# Inhaltsverzeichnis


**1 Introduction** **2**
- 1.1 What is machine learning . . . 2
- 1.2 What is deep learning? . . . 4
  - 1.2.1 What is a neural network (NN) . . . 4
- 1.3 Examples for Deep Learning . . . 4

**2 Tools for Deep Learning** **6**
- 2.1 Software . . . 6
- 2.2 Hardware . . . 6
- 2.3 Datasets . . . 6

**3 Machine learning basics** **6**
- 3.1 Linear Algebra . . . 6
- 3.2 Random variable and probability distribution . . . 8
  - 3.2.1 One random vector . . . 8
  - 3.2.2 Multiple random vectors . . . 10
  - 3.2.3 Kernel based density estimation . . . 10
- 3.3 Kullback-Leibler divergence and cross entropy . . . 12
  - 3.3.1 E3.5 KLD between normal and Laplace distribution . . . 14
- 3.4 Probabilistic framework for machine learning . . . 14
  - 3.4.1 Role of a NN . . . 18

**4 Dense Neural Networks** **18**
- 4.1 Fully connected neural networks - Neuron . . . 18
- 4.2 Chapter 4.2 Layer of Nurons . . . 20
- 4.3 Feedforward neural network . . . 20
- 4.4 Activation function . . . 22
  - 4.4.1 Sigmoid activation function . . . 22
  - 4.4.2 hyperbolic tangent activation function . . . 24
  - 4.4.3 rectifier linear unit(ReLU . . . 24
  - 4.4.4 Softmax activatoin function(classification problem) . . . 24
  - 4.4.5 Special case c=2, binary classification problem . . . 26





- 4.5 Universal approximation . . . 26
  - 4.5.1 E4.3 Regression with 1 hidden layer . . . 26
- 4.6 Loss and cost function . . . 28
  - 4.6.1 Regression Problem . . . 28
  - 4.6.2 Classification . . . 28
  - 4.6.3 Semantic segmentation . . . 30
- 4.7 Training . . . 30
  - 4.7.1 Chainrule of derivative (back propagation) . . . 32
- 4.8 4.8 Implementation of DNN's in Python . . . 34

**5 Advanced optimization techniques** **34**
- 5.1 Difficulties in optimization . . . 34
  - 5.1.1 E5.1: sigmoid vs. ReLU . . . 36
- 5.2 Momentum method . . . 38
  - 5.2.1 Nesterov Momentum . . . 38
- 5.3 5.3 Learning rate schedule . . . 38
- 5.4 Input and batch normalization . . . 38
  - 5.4.1 E5.3 A Perceptron . . . 38
  - 5.4.2 Batch normalization . . . 40
- 5.5 Parameter initialization . . . 42
- 5.6 Improved (network)-model . . . 42

**6 Overfitting and regularization** **44**
- 6.1 Model capacity and overfitting / underfitting . . . 44
- 6.2 Weight norm penalty . . . 44
- 6.3 Early stopping . . . 44
- 6.4 Data augmentation . . . 44
- 6.5 Ensamble learning . . . 46
- 6.6 Dropout . . . 46
- 6.7 Hyperparameter optimization . . . 48

**7 Convolutional neural networks (CNN)** **50**
- 7.1 Convlutional layer . . . 50
  - 7.1.1 Properties of convolutional layer . . . 52
- 7.2 Modified convolutions . . . 54
- 7.3 Pooling and unpooling layer . . . 54
- 7.4 Deconvolutional layer . . . 56
- 7.5 Flatten layer . . . 56
- 7.6 Global average pooling layer . . . 56
- 7.7 Architecture of CNNs . . . 56

**8 Reccurent/recursive neural networks (RNN)** **58**

8.1 Recruent layer and recurrent neural network . . . . . 58
8.2 Bididirectional recurrent neural network(BRNN) . . . . . 60
8.3 Long short-term memory (LSTM) . . . . . 62

**9 Unsupervised and generative models 64**
9.1 Autoencoder(AE) . . . . . 64
9.2 Variational Autoencoder . . . . . 66



# Chapter 1: Introduction

Date: 03/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

•p.4 expert talks are not relevant for the exam

- **scalar**: $x, A, \alpha$
- column **vector**: $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = [x_i] \in \mathbb{R}^N$ with element $x_i = [\underline{x}]_i$
- **matrix**: $\mathbf{A} = [a_{ij}]_{1 \le i \le M, 1 \le j \le N} = [a_{ij}] \in \mathbb{R}^{M \times N}$ with element $a_{ij} = [\mathbf{A}]_{ij}$
- 3D, 4D **tensor**: $\mathbf{A} = [a_{ijk}], \mathbf{A} = [a_{ijkl}]$
- **transpose** of vector and matrix: $\underline{x}^T = [x_1, \ldots, x_N], \mathbf{A}^T = [a_{ji}]_{ij}$
- **determinant** of a square matrix $\mathbf{A}$: $|\mathbf{A}|$
- **inverse** of a square matrix $\mathbf{A}$: $\mathbf{A}^{-1}$
- **trace** of a square matrix $\mathbf{A}$: $\mathrm{tr}(\mathbf{A}) = \sum_i a_{ii}$

Abbildung 1.1: Mathematical notations

•Element notation (element of vector): $[x_i = \underline{a} + \underline{b}]_i$

## 1.1. What is machine learning

•Signal processing is not a subset of ML or the other way around, they are identical in the task
•Basic difference is in how to design the processing rule
•Regression means to calculate (SP,ML) a continuous-valued output in the real numbers from a signal, can be one-dimensional
•Classification means to calculate (SP,ML) a discrete-valued output from the natural numbers
•p. 1-5 filter in the middle is the view of a pixel and its 8 neighbours

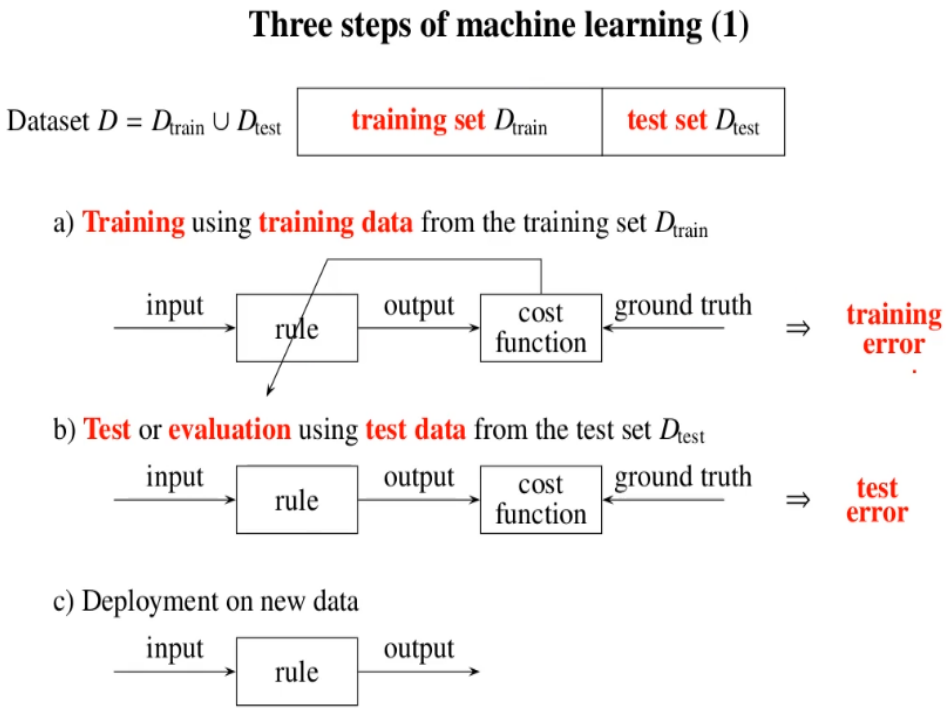



Abbildung 1.2: Three steps of machine learning

•over-fitting just memorizes the training set so test set is needed
•testerrorrate is similar to training error rate → no over-fitting
•Test set and training set need to be disjuct concatenated
•TODO: Summary supervised learning and unsupervised learning

## 1.2. What is deep learning?

•feature extraction: a feature is a clustered subset of information for recognition
•fish example: fish length, color etc.
•Needs human experience, can't be calculated
•DNN solves the problem without feature extraction



### 1.2.1. What is a neural network (NN)

•Cascaded pipeline of layers

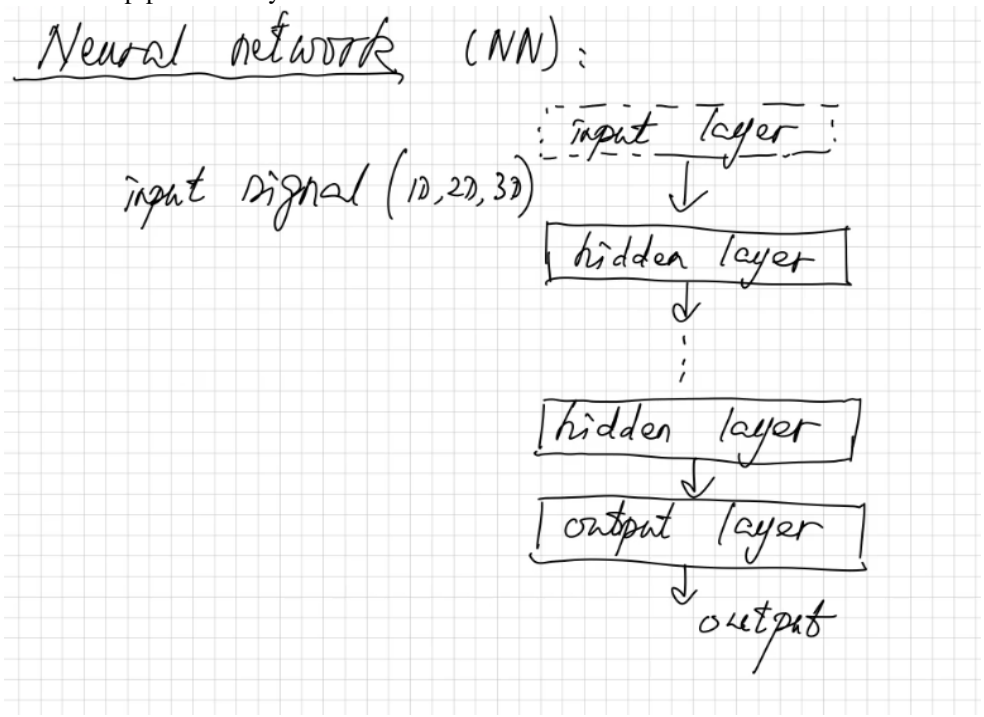



Abbildung 1.3: Neural Network

## 1.3. Examples for Deep Learning

•Semantic image segmentation is pixelwise classification

# Chapter 2: Tools for Deep Learning

Date: 02/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

## 2.1. Software

## 2.2. Hardware

## 2.3. Datasets

•Overview for Training Datasets up to ImageNet they are for teaching



# Chapter 3: Machine learning basics

Date: 02/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

## 3.1. Linear Algebra

•TODO get Math nicely into the Context book



**Vector norms**

Given a vector $\underline{x} = [x_i] \in \mathbb{R}^M$. There are different definitions for the vector norm:

- **2-norm** or $l_2$**-norm** or **Euclidean norm**: $\|\underline{x}\|_2 = \sqrt{\sum_{i=1}^{M} x_i^2} = \sqrt{\underline{x}^T \underline{x}}$
- **1-norm** or $l_1$**-norm**: $\|\underline{x}\|_1 = \sum_{i=1}^{M} |x_i|$
- **0-norm** or $l_0$**-norm**: $\|\underline{x}\|_0$ = number of non-zero elements in $\underline{x}$

Comments:

- $\|\underline{x}\|_2^2$ represents the energy of $\underline{x}$.
- $\|\underline{x}\|_0$ measures the sparsity of $\underline{x}$.
- Different vector norms have different unit-norm contour lines $\{\underline{x} \mid \|\underline{x}\|_p = 1\}$.

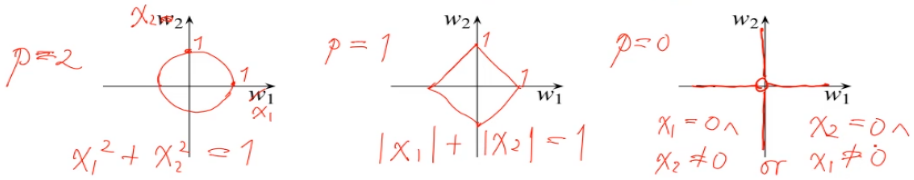



Abbildung 3.1: Vector Normes

## 3.2. Random variable and probability distribution

### 3.2.1. One random vector

- **scalar**: $x, A, \alpha$
- column **vector**: $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = [x_i] \in \mathbb{R}^N$ with element $x_i = [\underline{x}]_i$
- **matrix**: $\mathbf{A} = [a_{ij}]_{1 \le i \le M, 1 \le j \le N} = [a_{ij}] \in \mathbb{R}^{M \times N}$ with element $a_{ij} = [\mathbf{A}]_{ij}$
- 3D, 4D **tensor**: $\mathbf{A} = [a_{ijk}], \mathbf{A} = [a_{ijkl}]$
- **transpose** of vector and matrix: $\underline{x}^T = [x_1, \ldots, x_N], \mathbf{A}^T = [a_{ji}]_{ij}$
- **determinant** of a square matrix $\mathbf{A}$: $|\mathbf{A}|$
- **inverse** of a square matrix $\mathbf{A}$: $\mathbf{A}^{-1}$
- **trace** of a square matrix $\mathbf{A}$: $\mathrm{tr}(\mathbf{A}) = \sum_i a_{ii}$

Abbildung 3.2: One random vector

PDF for a discrete-valued RV:
$p(\underline{x}) = \sum_i p_i \delta(\underline{x} - \underline{x_i},)$ $\delta(\underline{x}:)$ Dirac function
**cumulative distribution function CCDF**
$F(\underline{x}) = \int_{-\infty}^{\infty} p(\underline{z}) d\underline{z}, \quad p(\underline{x}) = \frac{\partial^d F(\underline{x})}{\partial x_i \ldots \partial x_d}$



- **scalar**: $x, A, \alpha$
- column **vector**: $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = [x_i] \in \mathbb{R}^N$ with element $x_i = [\underline{x}]_i$
- **matrix**: $\mathbf{A} = [a_{ij}]_{1 \le i \le M, 1 \le j \le N} = [a_{ij}] \in \mathbb{R}^{M \times N}$ with element $a_{ij} = [\mathbf{A}]_{ij}$
- 3D, 4D **tensor**: $\mathbf{A} = [a_{ijk}], \mathbf{A} = [a_{ijkl}]$
- **transpose** of vector and matrix: $\underline{x}^T = [x_1, \ldots, x_N], \mathbf{A}^T = [a_{ji}]_{ij}$
- **determinant** of a square matrix $\mathbf{A}$: $|\mathbf{A}|$
- **inverse** of a square matrix $\mathbf{A}$: $\mathbf{A}^{-1}$
- **trace** of a square matrix $\mathbf{A}$: $\mathrm{tr}(\mathbf{A}) = \sum_i a_{ii}$

Abbildung 3.3: Moments of a vector

special case d = 1: $\underline{X} \to X \in \mathbb{R}$
$\mu \to \mu \in \mathbb{R}$
$\underline{\underline{C}} \to$ variance of $X = Var(X) = \delta^2 = E[(X - \mu)^2] = ... = E(X^2) - \mu^2$
$\delta = \sqrt{Var(X)}$ : standard deviation
For any function $g(\underline{X})$ of $\underline{X}$ : $E[g(\underline{x})] = \int g(x) \cdot p(\underline{x}) d\underline{x} = (d.v.) \sum_i g(\underline{x}_i \cdot P(\underline{x}_i))$

**Multivariate normal (Gaussian) distribution**

**PDF**

$$\underline{X} \in \mathbb{R}^d \sim N(\underline{\mu}, \mathbf{C})$$

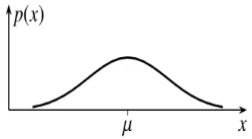
$$p(\underline{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{C}|^{1/2}} e^{-\frac{1}{2}(\underline{x}-\underline{\mu})^T \mathbf{C}^{-1}(\underline{x}-\underline{\mu})}$$

$$\ln(p(\underline{x})) = -\frac{d}{2}\ln(2\pi) - \frac{1}{2}\ln(|\mathbf{C}|) - \frac{1}{2}(\underline{x}-\underline{\mu})^T \mathbf{C}^{-1}(\underline{x}-\underline{\mu})$$

$N(\underline{0}, \mathbf{I})$ is called the **standard normal distribution**.

$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ identity matrix

**1D-Visualization**




**Moments**

$$\mathrm{E}(\underline{X}) = \underline{\mu}, \qquad \mathrm{Cov}(\underline{X}) = \mathbf{C}$$

Abbildung 3.4: Multivariate normal (Gaussian) distribution

**one-hot coding :** Only one bit is 1 e.g. 0100 one cold coding is the inverse
Coding for class label, random vector y of length c so the identity matrix with dimension c is used for class labels
Reforumulation of the PMF (categorical distribution) by one-hot coding of the classes $\underline{x} = [x_i] \in \{\underline{e}_1, \underline{e}_2, .., \underline{e}_c\}$ , i.e. all $x_i = 0$ except for one single element equal to 1

PMF: $P(\underline{X} = \underline{x}) = P(\underline{x}) = \begin{cases} P_i \text{ if } \underline{x} = \underline{e}_1 \text{ or } x_1 = 1 \\ ... \\ P_c \text{ if } \underline{x} = \underline{e}_c \text{ or } x_c = 1 \end{cases} = {p_1}^{x_1} \cdot {p_2}^{x_2} \cdot, ... \cdot {p_c}^{x_c} = \prod_{i=1}^{c} p_i^{x_i}$

$$ln(P(\underline{x}) = sum_{i=1}^{c} x_i \cdot ln(p_i) = [x_i, ..., x_c] \cdot \begin{bmatrix} ln(P_i) \\ ... \\ ln(P_c) \end{bmatrix} = \underline{x}^T \cdot ln(\underline{P})$$

ln function applied element wise

### 3.2.2. Multiple random vectors

•3-16 Table for distributions
•product rule for probability $p(\underline{x}, \underline{y}) = p(\underline{x}) \cdot p(\underline{y})$
•Bayes rule $p(\underline{y}|\underline{x}) = p(\underline{y}|\underline{x}) \cdot ) \frac{p(\underline{x}}{p(\underline{y}}$
•Independent and identically distributed
$\underline{x}$ and $\underline{y}$ are independent if:
$p(\underline{x}, \underline{y}) = p(\underline{x}) \cdot p(\underline{y}) \leftrightarrow p(\underline{x}|\underline{y})$ or $p(\underline{x}|\underline{y}) = p(\underline{y})$



$\underline{x}_1, ..., \underline{x}_N$ are independent and identically distributed (i.i.d)
$P(\underline{x}_1, ..., \underline{x}_N) = \prod_{i=1}^{N} p_i(\underline{x}_i), \underline{X}_i \sim p_i(\underline{x}_i)$
$p_i(\underline{x}_i) = p(\underline{x}_i) \rightarrow p(\underline{x}_1, ..., \underline{x}_N) = \prod_{i=1}^{N} p(\underline{x}_i)$

### 3.2.3. Kernel based density estimation

PDF: $p(\underline{x})$ of $\underline{X} \in \mathbb{R}^d$ unknown , only i.i.d samples $\underline{x}(n), 1 \leq n \leq N$
kernel-based estimate $\hat{p}(\underline{x}) of p(\underline{x})$ from $\underline{x}(n)$
kernel function $k(\underline{x})$, like a PDF
1. $k(\underline{x}) \geq 0 \forall \underline{x}$
2. $\int k(\underline{x}) d\underline{x} = 1$
$\hat{p}(\underline{x}) = \frac{1}{N} \sum_{n=1}^{N} k(\underline{x} - \underline{x}(n))$

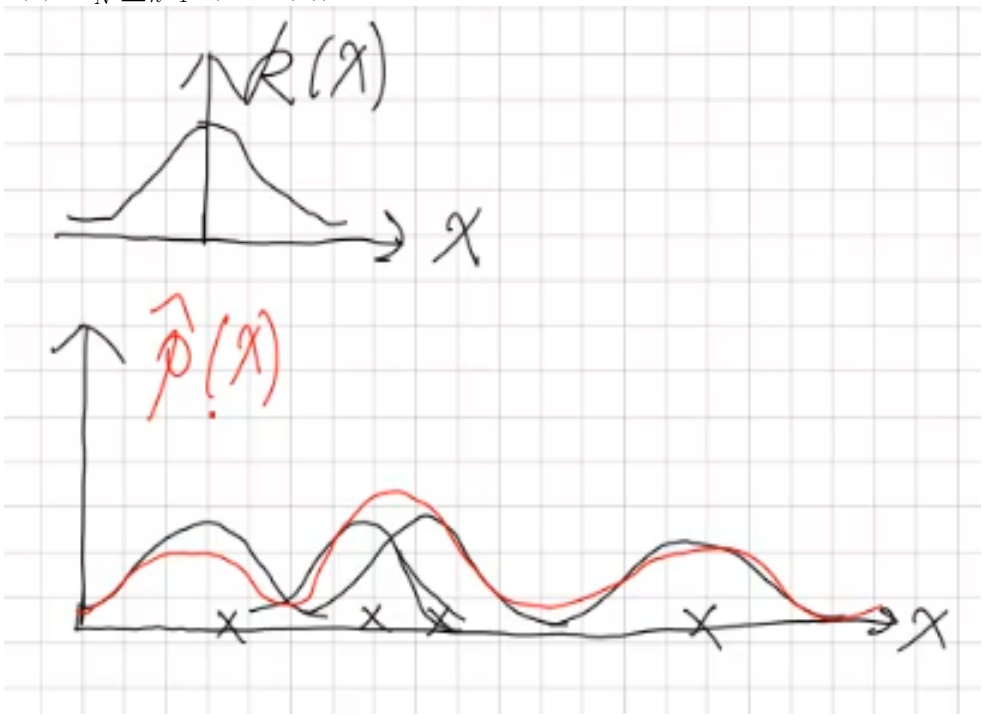



Abbildung 3.5: Kernel function

**Smooth Gaussian Kernel:**
$N(\underline{0}, \underline{I}) : k(\underline{x}) = \frac{1}{2\pi^{\frac{d}{2}}} \cdot e^{-\frac{1}{2}||\underline{x}||^2}$

**Dirac Kernel**
$k(\underline{x}) = \delta(\underline{x})$ : Dirac function
$\delta(\underline{x}) = \begin{cases} \infty, \underline{x} = \underline{0} \\ 0, \underline{x} \neq \underline{0} \end{cases}$
$\int \delta(\underline{x}) d\underline{x} = 1$
sampling property : $\int \delta(\underline{x} - \underline{x}_0 f(\underline{x}) d\underline{x}) = f(\underline{x}_0)$
**empirical distribution**
$\hat{p}(\underline{x}) \cdot \frac{1}{N} \sum_{n=1}^{N} \delta(\underline{x} - \underline{x}(n))$

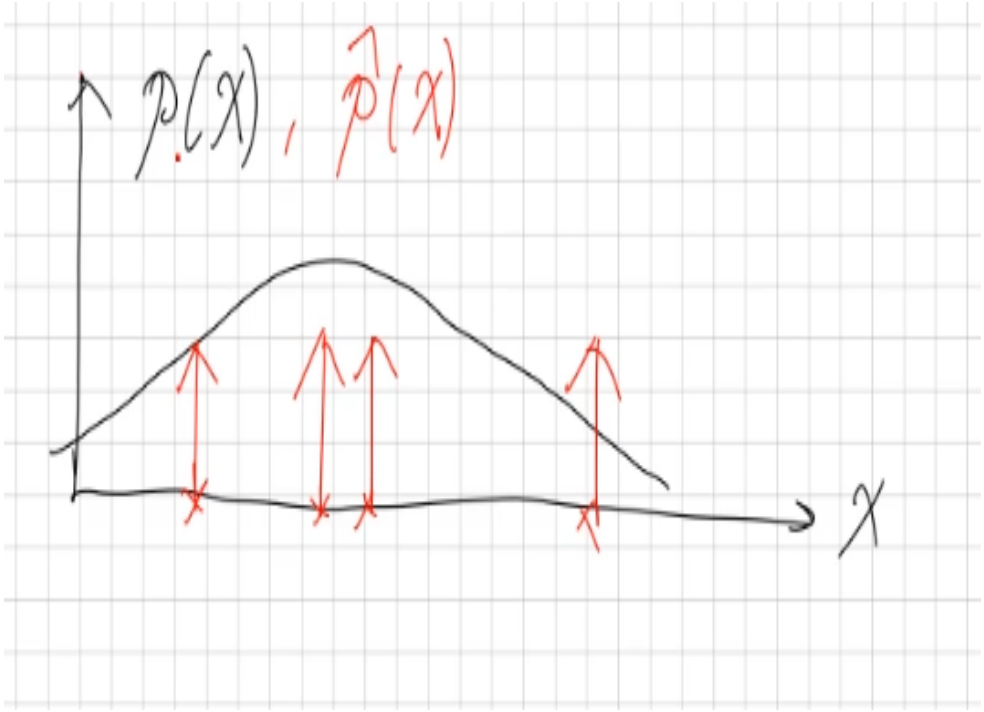


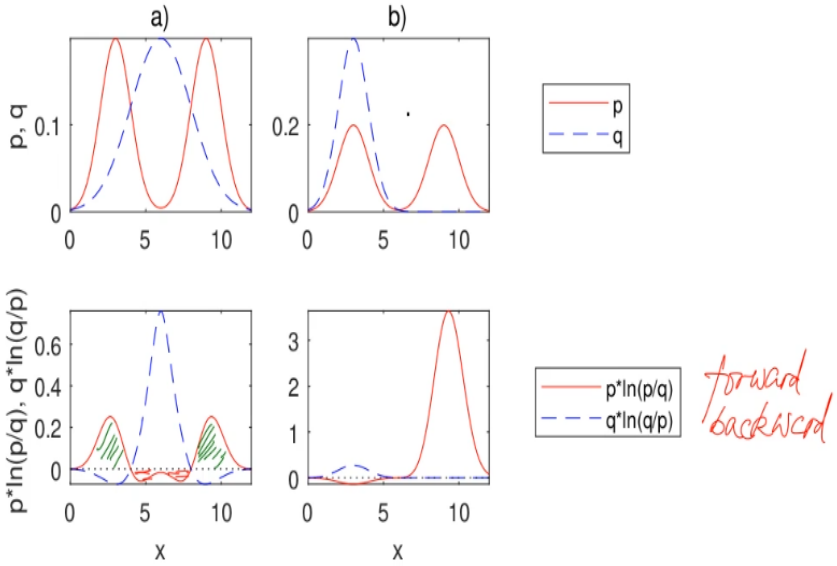

Abbildung 3.6: Estimated PDF function

## 3.3. Kullback-Leibler divergence and cross entropy

Dissimilarity measure between 2 distributions:
**Case A: continuous-valued random variables: PDF**
$\underline{X} \sim p(\underline{x})$ : true statistical distribution of $\underline{X}$
$q(\underline{x})$: approximation for $p(\underline{x})$, e.g. by DNN
**KL divergence (KLD) between p and q:**
$D_{KL}(p||q) = \int p(\underline{x}) \cdot ln(\frac{p(\underline{x})}{q(\underline{x})}) d\underline{x}$



$= E_{\underline{X} \sim p}[ln(\frac{p(\underline{X})}{q(\underline{X})})]$
expectation over $p(\underline{x})$
DKL is real valued scalar positive or negative or 0
**Case B: discrete-valued random vector: PMF**
$\underline{X} \in \{\underline{x}_1, ...\underline{x}_c\}$ $\sim$: true PMF of $\underline{X} \sim Q(\underline{x})$: approximation for $P(\underline{x})$
$D_{KL}(P||Q) = \sum_{i=1}^{c} P(\underline{x}_i) \cdot ln(\frac{P(\underline{x})}{Q(\underline{x})}) = E_{\underline{X} \sim P}[ln(\frac{P(\underline{x})}{Q(\underline{x})})]$
Properties of the KL divergence: P1) Nonnegative $D_{KL}(P||Q) \geq 0 \forall p, q$
P2) Eguality $D_{KL}(P||Q) = 0$ iff(if and only if) $p(\underline{x}) = q(\underline{x})$
proof for ßufficient": $ln(\frac{p(\underline{x})}{q(\underline{x})}) = 0 \forall \underline{x}$
P1 and P2: $D_{KL}(p||1)$ is a suitable metric for approximation p by q
P3 Asymmetry
$D_{KL}(p||q) = E_{\underline{X} \sim p} = ln(\frac{p(\underline{x})}{q(\underline{x})}) \neq D_{KL}(q||p) = E_{\underline{X} \sim q} = ln(\frac{q(\underline{x})}{p(\underline{x})})$
forward KLD backward KLD
$D_{KL}$ is not a true distance measure with $D(\underline{x}, \underline{y}) = D(\underline{y}, \underline{x})$




- When minimizing $D_{\mathrm{KL}}(p\|q)$, a) is better than b) because $q(x)$ is broad.
- When minimizing $D_{\mathrm{KL}}(q\|p)$, b) is better than a) because $q(x)$ is narrow.

Abbildung 3.7: Forward vs. Backward KL divergence

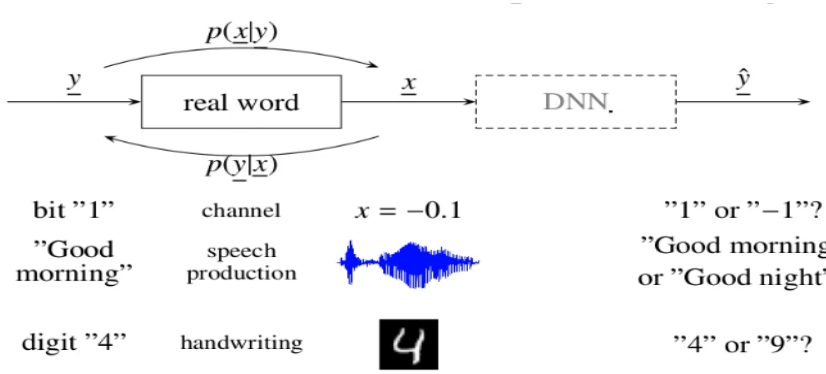
### 3.3.1. E3.5 KLD between normal and Laplace distribution

$p(x) =\sim N(0,\sigma^2), p(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$
$q(x) \sim$ Laplace: $(0,b) q(x) = \frac{1}{2b} e^{-\frac{|x|}{b}}$
Task: choose b to best approximate p by q.
$\frac{p(x)}{q(x)} = \sqrt{\frac{2}{\pi}} \cdot \frac{b}{\sigma} cdotexp(-\frac{x^2}{2\sigma^2} + \frac{|x|}{b})$
$D_{KL}(p||q) = E_{\underline{X}\ simp} = ln(\frac{p(\underline{x})}{q(\underline{x})}) = ln(\sqrt{\frac{2}{\pi}} \cdot \frac{b}{\sigma}) + E_{X\sim p}((-\frac{x^2}{2\sigma^2} + \frac{|x|}{b}))$
$E_{\underline{X}\ simp} = \sigma^2$
$E_{\underline{X}\ simp}(|x|) = \int_{-\infty}^{\infty} |x| \cdot \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{x^2}{2\sigma^2}) dx = 2\int_0^{\infty} | \cdot \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{x^2}{2\sigma^2}) dx = \sqrt{\frac{2}{\pi}} \cdot \sigma$
Let $\alpha = \frac{\sigma}{b}, D_{KL}(p||q) = ... = \sqrt{\frac{2}{\pi}} \cdot \alpha - ln(\alpha) + ln(\sqrt{\frac{2}{\pi}}) - \frac{1}{2} \frac{dD_{KL}(p||q)}{d\alpha} = \sqrt{\frac{2}{\pi}} - \frac{1}{\alpha} =$
$0 \rightarrow \alpha = \sqrt{\frac{2}{\pi}}$, i.e. $b \approx 0,86$
$D_{KL,min}(p||q) = D_{KL}(p||q)|\alpha = \sqrt{\frac{2}{\pi}} = .. = \frac{1}{2} - ln(\frac{\pi}{2}) \approx 0,048$
•Probable exam question calculate this for 2 distributions
P4) Additive :
$\underline{X}) = (\underline{x}_1, \underline{x}_2), \underline{x}_1$ and $\underline{x}_2$ are independent, $i.e.$
$p(\underline{x})?p_1(\underline{x}_1) \cdot p_2(\underline{x}_2), q(\underline{x})?q_1(\underline{x}_1) \cdot q_2(\underline{x}_2)$
Then: $D_{KL}(p||q) = D_{KL}(p_1||q_1) + D_{KL}(p_2||q_2)$
P5) Relation to cross entropy :
Definiton of entropy 3-24 probability always greater than 0 but smaller than 1
$D_{KL}(p||q) = \int pln(\frac{p}{q}) d\underline{x} = \int pln(p) dx - \int pln(q) dx = -H(p) + H(p,q)$ or
cross entropy: $H(p,q) = D_{KL}(p||q) + H(p) \geq H(p) \geq 0$
For a given (fixed) $p(\underline{x})$ : $H(p)$ fixed
**Hence:** min $D_{KL}(p||q) \leftrightarrow minH(p,q)$
Not the case for backward KLD $D_{KL}(q||p)$!
Minimizing is not the same anymore because then it is $H(q)$ and thats what we are trying to optimize

## 3.4. Probabilistic framework for machine learning

valid for both SP and ML
valid for both regression problem and classicication problem






- $\underline{y}$: **latent variable**, hidden, not directly measurable, quantity of interest
- $\underline{x}$: **observed variable**, measurement, input for DNN
- $\hat{\underline{y}}$: output of DNN as estimate for $\underline{y}$
- real world: describes how is $\underline{x}$ generated from $\underline{y}$
- DNN: describes how to estimate $\underline{y}$ from $\underline{x}$

Abbildung 3.8: Probabilistic framework of supervised learning

Both $\underline{y}$ and $\underline{x}$ are modeled as random variables. They are described by the joint **data generating distribution**

$$p(\underline{x}, \underline{y}) = p(\underline{x}|\underline{y})p(\underline{y}) = p(\underline{y}|\underline{x})p(\underline{x}).$$

$p(\underline{y})$ prior PDF of $\underline{y}$ or **prior**, available before any measurement of $\underline{x}$

$p(\underline{x}|\underline{y})$ **likelihood**. It describes the real word, the generation of $\underline{x}$ from $\underline{y}$.
It is a kind of channel-sensor model, e.g.
- bit: communication channel + receiver
- speech: speech production system + microphone
- digit: handwriting + camera

$p(\underline{x})$ prior PDF of $\underline{x}$, also called **evidence**

$p(\underline{y}|\underline{x})$ posterior PDF of $\underline{y}$ after a measurement $\underline{x}$ or **posterior**

Abbildung 3.9: The data generating distribution

Bayes Rule:

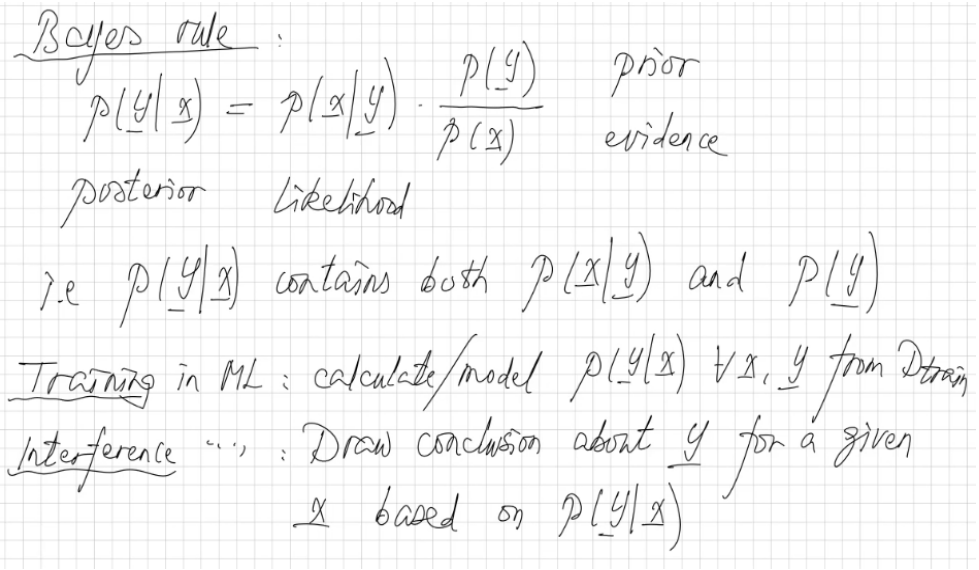



Abbildung 3.10: Bayes Rule in DL

a) **Maximum a posterior (MAP)** inference:

$$\max_{y} p(y|x) = p(x|y)\frac{p(y)}{p(x)}$$

b) **Minimum Bayesian risk (MBR)** inference:

$$\min_{\hat{y}} \mathrm{E}_{X,Y}[l(Y, \hat{y}(X))] = \int l(y, \hat{y}(x))p(x, y)\mathrm{d}x\mathrm{d}y$$

see course DPR and SASP for more details.

Abbildung 3.11: Bayes decision theorem

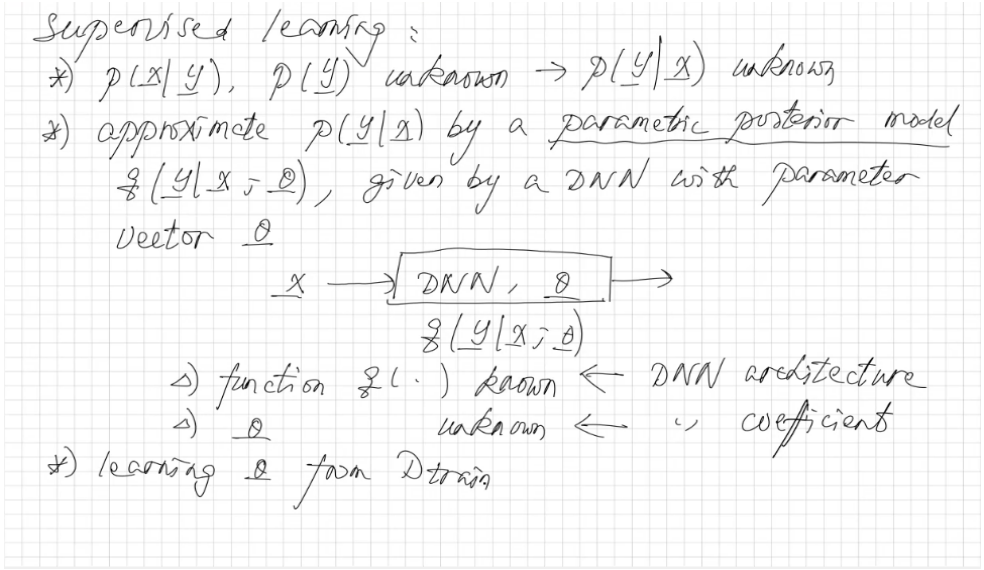



Abbildung 3.12: Supervised learning

**Learning Criterion:**

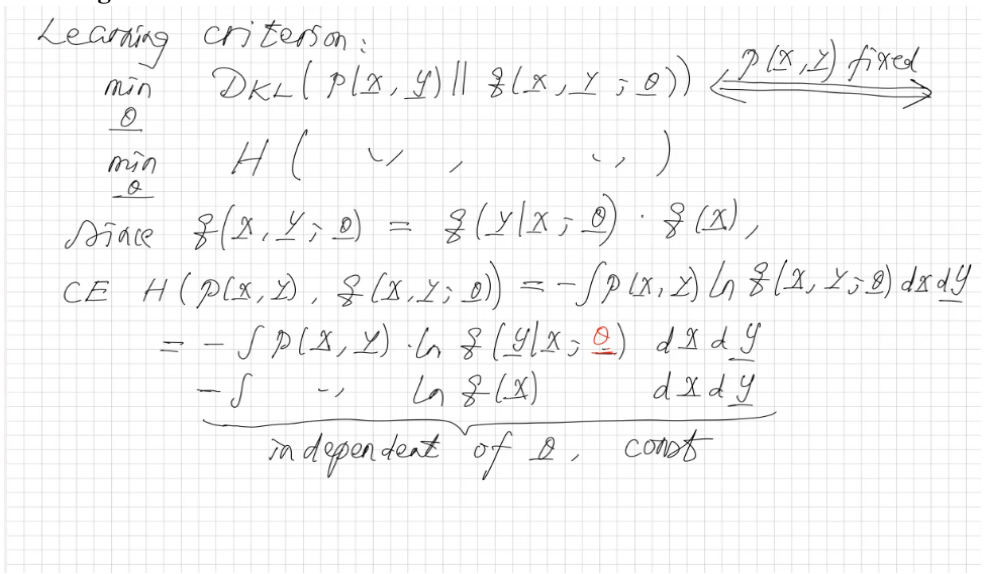



Abbildung 3.13: Calc part1

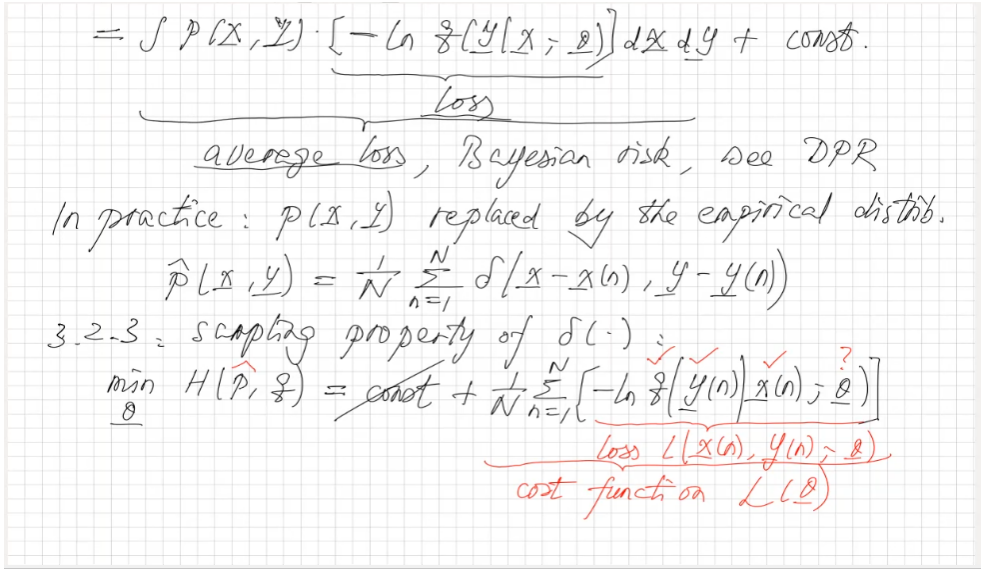



Abbildung 3.14: Calc part2

### 3.4.1. Role of a NN

1. approximate true posterior $p(\underline{y}|\underline{x})$ by $q(\underline{y}|\underline{x};\Theta)$
2. learn $\underline{\Theta}$ from $D_{train}$



# Chapter 4: Dense Neural Networks

Date: 05/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

A general model for $q(\underline{y}|\underline{x};\Theta)$
*) can learn any linear or nonlinear mapping
*) suitable for both regression and classification problems
artificial NN: mimic biological NN (brain)

## 4.1. Fully connected neural networks - Neuron

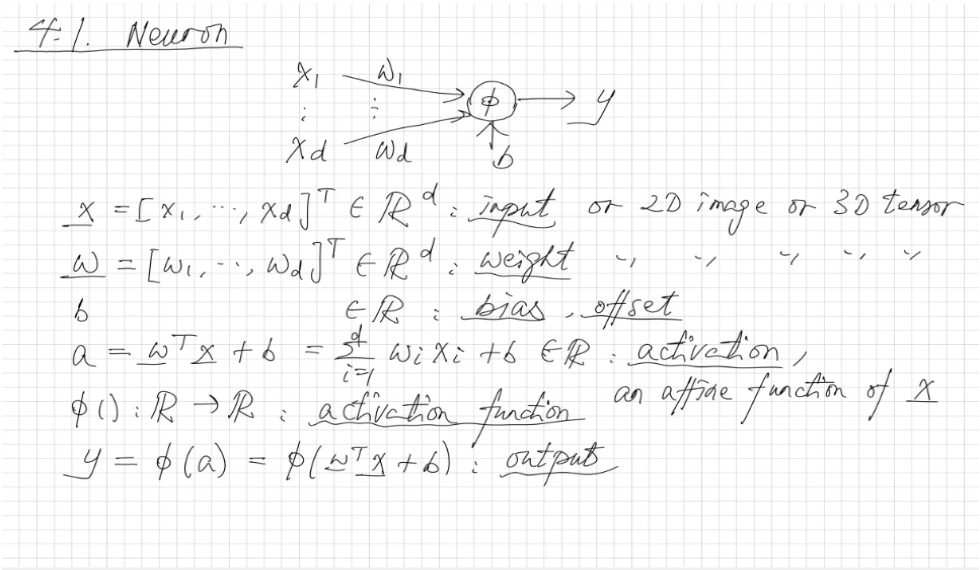



Abbildung 4.1: Neuron

## 4.2. Chapter 4.2 Layer of Nurons

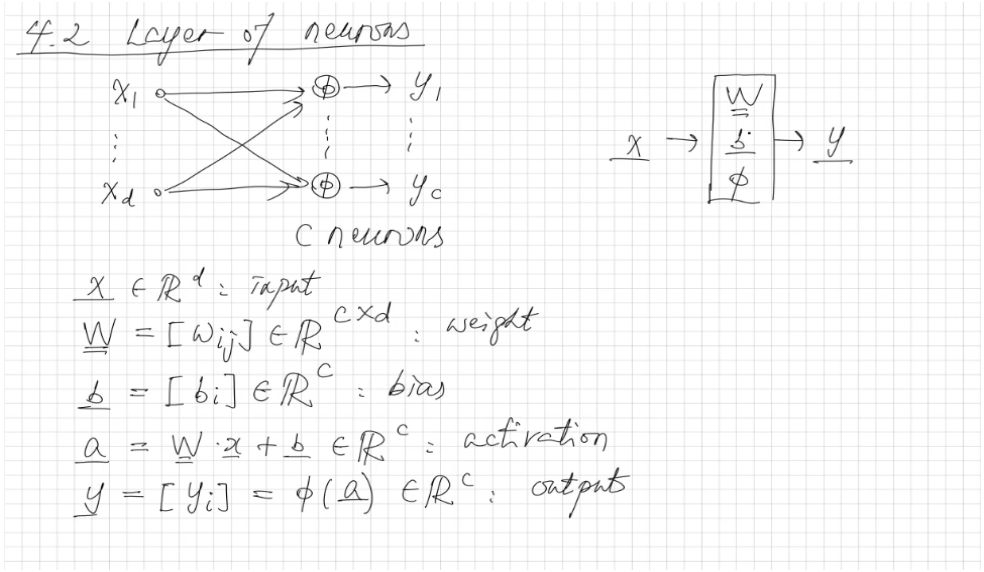



Abbildung 4.2: Layer of Neurons

2 meanings of $\phi()$:

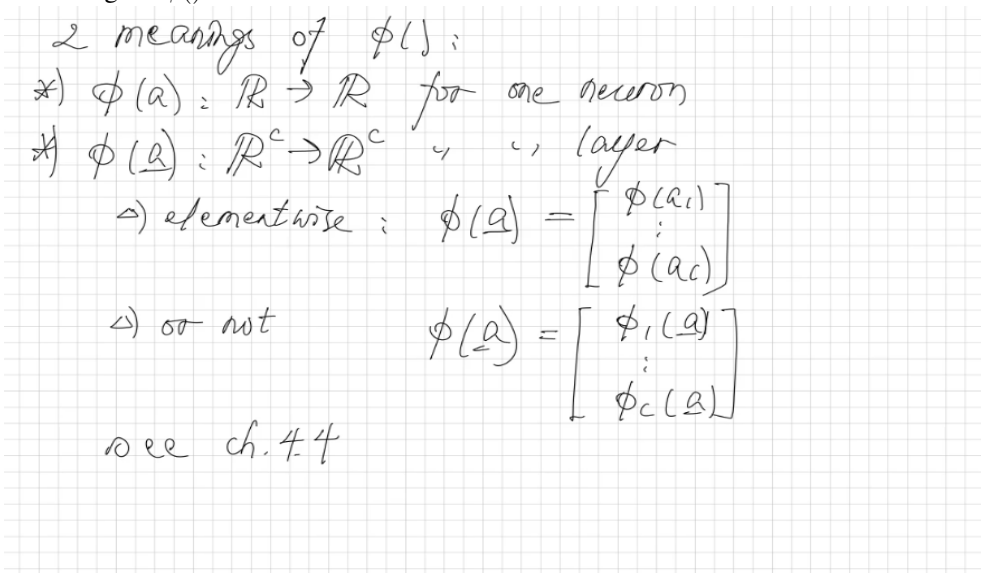



Abbildung 4.3: Meanings of phi



Comments :

•no interconnections between neurons in the same layer

•dense layer, fully connected layer: •each input $x_j$ connected to each neuron $i$

$\rightarrow c \cdot d$ weights and $w_{ij}$ an c biases $b_i$ , $1 \leq i \leq c, 1 \leq j \leq d$,

$\rightarrow c \cdot (d+1)$ parameters

## 4.3. Feedforward neural network

A cascade of dense layers

**Layer** $1 \leq l \leq L$ :

$M_l - 1$ number of the input neurons

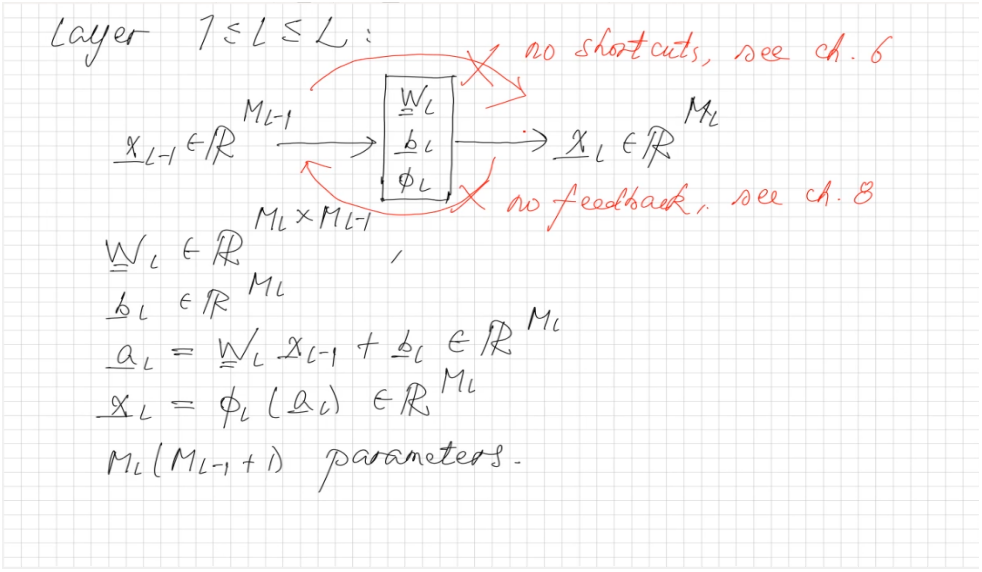



Abbildung 4.4: Feedforward multilayer neural network

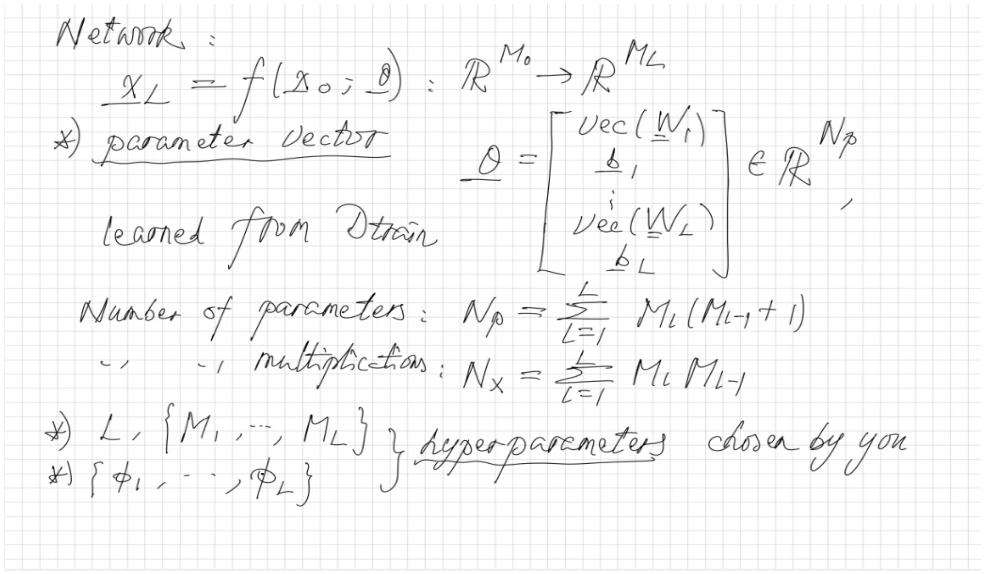



Abbildung 4.5: Network Parameters

## 4.4. Activation function

Mild requirements on $\phi()$:
•nonlinear in general → fundamental
•smooth, differentiable → for training
•simple calculation → low complexity
•Slides 4-6; 4-7 activation function types



### 4.4.1. Sigmoid activation function

$$\phi(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$$

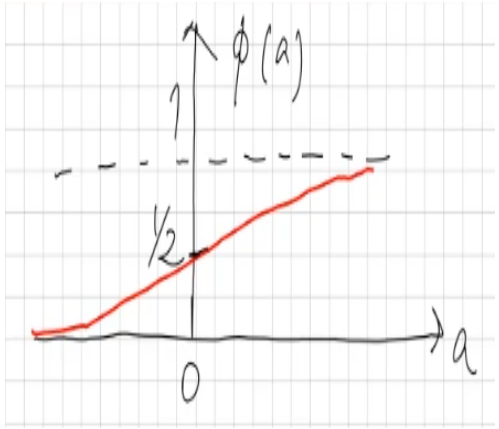



Abbildung 4.6: Sigmoid function

•$0 < \phi < 1 \hat{=}$ prob.
•symmetry: $\phi(-a) = 1 - \phi(a)$
•derivative: $\frac{d\phi(a)}{da} = ... = \frac{e^{-a}}{(1+e^{-a})^2} = \phi(a) \cdot \phi(-a) = \phi(a)(1 - \phi(a)), \in (0, 1)$
easy calculative
•widely used in conventional NN (shallow)

### 4.4.2. hyperbolic tangent activation function

$$\phi(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

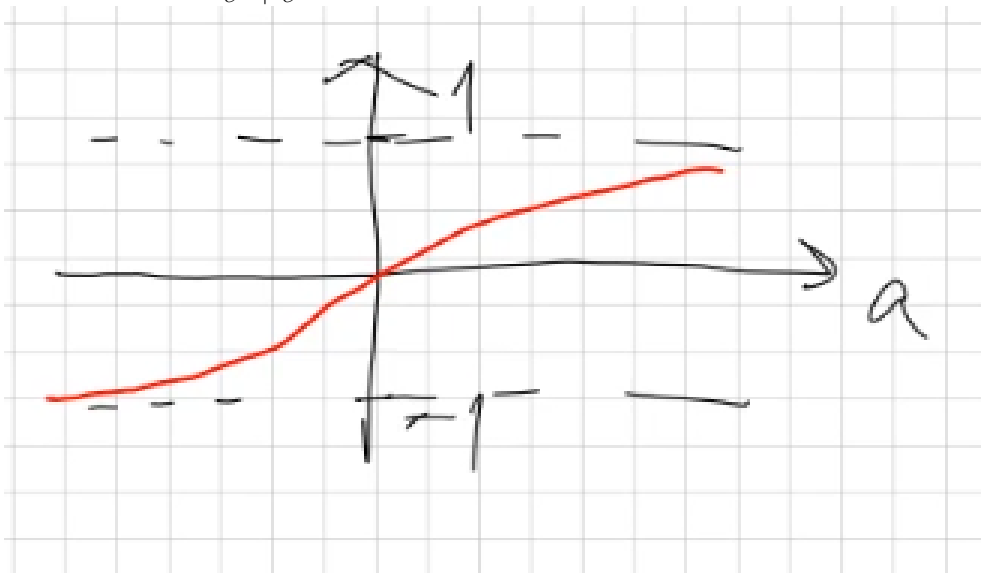



Abbildung 4.7: HyperbolicTangentActivation

like sigmoid but another output range



### 4.4.3. rectifier linear unit(ReLU

$$\phi(a) = ReLU(a) = max(a, 0) = \begin{cases} a & a \geq 0 \\ 0 & a < 0 \end{cases}$$

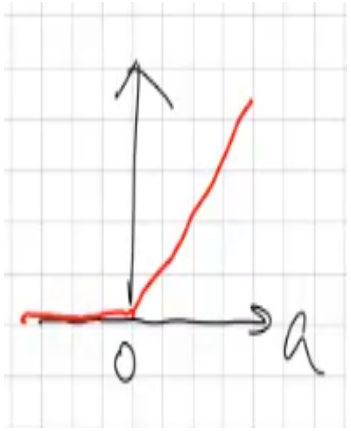



Abbildung 4.8: ReLuA ctivation Function

- $\equiv$ diode

simple calculation

- $\frac{d\phi}{da} = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases} = u(a), u(0) = 0$ typically used
- most popular in DNN
- Details on 4-7

### 4.4.4. Softmax activatoin function(classification problem)

$\phi(\underline{a} : \underline{a} = [a_i] \in \Re^c \rightarrow \Re^c$

$$\phi(\underline{a}) = softmax(\underline{a}) = \begin{bmatrix} \phi_1(\underline{a}) \\ \vdots \\ \phi_c(\underline{a}) \end{bmatrix}$$

$\phi(\underline{a}) = \frac{a^{a_i}}{\sum_{j=1}^{c} e^{a_i}}, \in (0, 1), \sum_{i=1}^{c} \phi_i(\underline{a}) = 1$

- maps $\underline{a} \in \Re^c$ to a categorical PMF with $c$ classes
- $a_i$ large$\rightarrow$ $\phi_i(\underline{a})$ close to 1
- $a_i$ small $\rightarrow$ $\phi_i(\underline{a})$ close to 0
- used in the output layer for classification problems

### 4.4.5. Special case c=2, binary classification problem

$$: \phi_1(\underline{a}) = \frac{e^{a_i}}{e^{a_1} + e^{a_2}} = \frac{1}{1 + e^{-(a_1 - a_2)}} = \sigma(a_1 - a_2)$$

$$\phi_2(\underline{a} = \frac{e^{a_2}}{e^{a_1} + e^{a_2}} = 1 - \phi_1(\underline{a}) = \sigma(a_2 - a_1)$$

i.e. softmax

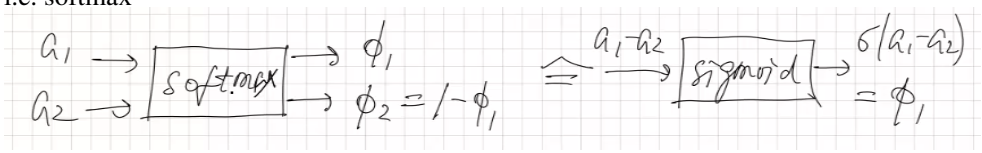



Abbildung 4.9: Outputlayer Softmax

one sigmoid output is sufficient for binary classification instead of 2 output softmax!

**Derivative of softmax:**

$$\frac{\partial \phi_i(\underline{a})}{\partial a_j} = ... = \begin{cases} \phi : i(\underline{a}) \cdot (1 - \phi_i(\underline{a}) & i = j \\ -phi_i(\underline{a}) \cdot \phi_j(\underline{a}) & i \neq j \end{cases}$$

•4-9 for details on usage

## 4.5. Universal approximation

4.5 Universal approximation 4-10

**Universal approximation theorem** ⇔ existence of a solution

The **universal approximation theorem** states that a feedforward neural network with a linear output layer ($\phi_L(a) = a$) and

- *at least one hidden layer* with
- a *nonlinear* activation function

can approximate any continuous (nonlinear) function $\underline{y}(\underline{x}_0)$ (on compact input sets) to arbitrary accuracy.

Comments:

- arbitrary accuracy: with an increasing number of hidden neurons.
- valid for a wide range of nonlinear activation functions, but excluding polynomials.
- minimum requirement for universal approximation: $\mathbf{W}_2\phi_1(\mathbf{W}_1\underline{x}_0 + \underline{b}_1) + \underline{b}_2$.

For learning $\mathbf{W}_l$ and $\underline{b}_l$ from $D_{\text{train}}$:

- deep networks are better than shallow ones,
- some activation functions are better than others.

⇔ how to find a good solution.



### 4.5.1. E4.3 Regression with 1 hidden layer

True function: $f_0(x)$

Given: $x(n)$ and noisy function $y(n) = f_(x(n)) + z(n), 1 \leq n \leq N, z(n)$ is the noise

NN:

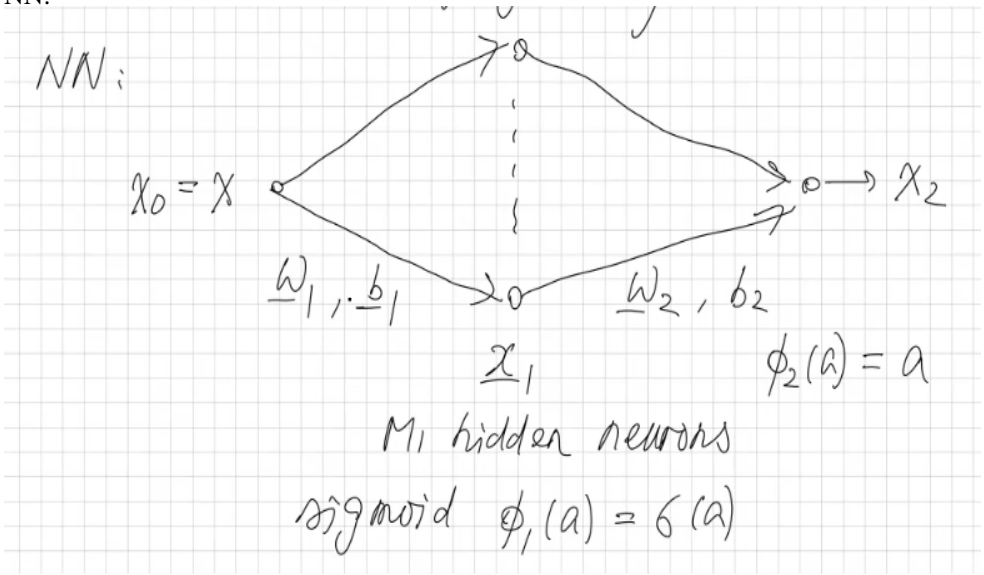



Abbildung 4.10: Example 4.3 Architecture

i.e. $x_2 = f(x, \underline{\theta}) = \underline{w}_2^T \sigma(\underline{w}_1 x + \underline{b}_1)$, column times row times scalar

$= \sum_{i=1}^{N_1} w_{2,i} \cdot \underbrace{\sigma(w_{1,i}x + b_{1,i})}_{M_1 \text{ nonllinear basis functions of } x} + b_2$

$\sigma(w_i x + b_i)\sigma(w_i(x + \frac{b_i}{w_i}))$

new center at $\frac{-b_i}{w_i}$ and new slope value $w_i$

Picture is for black 1 $w_i$ red another $w_i$ green another example $w_i$

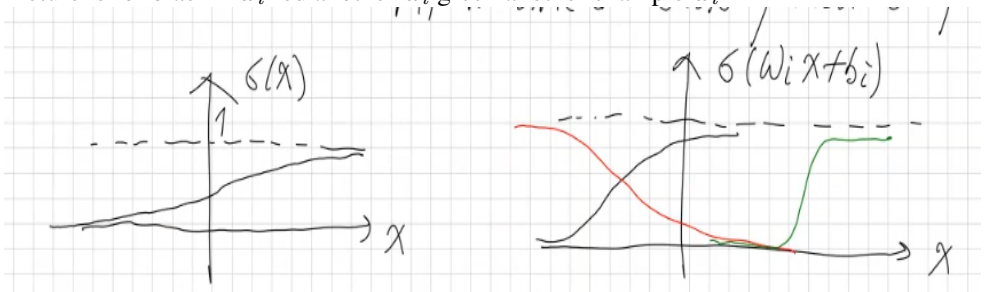



Abbildung 4.11: Sketch for sigmoid

## 4.6. Loss and cost function

**Review chapter 3.4**

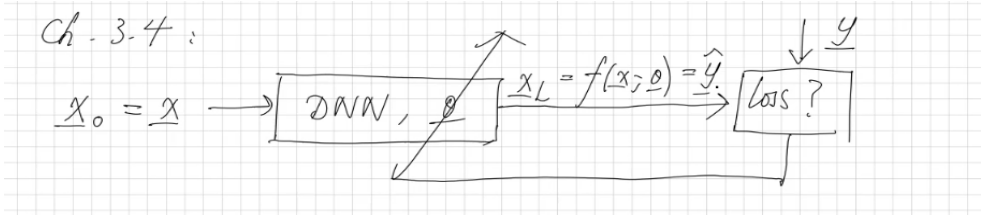



Abbildung 4.12: Probabalisitc Framework supervised learning

$\min_{\underline{\theta}} L(\underline{\theta}) = \frac{1}{N} = \sum_{n=1}^{N} (l(\underline{x}(n), y(n)_i \underline{\theta}$ cost function for $d_{train}$
$l(\underline{x}, \underline{y}_j \underline{\theta}) = -ln(q(\underline{y}|\underline{x}_j \underline{\theta})$ loss for one pair $(\underline{x}, \underline{y})$
$q() \leftrightarrow$ DNN ???

### 4.6.1. Regression Problem

$\underline{x} \in \Re^d$: random input
$\underline{y} \in \Re^c$ desired random output
Assumption: DNN estimates the mean of $\underline{y}$, i.e.
$\underline{y} = f(\underline{x}, \underline{\theta}) + \underline{z}, \underline{z}$ : noise
f is calculated by DNN
**case A:** : $\underline{z} \sim N(\underline{0}, \sigma^2 \underline{\underline{I}})$, white Gaussian noise
$\underline{y} \sim N(f(\underline{x}, \underline{\theta}), \sigma^2 \underline{\underline{I}})$
$q(\underline{y}|\underline{x}, \underline{\theta}) = \frac{1}{(2\pi\sigma^2)^{c/2}} exp(\frac{-1}{2\sigma^2} ||\underline{y} - f(\underline{x}, \underline{\theta})||^2)$
$l(\underline{x}, \underline{y}, \underline{\theta}) = -ln(q(\underline{y}|\underline{x}, \underline{\theta})) = const. + \frac{1}{\sigma^2} ||\underline{y} - f(\underline{x}, \underline{\theta})||^2$
$L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^{N} ||\underline{y}(n) - f(\underline{x}, \underline{\theta})||^2$
$\rightarrow l_2 - loss$, mean square error (MSE) loss , least squares (LS) method
min $L(\underline{\theta})$ is a parameter estimation problem
**case B:**
$\underline{z} \sim N(\underline{0}, \underline{\underline{C}})$, colored Gaussian noise
$L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^{N} [\underline{y} - (f(\underline{x}(n), \underline{\theta})]^T \cdot \underline{\underline{C}}^{-1} \cdot [\underline{y} - (f(\underline{x}), \underline{\theta})]$ , **weighted MSE loss**
Rarely used in real life applications:
•how to know $\underline{\underline{c}}$
•$\underline{\underline{C}}^{-1}$ expensive for large $\underline{\underline{C}}$



### 4.6.2. Classification

$\underline{x}\ in \Re^d$: input
$\underline{y} \in \{\underline{e}_1, \underline{e}_2, .., \underline{e}_c\}$
: class label for $\underline{x}$ in one-hot coding
Let $p_i = P(\underline{y} = \underline{e}i|\underline{x})$: true posterior probability
ch: 3.2 : $P(\underline{y}|\underline{x}) = \prod_{i=1}^{c} p_i^{y_i}$ true PMF
But $p_i$ unknown
**DNN:** •output $f(\underline{y}, \underline{\theta}) = [f_i(\underline{x}; \underline{\theta})] \in \Re^c$ an estimate for $[p_i]$
•i. e. $P(\underline{y}|\underline{x})$ approximated $Q(\underline{y}|\underline{x}; \underline{\theta}) = \prod_{i=1}^{c} f_i(\underline{x}; \underline{\theta})^{y_i}$
in order to ensure :
•$0 < f_i(\underline{y}; \underline{\theta}) < 1$

•$\sum_{i=1}^{c} f_i(\underline{x}; \underline{\theta}) = 1$,
softmax is used in the output layer :
$\underline{x}_L = f(\underline{x}; \underline{\theta}) = \phi_L(\underline{a}_l L) = softmax(\underline{a}_L)$, see ch.4.4
$\rightarrow$ Loss $l(\underline{x}, \underline{y}; \underline{\theta}) = -ln(Q(\underline{y}|\underline{x}; \underline{\theta})) = \sum_{i=1}^{c} y_i ln(f_i(\underline{y}; \underline{\theta})) = -\underline{y}^T ln(f(\underline{x}; \underline{\theta})) \geq 0$
categorical cross entropy(CE) loss
Special case : Binary classification , c = 2
ch.4.4: $softmax, c = 2 \leftrightarrows sigmoid$
i.e. one output neuron with sigmoid activation function $f(\underline{x}; \underline{\theta}) = \sigma(a_L)$ is sufficient
Let $y_1 = y, y_2 = 1 - y; f_1 = f; f_2 = 1 - f$
$\rightarrow l(\underline{x}, \underline{y}; \underline{\theta}) = -yln(f(\underline{x}; \underline{\theta}) + (1 - y)) \cdot ln(1 - f(\underline{x}; \underline{\theta}))$ , binary CE loss
**True probabilistic way toward cost functions**

### The probabilistic way toward the cost functions

- $p(\underline{x}, \underline{y})$: true but unknown data generating distribution, application specific
- $D_{\text{train}} = \{\underline{x}(n), \underline{y}(n), 1 \leq n \leq N\}$: training set, i.i.d. samples of $p(\underline{x}, \underline{y})$
- $p(\underline{y}|\underline{x})$: true posterior describing the desired inference $\underline{x} \rightarrow \underline{y}$
- $q(\underline{y}|\underline{x}; \underline{\theta})$: a parametric model (DNN) to approximate $p(\underline{y}|\underline{x})$

min. forward KL divergence $D_{\text{KL}}(p(\underline{x}, \underline{y}) \| q(\underline{x}, \underline{y}; \underline{\theta}))$

↓ ch. 3.3: $p$ fixed

min. cross entropy $H(p, q) = -\text{E}_{\underline{X}, \underline{Y} \sim p(\underline{x}, \underline{y})} \ln(q(\underline{Y}|\underline{X}; \underline{\theta}))$

↓ ch. 3.4: use empirical distribution $\hat{p}(\underline{x}, \underline{y})$

min. cross entropy $H(\hat{p}, q) = -\text{E}_{\underline{X}, \underline{Y} \sim \hat{p}(\underline{x}, \underline{y})} \ln(q(\underline{Y}|\underline{X}; \underline{\theta}))$

↓ ignore constant term

min. cost function $L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^{N} [-\ln(q(\underline{y}(n)|\underline{x}(n); \underline{\theta}))]$

↓ ch. 4.6: $q(\underline{y}|\underline{x}; \underline{\theta})$ for regression and classification?

$l_2$-loss or $l_1$-loss or categorical loss

### 4.6.3. Semantic segmentation

**pixelwise classification**
categorical cross entropy loss:
$l(\underline{\underline{X}}, \underline{\underline{Y}}; \underline{\theta}) = \sum_{m=1}^{M} \sum_{n=1}^{N} [-\underline{y}_{mn} \cdot ln(\hat{\underline{y}}_{mn}(\underline{\underline{X}}, \underline{\theta}))]$ sum over all pixels
Problem: imbalanced classes
e.g (c=2) background and foreground with 90 % background and 10 % foreground pixels , 90 % loss function for the foreground
→ $l(\cdot)$ cares more about the major class and less about the minor class but the minor class(foreground) is object of interest. → reduced segmentation accuracy for the minor class
**Solutions:**
•Weighted categorical CE loss
•region-based loss **Jaccard index only used for result evaluation not suitable as loss function for training**
•minimize loss, not max J or D for those indexes
•$|A| \in \mathbb{N}$, not differential with respect to $\underline{\theta}$
•$\underline{y}_{mn}$ contains 0 or 1 as desired, but $\underline{\hat{y}}$ contains real numbers $\in (0, 1) \in softmax$



→ adapted definition of J and D are necessary



### Soft Jaccard and Dice loss

Let again $\mathbf{Y} = [\underline{\underline{y}}_{mn}]_{mn}$ and $\hat{\mathbf{Y}} = [\underline{\hat{y}}_{mn}]_{mn}$. $\underline{y}_{mn} \in \{\underline{e}_1, \ldots, \underline{e}_c\}$ is the one-hot coding and $\underline{\hat{y}}_{mn} \in \mathbb{R}^c$ is the $c$-class softmax output for the pixel $(m, n)$. The **soft Jaccard loss** to be minimized in training for image segmentation is

$$\begin{aligned} \underline{\alpha} &= \sum_{m=1}^{M} \sum_{n=1}^{N} \underline{y}_{mn} \odot \underline{\hat{y}}_{mn} = [\alpha_i] \in \mathbb{R}^c, \\ \underline{\beta} &= \sum_{m=1}^{M} \sum_{n=1}^{N} (\underline{y}_{mn} + \underline{\hat{y}}_{mn}) = [\beta_i] \in \mathbb{R}^c, \\ J_i &= \frac{\alpha_i + \epsilon}{\beta_i - \alpha_i + \epsilon}, \\ l(\mathbf{X}, \mathbf{Y}; \underline{\theta}) &= 1 - \frac{1}{c} \sum_{i=1}^{c} J_i \end{aligned}$$

$\odot$ is the elementwise multiplication. $\alpha_i$ and $\beta_i$ represent arithmetic calculations of $|A_i \cap B_i|$ and $|A_i| + |B_i|$ for class $i$, respectively. $J_i$ is the soft Jaccard index for class $i$ where $\epsilon > 0$ is a suitable number to avoid $0/0$ if $\alpha_i = \beta_i = 0$. $l(\mathbf{X}, \mathbf{Y}; \underline{\theta})$ is the soft Jaccard loss differentiable w.r.t. $\underline{\theta}$. The **soft Dice loss** is defined in a similar way.

For 3D segmentation, $\underline{\alpha}$ and $\underline{\beta}$ are calculated by three-dimensional sums over all pixels.

Abbildung 4.13: Soft Jaccard / Dice loss

**Very good for strongly imbalanced classes**

## 4.7. Training

•training set $D_{train} = \{\underline{x}, \underline{y}, 1 \leq n \leq N\}$
•cost function $L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^{N} l(\underline{x}(n), \underline{y}; \underline{\theta})$
•task: $minL(\underline{\theta})$
•optimizer: optimization algorithm to solve the minimization task $L(\underline{\theta})$
No closed-form solution! Numerical minimization necessary, see AM (last part)
**in DL: gradient decent approach and variants(like hiking)**
need only 1. order derivative of $L(\underline{\theta})$

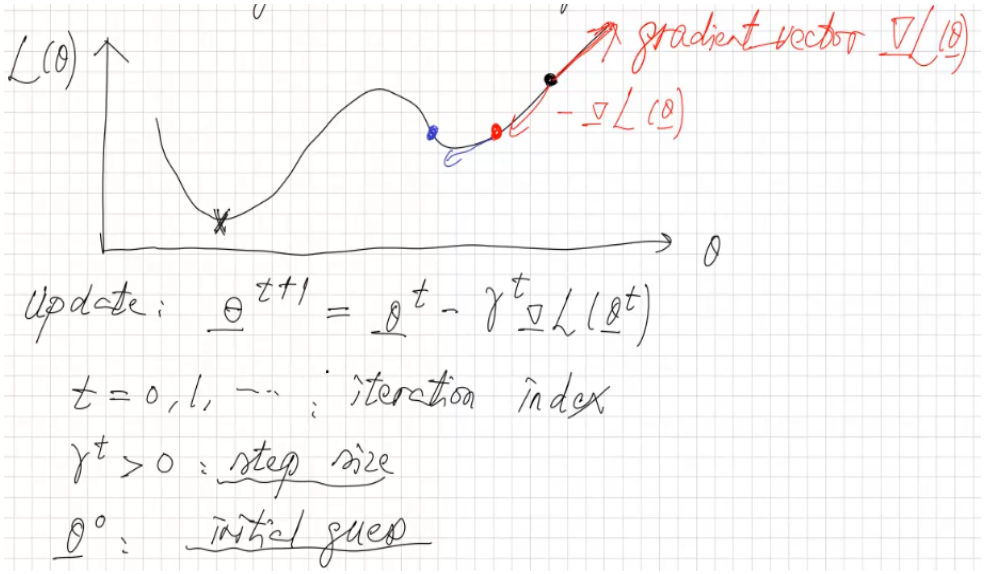



Abbildung 4.14: Gradient descent
calculation of the gradient vector $\nabla L(\underline{\theta})$ is non trivial

### 4.7.1. Chainrule of derivative (back propagation)

$$\frac{f(g(\theta))}{d\theta} = \frac{df}{dg} \cdot \frac{dg}{d\theta}$$

Layerindex L: $\frac{\partial L_{cost}(\underline{\theta})}{\partial w_{L,ij}} = \frac{\partial L_{cost}}{\partial \underline{x}_L} \cdot \frac{\partial \underline{x}_K}{\partial \underline{a}_L} \cdot \frac{\partial \underline{a}_L}{\partial w_{L,ij}} = \underbrace{\underline{\underline{J}}_L(\underline{x}_L) \cdot \underline{\underline{J}}_{\underline{x}_L}(\underline{a}_L)}_{\underline{\underline{J}}_L(\underline{a}_L)} \cdot \underline{\underline{J}}_{\underline{a}_L}(w_{L,ij}),$

Jacobi matrices see ch. 3.1
Notation: $\underline{\underline{J}}_{\underline{y}}(x) = \frac{\partial y}{\partial x}$
Layer $L-1$:
$\frac{\partial L(\underline{\theta})}{\partial w_{L-1,ij}} = \frac{\partial L}{\partial \underline{a}_L} \cdot \frac{\partial \underline{a}_L}{\partial \underline{a}_{L-1}} \cdot \frac{\partial \underline{a}_{L-1}}{\partial w_{L-1,ij}} = \underbrace{\underline{\underline{J}}_L(\underline{a}_L) \cdot \underline{\underline{J}}_{\underline{a}_L}(\underline{a}_{L-1})}_{\underline{\underline{J}}_L(\underline{a}_{L-1})} \cdot \underline{\underline{J}}_{\underline{a}_{L-1}}(w_{L-1,ij})$

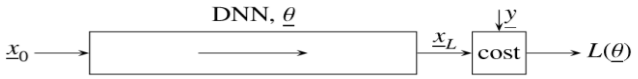
Layer 1: $\frac{\partial L(\underline{\theta})}{\partial w_{1,ij}} = \underline{\underline{J}}_L(\underline{a}_L) \cdot \underline{\underline{J}}_{\underline{a}_L}(\underline{a}_{L-1}), ..., \underline{\underline{J}}_{\underline{a}_2}(\underline{a}_1) \cdot \underline{\underline{J}}_{\underline{a}_1}(w_{1,ij})$



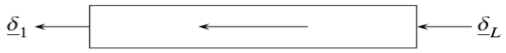


**Forward pass vs. backward pass in DNN**

**Forward pass** to calculate $\underline{x}_L$ from $\underline{x}_0$:




**Backward pass** or **backpropagation** of so called **error vectors** $\underline{\delta}_l^T := \mathbf{J}_L(\underline{a}_l) = \frac{\partial L(\underline{\theta})}{\partial \underline{a}_l} \in \mathbb{R}^{1 \times M_l}$:




For $l = L-1, \ldots, 1$

$$\underline{\delta}_l^T = \underline{\delta}_{l+1}^T \cdot \mathbf{J}_{\underline{a}_{l+1}}(\underline{x}_l)\mathbf{J}_{\underline{x}_l}(\underline{a}_l) = \square \cdot \square$$

$$\frac{\partial L(\underline{\theta})}{\partial w_{l,ij}} = \underline{\delta}_l^T \cdot \mathbf{J}_{\underline{a}_l}(w_{l,ij}) = \square \cdot \square$$

$$\frac{\partial L(\underline{\theta})}{\partial b_{l,i}} = \underline{\delta}_l^T \cdot \mathbf{J}_{\underline{a}_l}(b_{l,i}) = \square \cdot \square$$

Abbildung 4.15: Forward pass vs. backward pass

**Calculations for backpropagation on Slide 4-24 to 4-27**
more about omptimizer in ch.5
more about model in ch. 6-8
**Batch gradient descent**
calculate $\nabla L(\underline{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \nabla l(\underline{x}(n), \underline{y}(n); \underline{\theta})$
for the whole training set $D_{train}$
Difficult for too large N!
e.g. MNIST: 60.000 training smaples
ILSVRC: 1.2M training samples
$\rightarrow$ not enough ram to keep $D_{train}$ in memory
Solution: stochastic gradient descent (SGD), i.e. calculate $\nabla L$ and update $\underline{\theta}$ for each minibatch, a block of B samples from $d_{train}$:
$\underline{\theta}^{t+1} = \underline{\theta}^t - \nabla L((t); \underline{\theta})|_{\underline{\theta}=\underline{\theta}_t}$
$L(t; \underline{\theta}) = \frac{1}{B} \sum_{n=t'B+1}^{(t'+1)\cdot B} l(\underline{x}(n), \underline{y}; \underline{\theta})$
$B \in \mathbb{N}$ : minibatch size, small enough to fit in RAM
$N/B \; in \mathbb{N}$ : number of minibatches in $d_{train}$
$t = 0, 1, ..,$: iteration index
$t' = mod(t, \frac{N}{B}) \in \{0, 1, .., \frac{N}{B-1}$: minibatch index
$\nabla L(t; \underline{\theta})$ more noisy than $\nabla L(\underline{\theta})$
$\rightarrow$ stochastic gradient descents
**Evaluation during training:**
$\hat{\underline{\theta}}_k$ after k-th epoch: $k = 1, 2, ..$

**Technical performance metrics:** •training loss: $L(\hat{\underline{\theta}}_k)$ calculated form training set $D_{train}$
•test loss::$L(\hat{\underline{\theta}}_k)$ calculated form test set $D_{test}$
and objective performance metrics for classification
•**training error rate :** Error rate of $DNN(\hat{\underline{\theta}}_k)$ for $D_{train}$
•**test error rate :** Error rate of $DNN(\hat{\underline{\theta}}_k)$ for $D_{test}$
or: •training/test accuracy = 1- training/test error rate

## 4.8. 4.8 Implementation of DNN's in Python

•$60000/128)468,795 \approx 469$ mini-batches (slide 4-33)
•see python script KerasDemo



# Chapter 5: Advanced optimization techniques

Date: 15/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

## 5.1. Difficulties in optimization

2 Questions: •universal approximation: Existence of a NN for each task
Will you find it?
•Why was the conventional NN not successful?

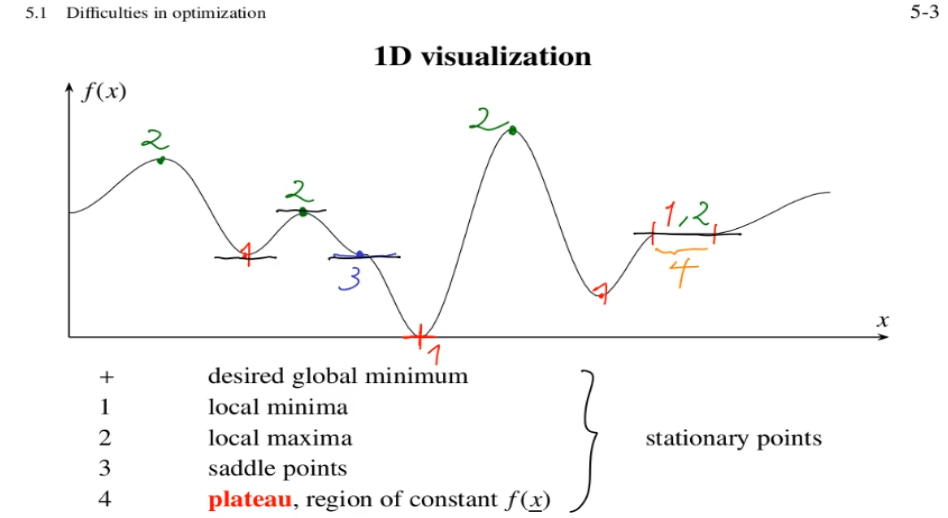



Abbildung 5.1: 1D Visualization Gradient Descent

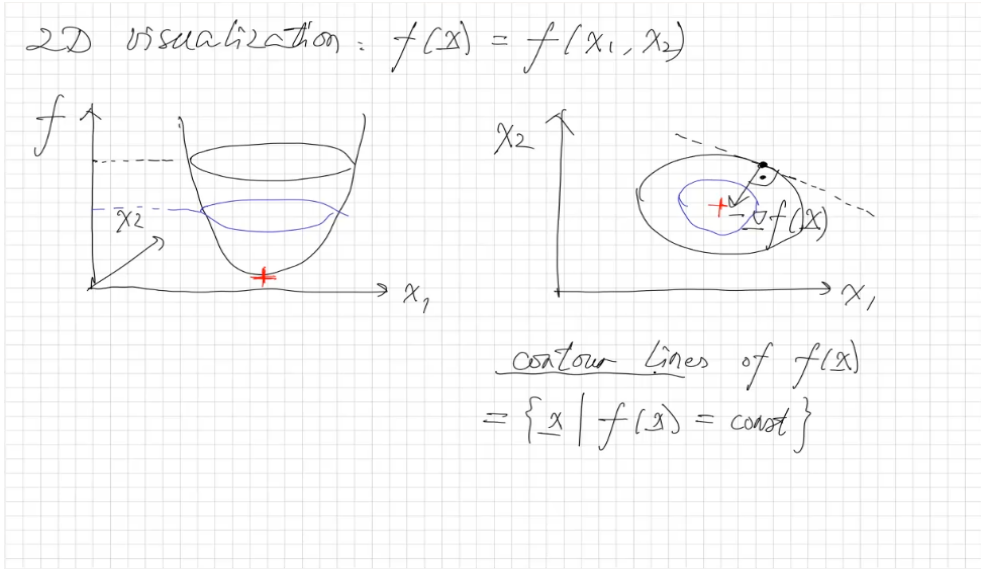


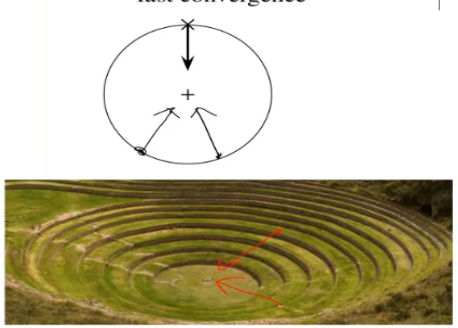

Abbildung 5.2: 2D Contour lines

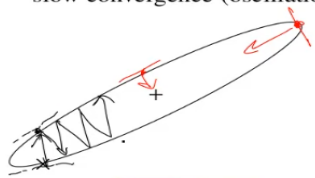


**D2) Ill conditioning**

**well conditioned** (circular) contour lines
equal curvatures
$-\nabla L(t; \underline{\theta})$ points to local minimum
fast convergence

**ill conditioned** (narrow) contour lines
strongly different curvatures
$-\nabla L(t; \underline{\theta})$ mostly points to wrong directions
slow convergence (oscillation)





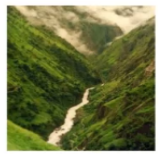


Abbildung 5.3: Ill conditioning

**D3) Saddlepoint and plateau** $\nabla L = \underline{0}$ no update of $\underline{x}^t$ because correction term is 0
in practice: $\nabla L \approx \underline{0} \rightarrow$ very slow convergence

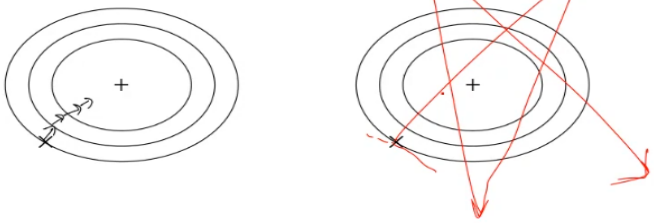




**D4) Sensitive to the choice of the step size**

too small step size $\gamma^t$
slow convergence

too large step size $\gamma^t$
oscillation and no convergence



- The optimum step size $\gamma^t$ depends on the cost function $L(t; \underline{\theta})$ and the current position $\underline{\theta}^t$ and is unknown in advance.

Abbildung 5.4: Sensitive Choice for step size

**Vanishing Gradient:**
Backpropagation of the error vector
$\Theta = \frac{\partial L(\underline{\theta}}{\partial \underline{a}_L} \in \Re^{1 \times M_L}$ in 4.7:
$\underline{\delta}_1^T = \underline{\delta}_L^T \cdot \underline{\underline{j}}_{\underline{a}_L}(\underline{a}_{L-1}), .., \underline{\underline{J}}_{\underline{a}_2}(\underline{a}_1)$, L Layers
$\underline{\underline{J}}_{\underline{a}_{L+1}}(\underline{a}_L) = \underline{\underline{J}}_{\underline{a}_{L+1}}(\underline{x}_l) \cdot \underline{\underline{J}}_{\underline{x}_L}(\underline{a}_L) = \underline{\underline{W}}_{l+1} \cdot diag(\phi_L'(\underline{a}_L))$
if $||\underline{\underline{J}}_{\underline{a}_{L+1}}(\underline{a}_L)|| < 1, \forall l$ matrix norm , then $||\underline{\delta}_L|| \rightarrow 0$
vanishing gradient $\rightarrow$ no update of $\underline{\theta}^T$ stop learning. With the multiplication of a lot of terms smaller than 1 the gradient turns to a very small number close to zero
•if $||\underline{\underline{J}}_{\underline{a}_{l+1}}(\underline{a}_l)|| > 1, \forall l$, then $||\underline{\delta}_l|| \rightarrow \infty$
exploding gradient $\rightarrow$ divergence
no problem for shallow NN's because only small number of layers
bo problem for deep NN's because of large number of layers
•vanishing gradient less serious for the last layers $(l \uparrow)$
•more serious for first layers $(l \downarrow)$

### 5.1.1. E5.1: sigmoid vs. ReLU

**a) sigmoid or tanh:**
$\phi(a) = \sigma(a) = \frac{1}{1 + e^{-a}} \in (0, 1)$

if $|a| >> 1 : \phi'(a) \approx 0$ due to saturation (shape of function )
$\rightarrow$ sigmoid bad for DNN die to vanishing gradient
**b) ReLU**
$\phi(a) = max(0, a)$
$\phi'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases} = u(a)$
$\rightarrow$ ReLu less serious for vanishing gradient

## 5.2. Momentum method

an improvement of SGD
•change from $\underline{\theta}^t$ to $\underline{\theta}^{t+1} : \Delta\underline{\theta}^t = \beta \cdot \Delta\underline{\theta}^{t-1} \underbrace{-\gamma^t \underline{\nabla} L(t;\underline{\theta})|_{\underline{\theta}=\underline{\theta}^t}}_{\text{gradient part}}$
•update: $\underline{\theta}^{t+1} = \underline{\theta}^t + \delta\underline{\theta}^t$
$0 \leq \beta \leq 1$ : **momentum factor**
$\beta = 0$ : SGD
$\beta > 0$ : recursive smoothing of $\Delta\underline{\theta}^t$
**Effect of the momentum:**
•reduce noise in stochastic gradient (D1)
•reduce oscillation and accelerate the convergence of the gradient method for ill conditioned contour lines(D2)

### 5.2.1. Nesterov Momentum

a variant of momentum method
$\Delta\underline{\theta}^t = \beta\Delta\underline{\theta}^{t-1} + \underbrace{(-\gamma^t \underline{\nabla} L(t;\underline{\theta})|_{\underline{\theta}=\underline{\theta}^t+\beta\Delta\underline{\theta}^t)}}_{\text{lookahead gradient at the predicted position } \underline{\theta}^{t-1}+\beta\Delta\underline{\theta}}$

## 5.3. 5.3 Learning rate schedule

How to choose the step size /learning rate $\gamma^t$?
Many possible schedules
**S1 constant learning rate:**
$\gamma^t = \gamma = const., \forall t$
But difficulty D4:
•$\gamma$ too small: slow convergence



•$\gamma$ too large: no convergence / oscillation around local minimum
$\rightarrow$ Need a trade-off between
•fast convergence at the beginning: $\gamma \uparrow$ (large step size)
•less oscillation afterwards : $\gamma \downarrow$ reduce stepsize with growing number of iterations: **learning rate decay**
**Weakness:**
•manual choice of $\gamma_0, t_0, c$
•same schedule for all parameters in $\underline{\theta}$
**Adaptive schedules:** •different schedules for different parameters in $\underline{\theta}$
•$\gamma^t$ calculated dynamically depending on $\underline{\theta}^t$
Slide 5-16 Adam very popular

## 5.4. Input and batch normalization

### 5.4.1. E5.3 A Perceptron

a) large offset : $\underline{x}(n) = \underset{offset}{\underline{x}_0} + \tilde{\underline{x}}(n), ||\underline{x}_0|| >> \tilde{\underline{x}}(n), \forall n$
$\rightarrow \underline{\underline{R}} \approx \underline{x}_0\underline{x}_0^T$, rank one
b) strongly different variances: e.g $\underline{x}(n) = \begin{bmatrix} large \\ small \\ \vdots \\ small \end{bmatrix} \sim \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \underline{e}_1, \forall n$
$\rightarrow \underline{\underline{R}} \sim \underline{e}, \underline{e}^T$, rank one
**AM:** $\underline{\underline{R}}$ has approximately rank one
$\rightarrow$ $\underline{\underline{R}}$ has one large eigenvalue $\lambda_1$ with eigenvector $\underline{v}_1 = \underline{x}_0$ or $\underline{v}_1 \sim \underline{e}_1$ and d-1 small eigenvalues
$\rightarrow$ very narrow contour lines
$\rightarrow$ slow convergence (D2)
**Need input normalization**, e.g.: zero-mean unit-variance normalization
For each $x_i(n) = [\underline{x}_0(n)]_i, 1 \leq n \leq N$
$x_i(n) \leftarrow \dfrac{x_i(n) - \mu_i}{\sigma_i}$ with estimated mean $\mu_i = \dfrac{1}{N}\sum_{n=1}^{N} x_i(n)$
variance $\sigma_i^2 = \dfrac{1}{N-1}\sum_{n=1}^{N}(x_i(n) - \mu_i)^2$ (N-1 because unbiased mean wanted) •normalization for the whole training set D
•equally for all channels $x_i$
Advantage:
•no ill conditioning (D2)

•faster convergence

### 5.4.2. Batch normalization

Input normalization: done once for the data set
Hidden layers: data distribution changes
•over layers
•over time during training those to points are called internal covariate shift
→ slower convergence
**Batch normalization(BN):** : like input normalization
•for any hidden layer
•for each mini batch of length B

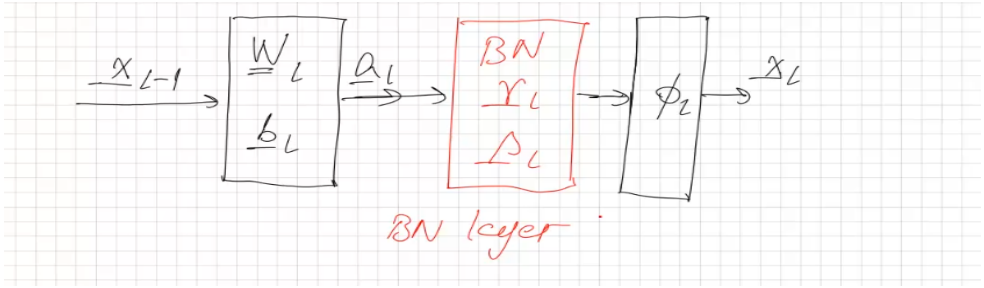



Abbildung 5.5: Sketch for mini batch normalization





## Batch normalization (1)

The **batch normalization** (**BN**) of the activation $\underline{a}_l(n)$ at layer $l$ for one minibatch $1 \le n \le B$ is defined as

$$\left[\underline{a}_l(n)\right]_i \leftarrow \gamma_{l,i}\frac{\left[\underline{a}_l(n)\right]_i - \mu_{l,i}}{\sqrt{\sigma_{l,i}^2 + \epsilon}} + \beta_{l,i}, \quad 1 \le n \le B,\ 1 \le i \le M_l.$$

It consists of two steps for each element $\left[\underline{a}_l(n)\right]_i$ of $\underline{a}_l(n)$:

- A zero-mean unit-variance normalization like input normalization.

$$\mu_{l,i} = \frac{1}{B}\sum_{n=1}^{B}\left[\underline{a}_l(n)\right]_i \quad \text{and} \quad \sigma_{l,i}^2 = \frac{1}{B-1}\sum_{n=1}^{B}\left(\left[\underline{a}_l(n)\right]_i - \mu_{l,i}\right)^2$$

  are the sample mean and variance of $\left[\underline{a}_l(n)\right]_i$ of this minibatch. $\epsilon$ is a small positive number (e.g. $10^{-5}$) to avoid division-by-zero. In contrast to the complete data set, $\sigma_{l,i}^2 \approx 0$ is likely for a short minibatch.

- Scale-and-offset $\gamma_{l,i}\square + \beta_{l,i}$ with two learnable parameters $\gamma_{l,i}$ and $\beta_{l,i}$ per neuron

Abbildung 5.6: Summary BN



## Batch normalization (2)

Why the second step $\gamma_{l,i}\square + \beta_{l,i}$?

- Allow a flexible data dynamic range for each neuron and does not change the expressiveness of the network.
  - If $\underline{\gamma}_l = \underline{\sigma}_l$ and $\underline{\beta}_l = \underline{\mu}_l$, batch normalization simplifies to the special case of no batch normalization. Normally, $\underline{\gamma}_l \neq \underline{\sigma}_l$ and $\underline{\beta}_l \neq \underline{\mu}_l$.
  - $\underline{b}_l$ is redundant due to $\underline{\beta}_l$ and can be omitted, i.e. $\underline{a}_l(n) = \mathbf{W}_l\underline{x}_{l-1}(n)$.
  - $\underline{\gamma}_l$ and $\underline{\beta}_l$ are learned from data like $\mathbf{W}_l$. Hence each neuron adjusts the individually optimum dynamic range of $\left[\underline{a}_l(n)\right]_i$ for the next nonlinear activation function $\phi_l(\cdot)$.
- Decouple the layers.
  - Batch normalization makes the dynamic range of one layer (partially) independent of the previous layers. This decouples the joint training of different layers to individual layers. It simplifies the learning, in particular for deep networks.
  - Batch normalization makes the surface of the cost function smoother.

Abbildung 5.7: Summary BN 2

## 5.5. Parameter initialization

SGD does a local search: The solution depends on the initial value $\theta^0$
$\Theta^0 = ??$ **2 Random onotialization of** $\underline{\underline{W}}_L$ , e.g. •normal or Gaussian distribution $N()$
$[\underline{\underline{W}}_L]_{ij} \sim$ i.i.d (independent identically distributed) $\sigma \cdot N(0,1) = N(0,\sigma^2)$
•uniform distribution $U(\cdot)$
$[\underline{\underline{W}}_L]_{ij} \sim$ i.i.dl $\sigma U(-1,1) = U(\sigma,\sigma), \sigma = const, \forall l,i,j$
$\rightarrow$ not optimal
3) He initialization
as 2) , but $\delta_l \sim \frac{1}{\sqrt{M_{L-1}}} \rightarrow$ constant activation flow after initialization
layer $l$ : $\underline{a}_l = \underline{\underline{W}}_l \cdot \underline{x}_{l-1} + \underline{b}_l \overset{b=0}{=} \underline{\underline{W}}_l \cdot \underline{x}_{l-1}$ or
$\underline{a} = \underline{\underline{W}} \cdot \underline{x}$,
$\underline{a} = [a_i] \in \mathbb{R}^{M_l}$
$\underline{\underline{W}} = [w_{ij}] \in \mathbb{R}^{M_l \times M_{l-1}}$
$\underline{x} = [x_i] \in \mathbb{R}^{M_{l-1}}$
$\rightarrow a_i = \sum_{j=1}^{M_{l-1}} w_{ij} \cdot x_j$
Assumptions:
•$x_i$ i.i.d zero mean , variance $\sigma_x^2$
•$w_{ij}$ i.i.d zero mean , variance $\sigma_w^2$
•$x_i$ and $w_{ij}$ independent
$\rightarrow$ •$E(a_i) = \sum_j E(w_{ij}) \cdot E(x_{ij}) = 0$
•$Var(a_i) = E(a_i^2) = E(\sum_j w_{ij} \cdot x_j)^2 = E(\sum_j \sum_k w_{ij} w_{ik} x_j x_k) = \sum_j \sum_k E(w_{ij} w_{ik} \cdot E(x_j x_k)) = \sum_{j=1}^{M_{l-1}} \sigma_w^2 \cdot \sigma_x^2 = M_{l-1} \cdot \sigma_w^2 \cdot \sigma_x^2$
**constant activation flow** $\equiv$ smooth information flow in the forward pass
$Var(a_i) = const., \forall i,l$
$\rightarrow \sigma_{w,l} \sim \frac{1}{\sqrt{M_{l-1}}}$, $M_{l-1}$ : fan-in
**4) Glorot initialization**
const. gradient flow in backward pass
$||\frac{\partial L}{\partial \underline{a}_l}|| = const, \forall l \rightarrow \sigma_{w,l} \sim \frac{1}{\sqrt{M_l}}, M_l$ : fan-out
compromise: $\sigma_{w,l} \sim \frac{1}{\sqrt{M_{l-1} + M_l}}$



## 5.6. Improved (network)-model

A) Better activation functions $\phi(\cdot)$, e.g.
•ReLU instead of sigmoid
•leaky reLu instead of ReLU
**B) skip connections, shortcuts , residual network (ResNet)**

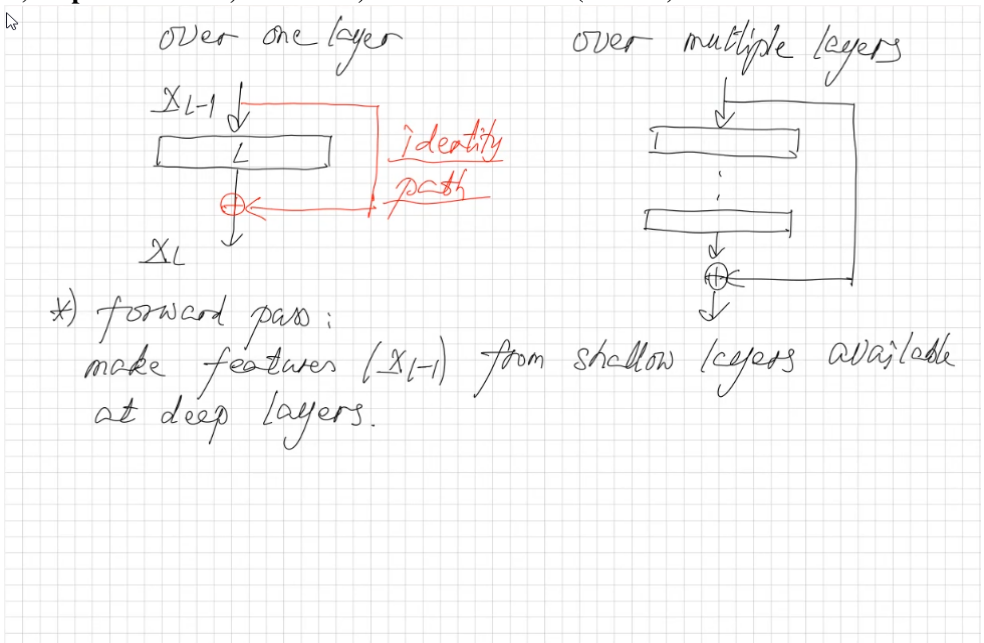



Abbildung 5.8: Layerskip / shortcuts

•backward pass<. no vanishing gradient over identity path.
$\underline{x}_l = \phi_l(\underline{\underline{W}}\underline{x}_{l-1} + \underline{b}_l) + \underline{x}_{l-1}$
$\frac{\partial \underline{x}_l}{\partial \underline{x}_{l-1}} = \frac{\partial \phi_l(\cdots)}{\partial \underline{x}_{l-1}} + \underline{\underline{I}}$
c) Many other architecture improvements see ch.10

# Chapter 6: Overfitting and regularization

Date: 25/05/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

## 6.1. Model capacity and overfitting / underfitting

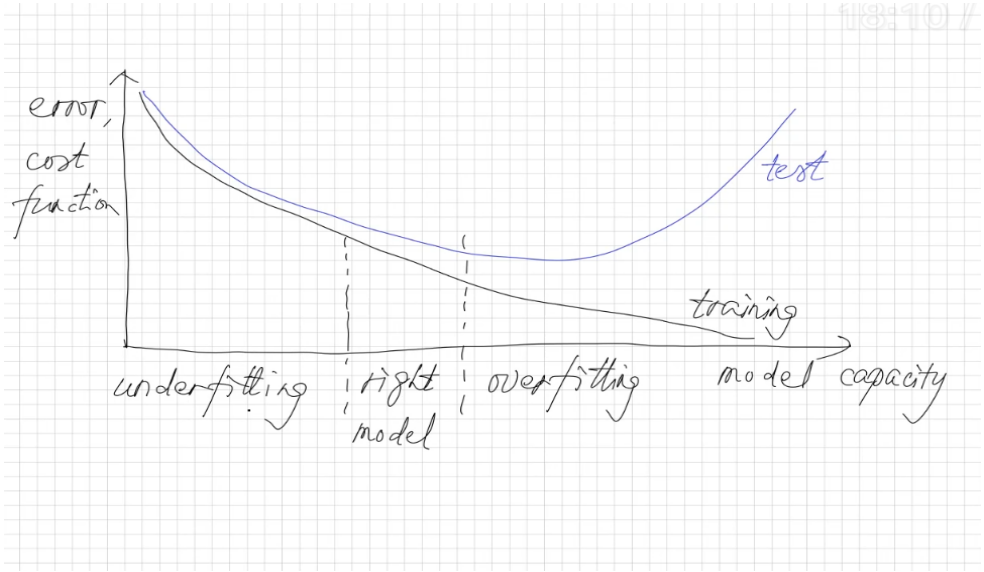



Abbildung 6.1: Model fitting curve

## 6.2. Weight norm penalty

change on cost function:
•old cost function: $L(\underline{\theta})$
•New regularized cost function: $L_r(\underline{\theta}) = L/\underline{\theta} + \sum_{l=1}^{L} \lambda_i P(\underline{\underline{W}}_L)$
$\lambda \geq 0$ : regularized parameters
$P(\underline{\underline{W}}_L) \geq 0$ : penalty terms, penalize $\underline{\theta}$ with large $P(\underline{\underline{W}}_L)$
$\rightarrow$ A compromise between $\min_{\underline{\theta}} L(\underline{\theta})$ and the min $\min_{\underline{\theta}} P(\underline{\underline{W}}_L)$
$\lambda_L$ determine the relative contributions of $L(\underline{\theta}), P(\underline{\underline{W}}_L)$.



$\lambda_L = 0, \forall L$ : no regularization
common choice of $P(\underline{\underline{W}}_L)$:
•a) l2-regularization: we use $l_2 - norm$ of $vec(\underline{\underline{W}}_L)$
$P(\underline{\underline{W}}_L) = ||vec(\underline{\underline{W}}_L)||_2^2 = \sum_i \sum_j W_{L,ij}^2 \rightarrow$ prefer $\underline{\theta}$ with small weight energy
$\rightarrow$ better generalization, see E6.1
Bias vector $\underline{b}_l$ : no amplification effext of $\underline{x}_{l-1} \rightarrow$ no need for regularization
**Mathematical analysis:**
$L_r(\underline{\theta}) = L(\underline{\theta}) + \lambda ||\underline{\theta}||^2$ for simplicity
$\underline{\nabla} L_r(\underline{\theta}) = \underline{\nabla} L(\underline{\theta}) + 2 \cdot \lambda \underline{\theta}$
$\underline{\theta}^{t+1} = \underline{\theta}^t - \gamma^t \underline{\nabla} L_r(\underline{\theta}^t)$
$\underbrace{(1 - 2\lambda\gamma^t)}_{0<<factor<1} \underline{\theta}^t - \gamma^t \underline{\nabla} L(\underline{\theta}^t)$
l2-regularization leads to **weight decay**
**b) l1-regularization:** use l1-norm of $vec(\underline{\underline{W}}_L)$
$P(\underline{\underline{W}}_L) = ||vec(\underline{\underline{W}}_L)||_1 = \sum_i \sum_j |W_{L,ij}|$
$\rightarrow$ prefer sparse $\underline{\underline{W}}_L$ with many zero elements

## 6.3. Early stopping

change on optimizer:

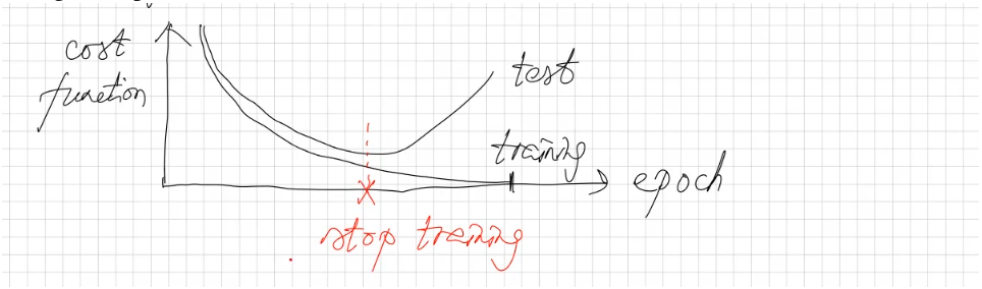



Abbildung 6.2: Early stopping

## 6.4. Data augmentation

change on dataset, training with $\infty$ many training data $\rightarrow$ no over fitting, in practice, training set is limited in size

**data augmentation:** Generation of artificial realistic training samples:

## 6.5. Ensamble learning

change dataset / model / cost function / optimizer



### Ensemble learning

**Ensemble learning** is a model averaging method to reduce the test error by combining an ensemble of models:

- train different (independent) models for the same task
- combine these models to reduce the test error
  - regression: average of the model outputs
  - classification: voting of the model outputs, e.g. 3× cat and 1× dog

It is unlikely that all models will make the same errors on the test set. Hence the averaged model is more robust.

The different models can be trained by using

- different subsets of the training set or
- different model architectures or
- different cost functions or
- different optimizers or
- combinations of them.

Abbildung 6.3: Ensamble learning explained

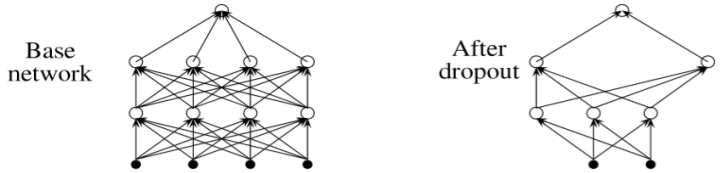
## 6.6. Dropout

change on model
An implicit ensemble method





### Dropout (1)




**Training**

- For each minibatch, **dropout** randomly removes some neurons in layer $l$ of a base network with a probability $d_l$, the **dropout rate**.
- This results for each minibatch in a random subnetwork for solving the same task.[2]

**Inference**

- No dropout.
- All outgoing weights of neurons in layer $l$ are weighted by $1-d_l$ to correct the large fan-in to the neurons (number of inputs) of the next layer.

Abbildung 6.4: Dropout explained

## Hyperparameters

What are they?

- In contrast to the model parameters $\underline{\theta}$ (weights and biases) to be learned from training data, **hyperparameters** $\underline{\eta}$ are configuration parameters of a machine learning algorithm which are not adapted during training.
- $\underline{\eta}$ is chosen before learning and remains fixed. It controls, together with $\underline{\theta}$, the behavior of the model $f(\underline{x}; \underline{\theta}, \underline{\eta})$, see next slide. Hence, they need an optimization.

Why are they not adapted?

a) Hyperparameters are often discrete valued (e.g. number of layers/neurons, type of activation function). Gradient descent is not suitable for this kind of integer optimization.

b) The training error is a monoton function of some hyperparameters, in particular those controlling the model capacity. An optimization of these hyperparameters would always maximize the model capacity (e.g. more layers, more neurons) resulting in overfitting. The training set alone is not suitable for hyperparameter optimization.

Abbildung 6.5: Hyperparameters explained

Solution for b): Use validation data set D
up to now separating dataset in 2 for training and test, but now split in three parts training set $D_{train}$ (large). test set $D_{test}$(small) and validation set $D_{val}$ (small, same size as test)





## Training set, validation set and test set

**Training set** $D_{\text{train}}$: It is used for training the model, i.e. learning the model parameters $\underline{\theta}$ (weights and biases) for a fixed hyperparameter vector $\underline{\eta}$.

**Validation set** $D_{\text{val}}$: It is never used in training. It is reserved for optimizing the hyperparameter parameters $\underline{\eta}$.

**Test set** $D_{\text{test}}$: It is never used in training and hyperparameter optimization. It is used to calculate the test error of the trained ($\underline{\theta}$) and tuned ($\underline{\eta}$) model $f(\underline{x}; \underline{\theta}, \underline{\eta})$ to exam its generalization capability. The motivations of using the test set are

- to avoid overfitting of $\underline{\theta}$ to the training set and
- to avoid overfitting of $\underline{\eta}$ to the validation set.

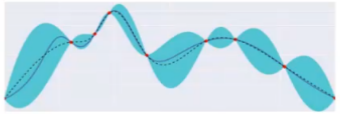
**E6.4: Early stopping**
Early stopping from ch. 6.3 is a hyperparameter optimization to determine the optimum value of the hyperparameter, the number of epochs. This is only possible by using a validation set because the training error often decreases continuously as the number of epochs increases.

Abbildung 6.6: Summary of data sets

**Training:** $\underline{\theta}$ and hyperparameter and $\underline{\eta}$
for $\underline{\eta}$ = ..... learn $\underline{\theta}$ of $f(\underline{x}; \underline{\theta}; \underline{\eta})$ frpm $D_{train}$
calculate validation error $(\underline{\eta})$ on $D_{val}$
min validation error $(\underline{\eta})$
calculate test error of $f(\underline{x}; \underline{\theta}; \underline{\eta})$ on $D_{test}$

**Hyperparameter optimization approaches**

- **Grid search** A blind exhaustive search on a grid in the hyperparameter space, e.g. $M_1 \in \{100, 150, 200\}$, $\phi_l \in \{\text{sigmoid, ReLU}\}, \ldots$
  +) simple,
  −) time-consuming, especially for many hyperparameters and a fine grid
- **Bayesian optimization** Treat hyperparameter tuning as an optimization problem
  – probabilistic model for the posterior of the cost function based on Bayes rule (and Gaussian distribution)
  – samples of hyperparameters and cost function: iteratively refine the model
  – posterior: which regions of the hyperparameter space are worth exploring
  – choose next hyperparameter value based on posterior



— target function, - - - predicted function,
• samples, ▬ confidence interval

https://github.com/fmfn/BayesianOptimization:
A Python package ready for use

Abbildung 6.7: Approaches to optimize hyperparameters



# Chapter 7: Convolutional neural networks (CNN)

Date: 15/06/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

ch. 4: dense networks
•consisting of dense layers only
•but rarely used. CNN widely used.
**Why?**
•Drawbacks on slide 7-1
Solution: CNN, consisitng of many convolutional layers, based on convolution operation
•good for local feature learning
•more efficent, i.e. low complexity

## 7.1. Convlutional layer

supports various datatypes:

| input data type: | $\underline{\underline{X}}_{l-1}, \underline{\underline{A}}_l; \underline{\underline{X}}_l$ | $\underline{\underline{W}}_l$ |
|---|---|---|
| 1D signal waveform | 2D matrix | 3D tensor |
| 2D image | 3D tensor | 4D tensor |
| 3D data cube (ct image ) | 4D tensor | 5D tensor |

One dimension more: multiple input/output signals/images/feature maps/channels
e.g. RGB: 3 times 2D images → 3D tensor
**2D convolutional layer l:**
Dimension refers to input data shape

$$\underline{\underline{X}}_{l-1} \rightarrow \begin{matrix} \underline{\underline{W}}_l \\ b_l \\ \phi_l \end{matrix} \rightarrow \underline{\underline{X}}_l$$

•input tensor $\underline{\underline{X}}_{l-1}, M_{l-1} \times N_{l-1} \times D_{l-1}$
$D_{l-1}$(channel depth ) 2D feature maps of size $M_{l-1} \times N_{l-1}$
•output tensor $\underline{\underline{X}}_l, M_l \times N_l \times D_l, \underline{\underline{X}}_l = \phi_l(\underline{\underline{A}}_l)$ $D_l$ 2D feature maps of size $M_l \times N_l$
•Kernel tensor: $\underline{\underline{W}}_l, K_l \times K_l \times D_{l-1} \times D_l$
$K_l$ : kernel width: $D_l$ 3D impuls responses/ filters /kernels of the size $K_l \times K_l \times D_{l-1}$
•bias vector: $b_l, D_l \times 1$
one bias for one output channel
•activation tensor: $\underline{\underline{A}}_l, M_l \times N_l \times D_l$

$$[\underline{\underline{A}}_l]_{mno} = \sum_{i=1}^{K_l} \sum_{j=1}^{K_l} \sum_{d=1}^{D_{l-1}} [\underline{\underline{W}}_l]_{ijdo} \cdot [\underline{\underline{X}}_{l-1}]_{m+i-1,n+j-1,d} + [\underline{b}_l]_o$$

index (mno, i=1, j=1 , ij, m+i-1, n+j-1 ) spatial correlation, called convolution in CNN
index (d=1 , d) sum of all input feature maps
index (o,o,o): one of $D_l$ output channel
$1 \leq m \leq M_l = M_{l-1} - K_l + 1$
$1 \leq n \leq N_l = N_{l-1} K_l + 1$
$1 \leq o \leq D_l$
output size reduced by $K_l - 1$ in each dimension
•$\phi_l(\cdot)$ : activation function elementwise

7.1 Convolutional layer 7-5

**2D convolutional layer $l$**

- The input $\mathbf{X}_{l-1}$ contains $D_{l-1}$ 2D feature maps (or channels) of size $M_{l-1} \times N_{l-1}$.
- The layer contains $D_l$ kernels to calculate $D_l$ 2D feature maps of size $M_l \times N_l$.
- Each kernel contains $D_{l-1}$ 2D filters of size $K_l \times K_l$, each performing a **sliding window** convolution or correlation.

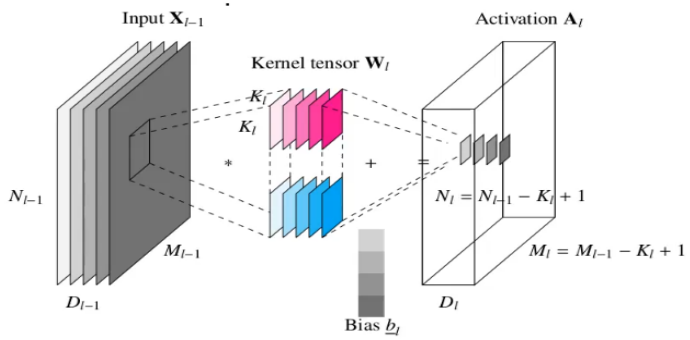



Abbildung 7.1: Visualization of CNN layer

## 7.1.1. Properties of convolutional layer

Typical kernel size $K_l \times K_l : 3 \times 3, 5 \times 5, ...m$ but also $1 \times 1$
typical number of input channels $D_l : 1 \sim 10 \sim 100$
Number of parameters:
$N_{p,l} = \underbrace{K_l^2 D_{l-1} D_l}_{\underline{W}_l} + \underbrace{D_l}_{\underline{b}_l} \approx K_l^2 D_{l-1} D_l$ quite small, independent on input size $M_{l-1} \times N_{l-1}$
Number of multiplications:
$N_{x,l} = \underbrace{M_l \cdot N_l \cdot D_l}_{\text{elem in } \underline{A}} \cdot K_l K_l D_{l-1} \approx M_l N_l N_{p,l}$, quite large , depends on input size.
•p1) sparse connection:



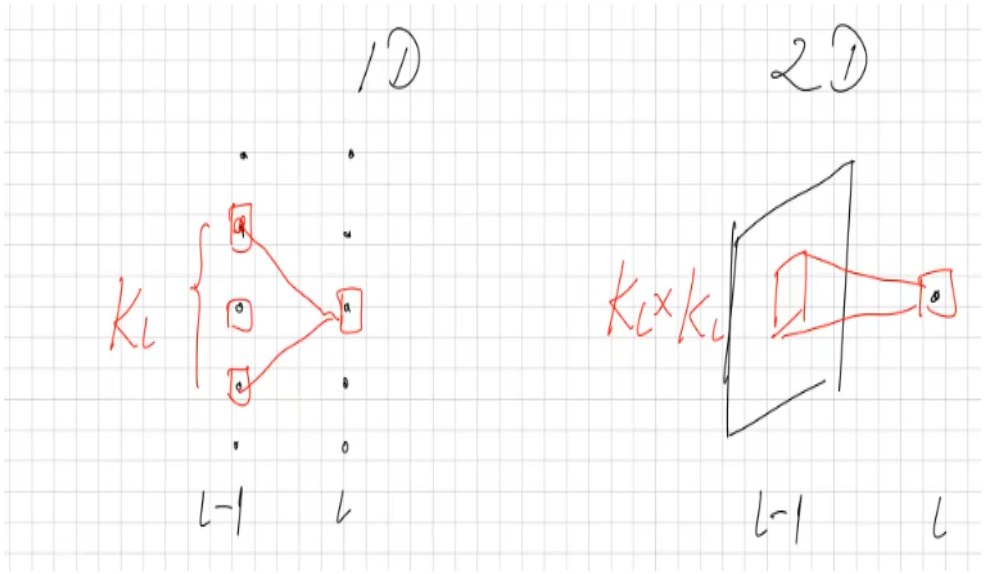



Abbildung 7.2: Illusatration sparse connection

small receptive field: $K_l or K_l \times K_l \equiv$ focus on local input patterns

in DNN:

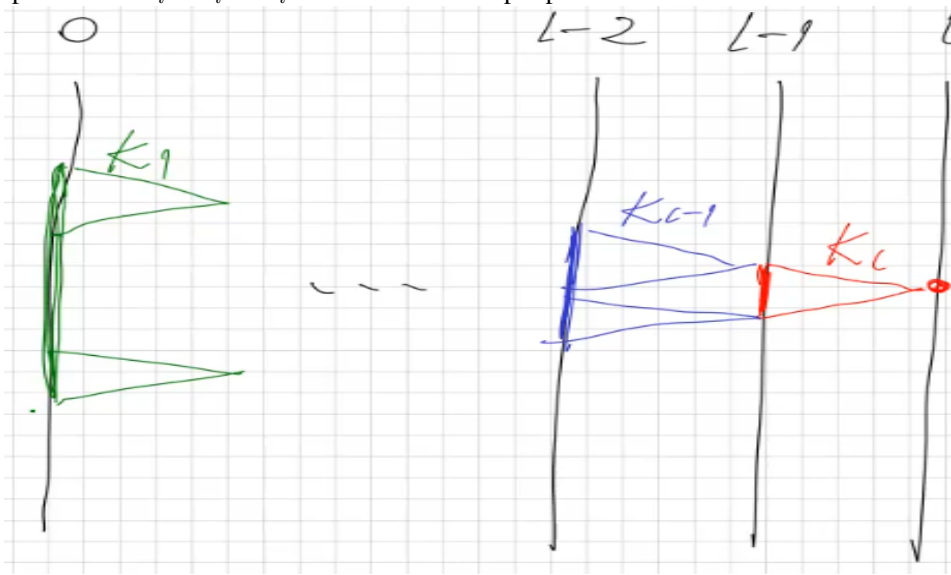



Abbildung 7.3: Illustraion receptive field

a neuron in layer l has te receptive field width : $K_l +, ..., + K_l - (l-1)$ for the input layer
i.e. a neuron in a deep layer (l ↑) can still be indirectly connected to all input neuron

p2) parameter sharing: the same kernel for all output neurons
•p1) + p2) give the following advanteges
•reduced computational complexity reduced memory complexity
•reduced model capacity and reduced overfitting
•p3) translation - equivariant ≡ shift-invariant
translation /shift of $\underline{\underline{X}}_{l-1}$ → same translation/shift of $\underline{\underline{X}}_{l}$
•p4) one channel/feature-map contains only one feature, e.g. only horizontal edges
→ need multiple channels/feature maps ($D_l > 1$)
•p5) $M_l = M_{l-1} - K_l + 1$ : images become smaller.

## 7.2. Modified convolutions

to:
•reduce complexity
•increase the receptive field
•...
**Minor modifications**
•Padding slide 7-9
•Stride slide 7-10 •Dilated convolution slide 7-11
**outputsize:**
Standard $M_l = M_{l-1} - K_l + 1$
including all modifications:
$P \geq 0$ : padding , 0
$s \geq 1$ : stride, 1
$D \geq 1$ : dilation , 1
$\left\lfloor \frac{M_{l-1} + 2P - (K_l - 1)D - 1}{S} \right\rfloor + 1$
Major modifications on $\underline{\underline{W}}_{l}$ •Standard convolution using padding:
$M \times N \times D_{l-1} K \times \times K \times D_{l-1} \times D_l M\ times N \times D_l$
3D joint spatial-channnel processing $\sum_i \sum_j \sum_d$
→ high complexity $N_{x,1} = MNK^2 D_{l-1} D_l$ To reduce the complexity:
•$1 \times 1$ convolution, i.e. K=1 Slide 7-12
•additional higher nonlinearity due to additional $\phi(\cdot)$
Idea: replace 1 layer of standard convolution by 2 layers of simpler convolutions
•$M \times N \times D_{l-1} \overset{1\times1\times D_{l-1}\times\bar{D}_{l-1}}{\longrightarrow} M \times N \times \bar{D}_{l-1}, \bar{D}_{l-1} < D_{l-1}$
•$M \times N \times \bar{D}_{l-1} \overset{K\times K\times\bar{D}_{l-1}\times D_l}{\longrightarrow} M \times N \times D_l\ N_{x,2} = MN - 1^2 \cdot D_{l-1} + \bar{D}_{l-1} + MN \cdot K^2 \cdot \bar{D}_{l-1} D_l$



$\rightarrow \frac{N_{x,2}}{N_{x,1}} = \frac{D_{l-1}}{K^2 D_l} + \frac{\bar{D}_{l-1}}{D_{l-1}} \approx \frac{\bar{D}_{l-1}}{D_{l-1}} < 1$
•**depth-wise separable convolution:**
separable spatial (2D) and channel (1D) processing
•$M \times N \times D_{l-1} \overset{K\times K\times 1\times D_{l-1}}{\longrightarrow} M \times N \times D_{l-1}$
depth wise (2D) convolution for each channel
•$M \times N \times D_{l-1} \overset{1\times1\times D_{l-1}\times D_L}{\longrightarrow} M \times N \times D_l$ point wise (1D) convolution = $1 \times 1$ convolution
$N_{x,3} = MNK^2 \cdot 1 \cdot D_{l-1} + MN \cdot 1^2 \cdot D_{l-1} D_l$
$\frac{N_{x,3}}{N_{x,1}} = \frac{1}{D_l} \frac{1}{K_l^2} \approx \frac{1}{K_l^2}$ for $D_l >> K_l^2$
•Comparison Slide 7-13

## 7.3. Pooling and unpooling layer

2D **pooling layer** with stride $s \in \mathbb{N}$

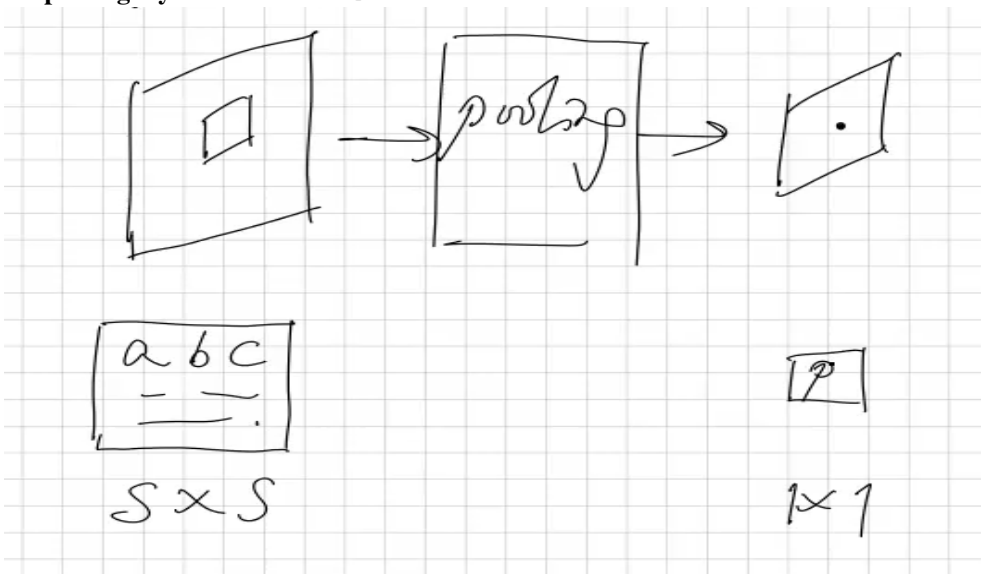



Abbildung 7.4: pooling layer visualization

•max pooling: $p = max()$ of a,b,c,...
•mean pooling $p = mean()$ of a,b,c,...
•l2-norm pooling: $p = l_2 - norm$ of a,b,c,...
widely used; 2x2 max pooling with s=2
Nonlinear operation
Effects explained on slide 7-15

**2D unpooling layer** e.g. 2x2
Corresponds to upsampling:
$a \to \begin{bmatrix} a & a \\ a & a \end{bmatrix}$
upsampling to restore the original image after size pooling

## 7.4. Deconvolutional layer

deconvolutional layer $\neq$ deconvolution, bad name
other names= fractionally strided convolution or transposed convolution or learnable upsample
**Goals:**
•replaces unpooling layer
•increase image resolution without changing the object shape in the image
**deconvolutional layer**
•zero insertion
•convolution (smoothing) with learnable kernel

| | **in deep learning** | **in signal processing** |
|---|---|---|
| $\underline{\underline{X}}_{l-1}, \underline{\underline{X}}_l$ | feature maps | input/output signal |
| $\underline{\underline{W}}_l$ | kernel | impulse response |
| $\sum_i \sum_j w_{ij} x_{m+i,n+j}$ | convolution | correlation |
| $x(n-n_0) \to y(n-n_0)$ | translation-equivariant | shift-invariant |
| $x(n-n_0) \to y(n)$ | translation-invariant | - |
| | padding | zero initialization |
| | stride | downsampling of output |
| | dilated convolution | polyphase downsampling of input |
| reverse convolution | - | deconvolution |
| | deconvolutional layer | lernable upsampling |

## 7.5. Flatten layer

$\underline{\underline{X}} \in \mathbb{R}^{M \times N \times D}$ flatten returns long single column $vec(\underline{\underline{X}}) \in \mathbb{R}^{MND}$
as an interface between convolutional layers and dense layers
conv. layer(feature learning) $\to$  $\to$ dense layer $\to$ output layer (combine and interpret high-level features and make decisions in the output layer)



## 7.6. Global average pooling layer

**GAP**: efficient alternative for flatten layer
$\underline{\underline{X}} = [x_{ij}] \in \mathbb{R}^{M \times N \times D} \to GAP \to \left[ \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} x_{ijd} \right]_{1 \le d \le D} \in \mathbb{R}^D$
•reduche each feature map/channel to a single value
•as an interface between convolutional and dense layers like the flatten layer
•much less coefficients/parameters for the dense layer after the GAP layer
•suitable for large DNN: reduce complexity

## 7.7. Architecture of CNNs

A CNN consists of typically
•convolutional layers
•optional max pooling layers
both types are used for hierarchical feature learning
•flatten or GAP layer to change the type of layer from convolutional to dense layer to make a decision
•dense layers for final classification or regression
•unpooling or deconvolutional layer is used for image segmentation to get the same output size as the input
Examples on slide 7-19 following.
Why deep CNN?
•large model capacity for difficult tasks
•multi-level (hierarchical) representation / features of the input image

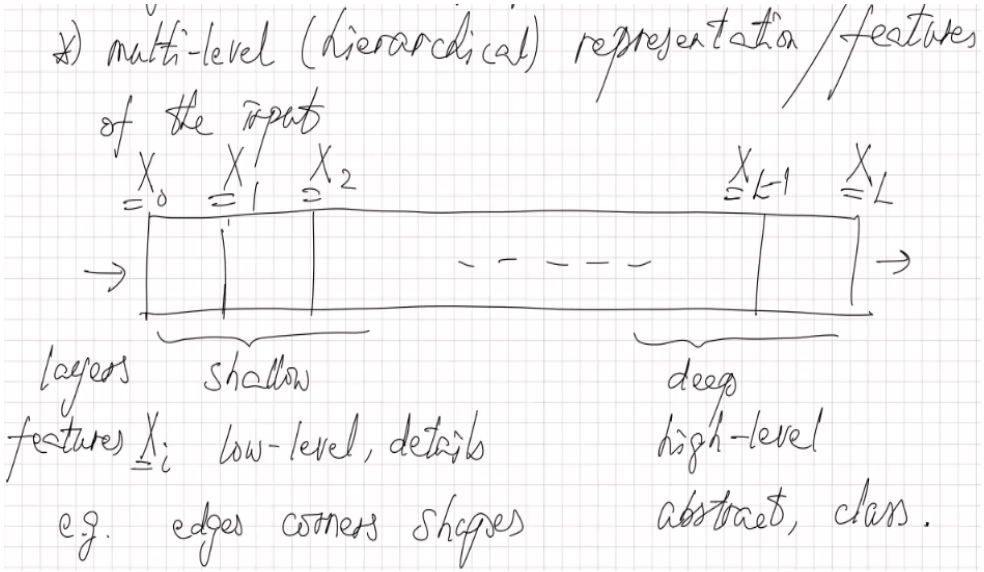



Abbildung 7.5: Visualization deep CNN



# Chapter 8: Reccurent/recursive neural networks (RNN)

Date: 15/06/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

NN with feedback.
Dense network(ch. 4) and CNN(ch. 7): feedforward

## 8.1. Recruent layer and recurrent neural network

**Recurrent layer:** l: $M_l$ recurrent neurons

$x_{l-1}(n) \rightarrow \begin{bmatrix} \underline{\underline{W}}_{l,x} \\ W_{l,s} \\ \underline{b}_l \end{bmatrix} \rightarrow x_l(n) = \underline{s}_l(n:)$ state

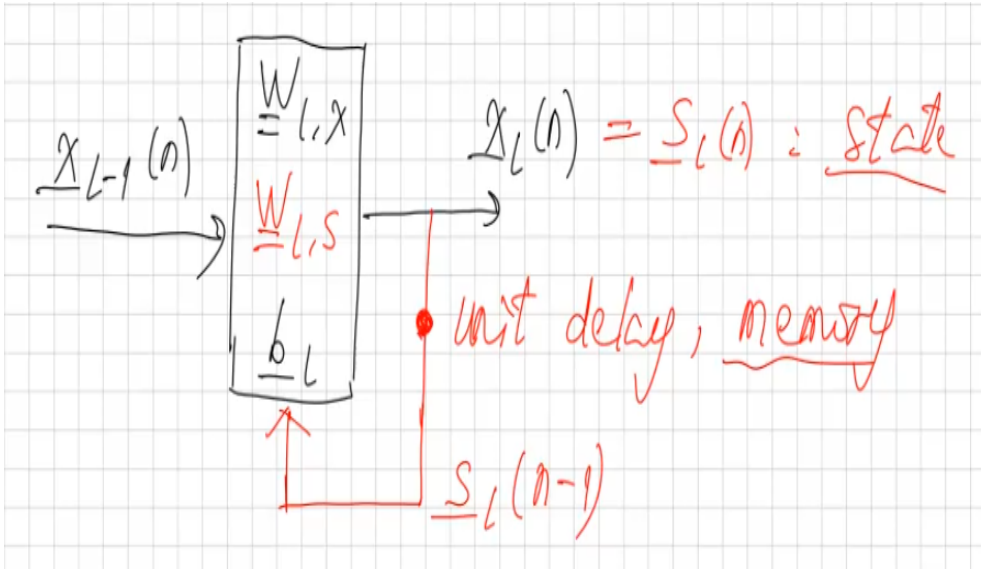



Abbildung 8.1: Recurrent layer visualization

$n = 1, 2, ..., B$ : time index in one minibatch
$x_{l-1}(n) \in \mathbb{R}_{M_l-1}$ : input at time $n$
$\underline{a}_l(n) = \underline{\underline{W}}_{l,x} \cdot \underline{x}_{l-1}(n) + \underline{\underline{W}}_{l,s} n - 1 + \underline{b}_l \in \mathbb{R}_{\mathbb{M}_<}$: activation
$\underline{\underline{W}}_{l,x} \in \mathbb{R}_{M_l \times M_{l-1}}, \underline{\underline{W}}_{l,s} \in \mathbb{R}^{M-l \times M_l}$ : weight matrices
$\underline{b}_l \in \mathbb{R}^{M_l}$ : bias

$\underline{x}_l(n) = \underline{s}_l(n) = \phi_l(\underline{a}_l(n)) \in \mathbb{R}^{M_l}$: output, memory state
$\phi_l(\cdot)$ activation function
$\underline{s}_l(0)$ : initial value, often zero
$\equiv$ first order recursive nonlinear filter for vector signal
$\underline{\underline{W}}_{l,s} = 0$ $\phi_l(a) = a$
feedforward dense layer linear first oder recursive filter IIR(1,0)
often:
•$\underline{\underline{W}}_{l,x}$ full matrix $\equiv$ dense layer
•$\underline{\underline{W}}_{l,1}$ diagonal $\equiv$ single neuron feedback
Visu on Slide 8-6
**Recurent neural netwrok (RNN)**
at least one recurrent layer.
Training of RNN:
•SGD: $\underline{\Theta}^{t+1} = \underline{\Theta}^t - \gamma^t \underline{\nabla} L(t; \underline{\Theta})|_{\underline{\Theta}=\underline{\Theta}^t}$
•backpropagation of derivatives thorugh layers
•for a parameter $\theta$ of a recurrent lyer: additional backpropagation through time (BPTT) $\equiv$ backpropagation throught unfoldet graph along the time axis
$\equiv$ chain rule and product rule of derivative
$\frac{\partial(f(\theta) \cdot g(\theta))}{\partial\theta}$
•Slide 8-10 $\underline{\underline{W}}_2$ is wrong before loss



## 8.2. Bididrectional recurrent neural network(BRNN)

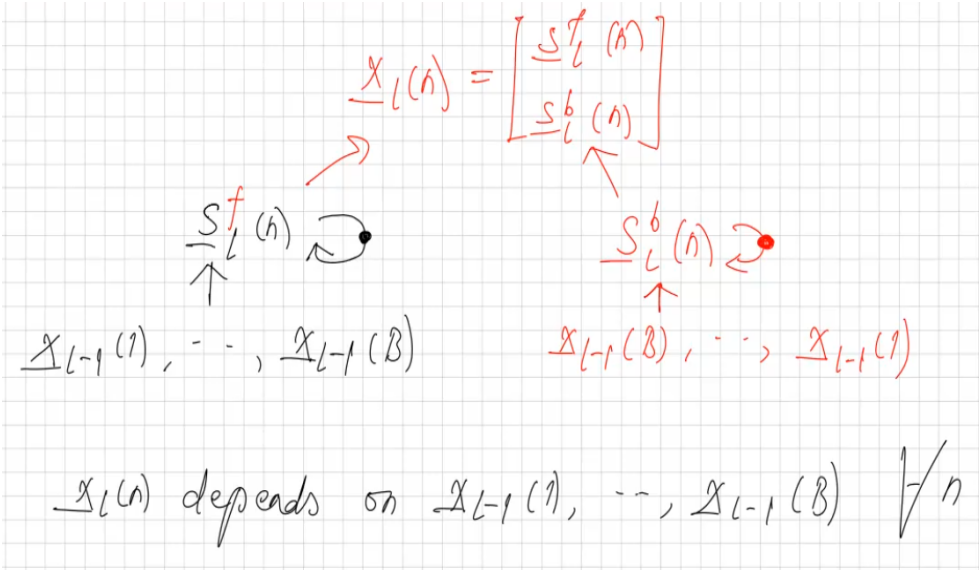



Abbildung 8.2: Graph BRNN

**Forward pass of a BRNN for one minibatch**

Given: $\underline{x}_{l-1}(n),\ 1 \le n \le B$
Initial values: $\underline{s}_l^f(0), \underline{s}_l^b(0)$
FOR $n = 1, 2, \ldots, B$ DO
$\underline{s}_l^f(n) = \phi_l(\mathbf{W}_{l,x}^f \underline{x}_{l-1}(n) + \mathbf{W}_{l,s}^f \underline{s}_l^f(n-1) + \underline{b}_l^f)$
$\underline{s}_l^b(n) = \phi_l(\mathbf{W}_{l,x}^b \underline{x}_{l-1}(B-n+1) + \mathbf{W}_{l,s}^b \underline{s}_l^b(n-1) + \underline{b}_l^b)$
$\underline{x}_l(n) = \begin{bmatrix} \underline{s}_l^f(n) \\ \underline{s}_l^b(B-n+1) \end{bmatrix}$

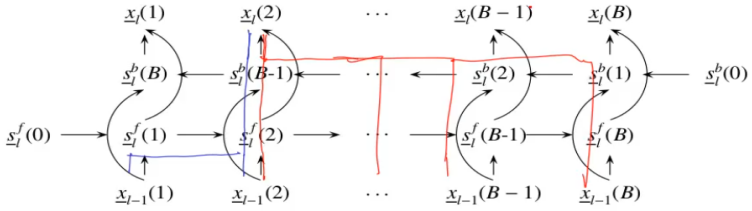



Abbildung 8.3: Unfolded graph BRNN

Training: backpropagation of derivatives through unfolded bidirectional graph.

## 8.3. Long short-term memory (LSTM)

RNN difficult to train due to vanishing/exploding gradient. CH.4.7: large number of layers(L) $\rightarrow$ long chain rule over layers
RNN: large number of time recursions(B) $\rightarrow$ long chain rule over time in addition
Difficult choice of B: B $\downarrow \rightarrow$ noisy gradient ; B $\uparrow \rightarrow$ vanishing/exploding gradient
Solution: LSTM

A **long short-term memory** (**LSTM**) cell/neuron/unit/block replaces a normal recurrent neuron in RNN. It contains[3]

- a memory storing the state $s(n)$ at time $n$ like a recurrent neuron and
- three multiplicative gates, the **input/forget/output gate**, which control the write/reset /read operation of the memory state.

### A LSTM layer $l$ (1)

containing $M_l$ LSTM cells:

| | |
|---|---|
| $\underline{x}_{l-1}(n) \in \mathbb{R}^{M_{l-1}}$ | layer input at time $n$ |
| $\underline{x}_l(n) \in \mathbb{R}^{M_l}$ | layer output at time $n$ |
| $\underline{s}_l(n) \in \mathbb{R}^{M_l}$ | memory state at time $n$, $\underline{s}_l(n) \neq \underline{x}_l(n)$ in contrast to RNN |
| $\underline{i}_l(n) \in \mathbb{R}^{M_l}$ | **input gate signal** at time $n$ |
| $\underline{f}_l(n) \in \mathbb{R}^{M_l}$ | **forget gate signal** at time $n$ |
| $\underline{o}_l(n) \in \mathbb{R}^{M_l}$ | **output gate signal** at time $n$ |

Abbildung 8.4: LSTM cell overview



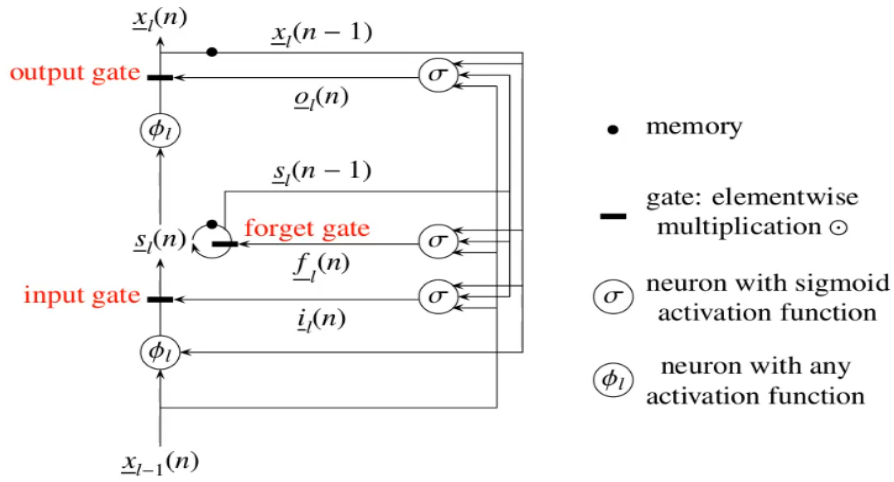



Abbildung 8.5: LSTM cell diagramm

**Gate signals**

$$\begin{aligned}
\underline{i}_l(n) &= \sigma(\mathbf{W}_{l,ix}\underline{x}_{l-1}(n) + \mathbf{W}_{l,is}\underline{s}_l(n-1) + \mathbf{W}_{l,io}\underline{x}_l(n-1) + \underline{b}_{l,i}),\\
\underline{f}_l(n) &= \sigma(\mathbf{W}_{l,fx}\underline{x}_{l-1}(n) + \mathbf{W}_{l,fs}\underline{s}_l(n-1) + \mathbf{W}_{l,fo}\underline{x}_l(n-1) + \underline{b}_{l,f}),\\
\underline{o}_l(n) &= \sigma(\mathbf{W}_{l,ox}\underline{x}_{l-1}(n) + \mathbf{W}_{l,os}\underline{s}_l(n-1) + \mathbf{W}_{l,oo}\underline{x}_l(n-1) + \underline{b}_{l,o}).
\end{aligned}$$

They all have the range $(0, 1)$. Typically, $\mathbf{W}_{l,*s}$ and $\mathbf{W}_{l,*o}$ are diagonal, i.e. each gate signal is affected by the memory state and output of the same LSTM cell.

**Update memory state**

$$\underline{s}_l(n) = \underline{f}_l(n) \odot \underline{s}_l(n-1) + \underline{i}_l(n) \odot \phi_l(\mathbf{W}_{l,sx}\underline{x}_{l-1}(n) + \mathbf{W}_{l,so}\underline{x}_l(n-1) + \underline{b}_{l,s})$$

new  normal recurrent layer

**Layer output**

$$\underline{x}_l(n) = \underline{o}_l(n) \odot \phi_l(\underline{s}_l(n))$$

Variants of feedback:

- only feedback from memory state: $\mathbf{W}_{l,*o} = \mathbf{0}$
- only feedback from output: $\mathbf{W}_{l,*s} = \mathbf{0}$

Abbildung 8.6: Lstm mathematical description

**Effects of gating:**
0 < gate signal < 1 input/output gate

| | input/output gate | forget gate |
|---|---|---|
| $\approx 0$ | close gate | clear memory |
| $\approx 1$ | open gate | keep memory |

Bidirectional LSTM-RNN also possible



# Chapter 9: Unsupervised and generative models

Date: 15/06/2020

*Lecturer:Bin Yang* *By: Nicolas Hornek*

## 9.1. Autoencoder(AE)

a DNN to learn an efficient representation/coding of input data

$$\underline{x} \rightarrow \begin{bmatrix} encoder, f_E, \\ \underline{\Theta}_E \end{bmatrix} \rightarrow \underline{z} \rightarrow \begin{bmatrix} decoder, f_D, \\ \underline{\Theta}_D \end{bmatrix} \rightarrow \hat{\underline{x}} \rightarrow loss \underset{\underline{y}=\underline{x}}{\uparrow}$$

$\underline{x} \in \mathbb{R}^d$ input

$\underline{z} = f_E(\underline{x}; \underline{\Theta}_E) \in \mathbb{R}^c$ : latent variable / representation $\equiv$ a hidden code for $\underline{x}$

$\hat{\underline{x}} = f_D(\underline{z}; \underline{\Theta}_D) = f_D(F_E(\underline{x}; \underline{\Theta}_E); \underline{\Theta}_D) = f(\underline{x}; \Theta) \in \mathbb{R}^d$ reconstruction for $\underline{x}$

$\underline{\Theta}_E, \underline{\Theta}_D, = \underline{\Theta} = \begin{bmatrix} \underline{\Theta}_E \\ \underline{\Theta}_D \end{bmatrix}$ : parameters

$\underline{y} = \underline{x}$ : self-copy, unsupervised !

**Encoder /decoder:**

•any NN

•typically symmetric

**Mostly undercomplete autoencoder:**: $c << d$

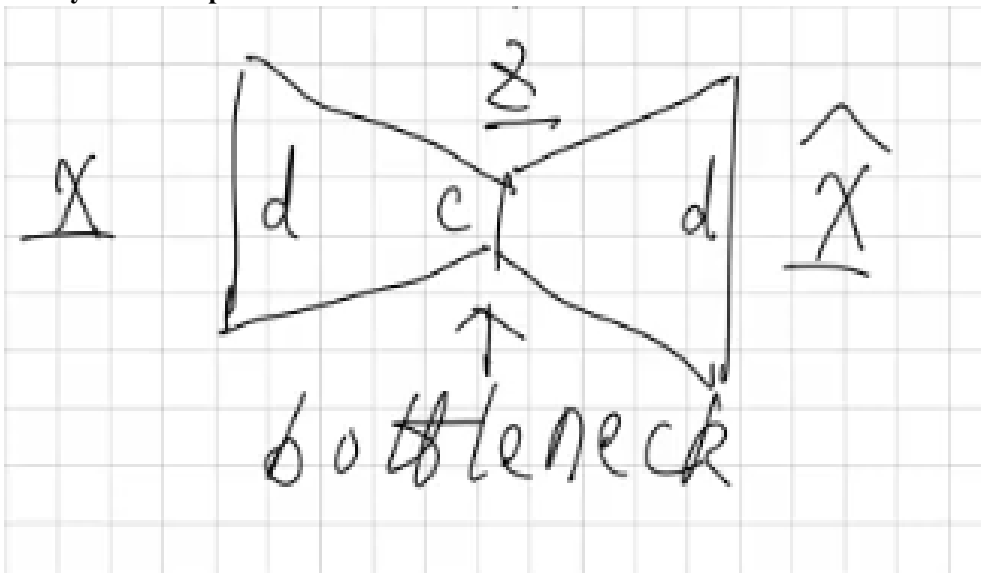



Abbildung 9.1: Undercomplete Autoencoder

overcomplete autoencoder (c » d) never used
Applications of autoencoder:
•a) Dimension reduction like PCA: c « d
•b) Denoising autoencoder
$\underline{x} + generatednoise/corruptions \rightarrow Autoencoder \rightarrow \underline{\hat{x}} \rightarrow \underline{y} = \underline{x} : \rightarrow$ clean input
reduce noise / corruption in input
$\underline{z}$: low dimensional ( c<d)
$\rightarrow$ keep only relevant information for $\underline{x}$ , drop noise/corruption information
c) reconstruction-based outlier/anomaly detection
•AE trained on normal data $\rightarrow \underline{\hat{x}} \approx \underline{x}$ for normal data
•if $\underline{x}$ outlier: $||\underline{\hat{x}} - \underline{x}||$ large $\rightarrow$ outlier can be detected
d) unsupervised preprocessing for other tasks
•automatic feature learning/extraction $\underline{z}$ instead of manual feature extraction in conventional machine learning

## 9.2. Variational Autoencoder

Theory of variational autoencoder (VAE): $p(\underline{x})$ : unknown distribution of $\underline{\underline{X}}$,
Onlz samples $\underline{x}(1), .., \xi(N)$ available
$q(\xi; \underline{\theta}$: parametric model for $p(\underline{x})$
$\underline{\theta}$ : parameters of VAE
Goal: $\min_{\underline{\theta}} D_{KL}(p||q)$
As in ch.3.4: cost function $L(\underline{\theta}) = D_{KL}(||q) = E_{\underline{x}\sim p} ln(\frac{p(\underline{\underline{X}})}{q(\underline{x};\underline{\theta})}) = \underbrace{E_{\underline{x}\sim p} ln(p(\underline{\underline{X}}))}_{\text{independent of } \underline{\theta}} - E_{\underline{x}\sim p(\text{unknown})}\, ln(q(\underline{x};\underline{\theta}))$
Replace $p(\underline{x})$ bz $\hat{p}(\underline{x}) = \frac{1}{N}\sum_{n=1}^{N} \delta(\underline{c} - \underline{x}_n)$
$\rightarrow L(\underline{\theta}) \approx const - E_{\underline{x}\sim p} ln(q(\underline{x};\underline{\theta})) = const. = \frac{1}{N}\sum_{n=1}^{N} - \underbrace{ln(q(\underline{x}(n);\underline{\theta}))}_{\text{loss } l(\underline{x}_n(n);\underline{\theta}) \text{ of VAE}}$
Difference to supervised learning in ch. 3.4 and ch. 4
•NO $\underline{y}$
•$p(\underline{x}, \underline{y}) \rightarrow p(\underline{x})$
$q(\underline{y}|\underline{x};\underline{\theta}) \rightarrow q(\underline{x}_n;\underline{\theta})$
Ho to model $q(\underline{x}_n;\underline{\theta})$?
VAE: $\underline{x}$ generated bz $\underline{z}$ with a known distribution
$\underline{z} \sim q(\underline{z})$, e.g. $\underline{z} \sim N(\underline{0}, \underline{I})$
$\underline{\mu}$

Important stuff to dimensions
Dear Student,

for understanding how batch operations work you have to think about how the data is typically stored in tensors for processing. Contrary to the notation that is often used in the literature, the input examples x are typically stored as row vectors in a matrix X. This means that if you have 10 input examples each being a vector of length 784, they will be stored in a matrix X of size 10x784.

If you want to implement a dense layer with 100 neurons, you then have to apply an affine mapping followed by the activation function a(). Fot the implementation this means that you input the matrix X with size 10x784 into the layer and as an output you want a matrix Y with size 10x100. The output of the layer is then calculated as Y=a(XW+B). In order for this operation to be defined the shapes of the matrices have to match. Here is the same operation with the shapes of the operands in brackets:
Y[10x100]=a(X[10x784]W[784x100]+B[10x100]) As you can see the shapes match and the overall function is defined. This might be confusing to you, since we are using row instead of column vectors but this allows for the batch dimension to be the first dimension and as far as I know all major DL frameworks like Tensorflow and Pytorch implement layers that expect the first dimension to be the batch dimension.

Hint: In Numpy and Tensorflow broadcasting is available
(https://numpy.org/doc/stable/user/basics.broadcasting.html?highlight=broadcasting),
so if you add the bias you dont have to use a matrix B of size 10x100. You can just add a vector of size 100.

So to summarize: You just have to work with row vectors and not with column vectors. I hope this is helpful for you.

With best regards, Felix Wiewel