2015