Deep Learning Building Blocks

Dr. Gerhard Paaß
Fraunhofer Institute for Intelligent Analysis and Information Systems (IAIS)
Sankt Augustin

TensorFlow

Course Overview

1. Intro to Deep Learning	Recent successes, Machine Learning, Deep Learning & types
2. Intro to Tensorflow	Basics of Tensorflow, logistic regression
3. Building Blocks of Deep Learning	Steps in Deep Learning, basic components
4. Unsupervised Learning	Embeddings for meaning representation, Word2Vec, BERT
5. Image Recognition	Analyze Images: CNN, Vision Transformer
6. Generating Text Sequences	Text Sequences: Predict new words, RNN, GPT
7. Sequence-to-Sequence and Dialog Models	Transformer Translator and Dialog models
8. Reinforcement Learning for Control	Games and Robots: Multistep control
9. Generative Models	Generate new images: GAN and Large Language Models

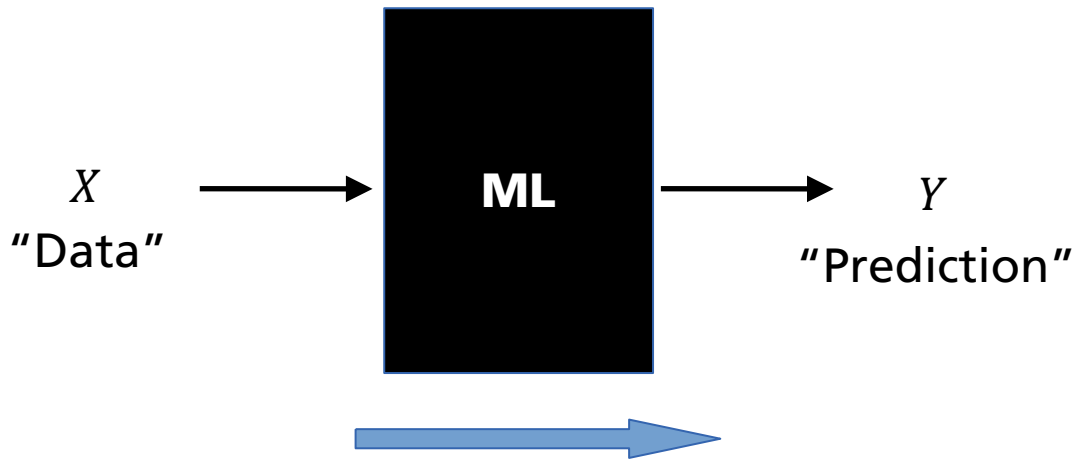
: link to background material, : link to images used in lecture, G. : Terms that may be asked in the exam

Deep Learning Building Blocks
Agenda

1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

Deep Learning as Black Box Approach

- **Deep Learning** (DL) is **Machine Learning**: Training of Deep Neural Networks
- DL algorithms are a subset of general machine learning methods.
- General abstract scheme of ML algorithms:



- ML algorithms "learns" mapping from input to output by example data
- Deep network: composed of many layers (operators)
  - Shallow Learning: non-deep ML algorithms

G. Deep Learning G. Shallow learning

Mapping Examples

Lioness at the Louisville Zoo by Ltshears / CC BY-SA 3.0

X

numeric representation

Y

numeric output

"lion"

- Most ML algorithms operate on numeric data
- ML algorithms "learns" a **function** $f(X) = Y$

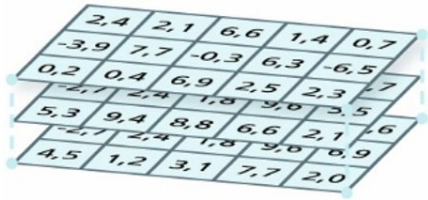
vector

0,1	-2,3	1,1	0,0	-1,7

matrix

1,9	4,3	2,9	-0,3	4,8
7,2	4,7	0,3	4,5	7,3
6,3	-1,6	0,9	0,5	2,9

tensor



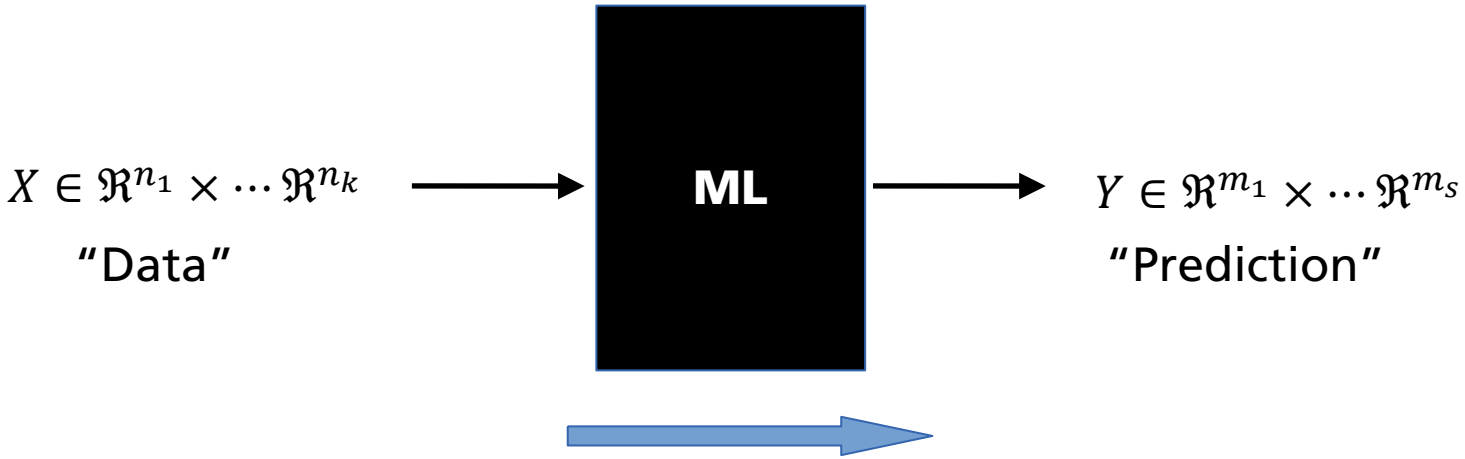
G. vector G. matrix G. tensor

Deep Learning Building Blocks
Agenda

1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Main Groups of Machine Learning Models
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output: decision tree, SVM, …
- More complex problems need post-processing

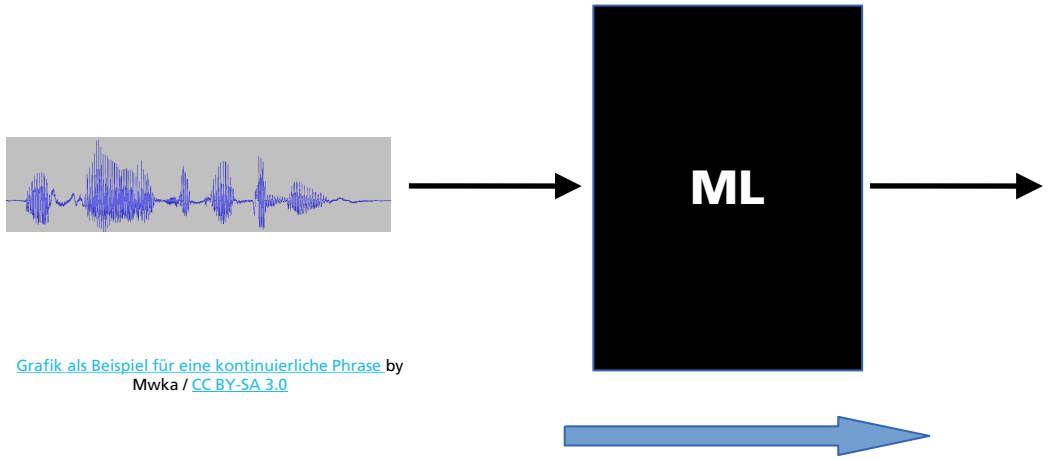


- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

G. mapping between multidimensional spaces

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing

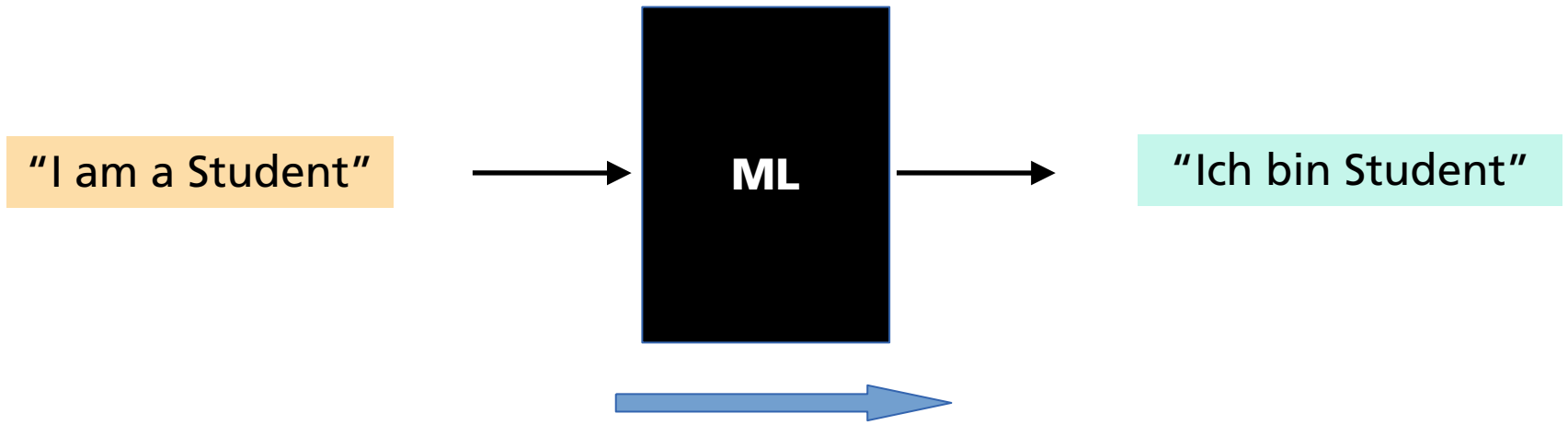


Grafik als Beispiel für eine kontinuierliche Phrase by Mwka / CC BY-SA 3.0

- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing



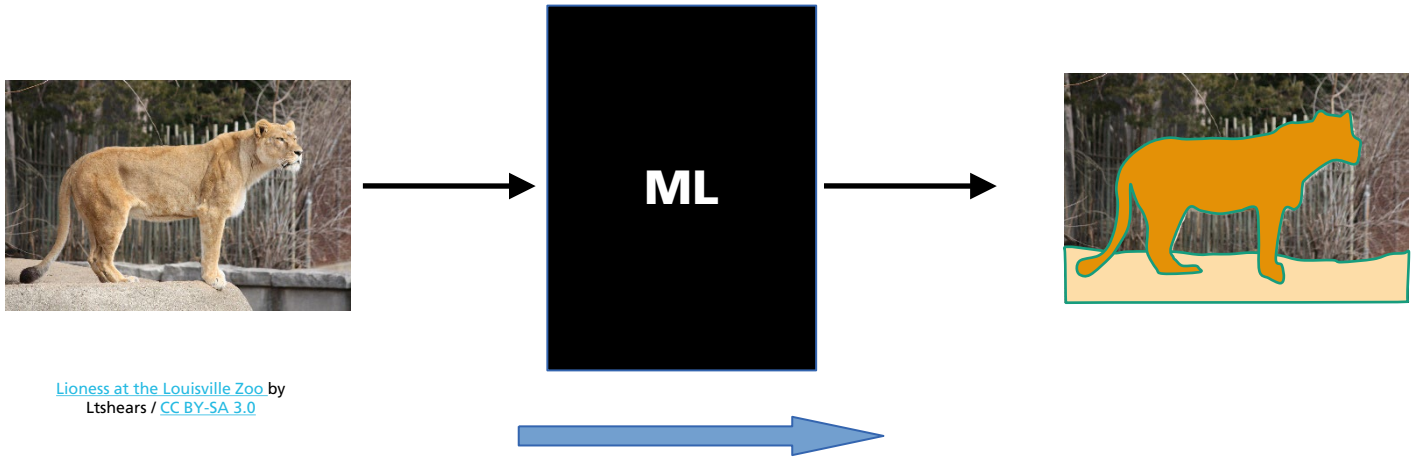
Example:

translation

- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing



Lioness at the Louisville Zoo by Ltshears / CC BY-SA 3.0

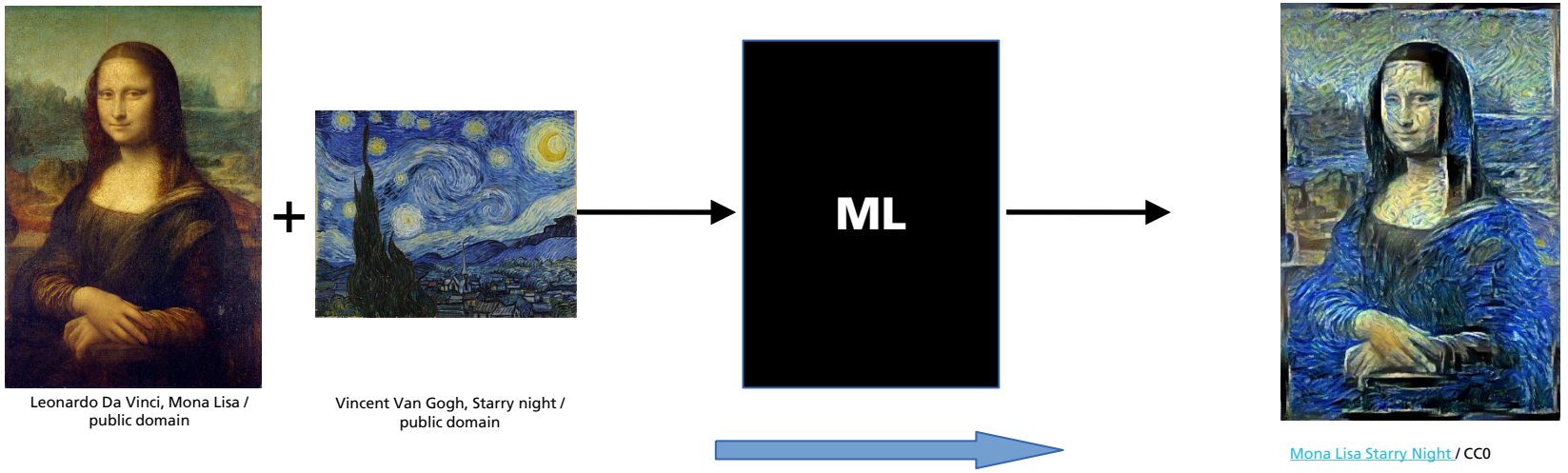
Example:

semantic segmentation

- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing



Leonardo Da Vinci, Mona Lisa / public domain

Vincent Van Gogh, Starry night / public domain

Mona Lisa Starry Night / CC0

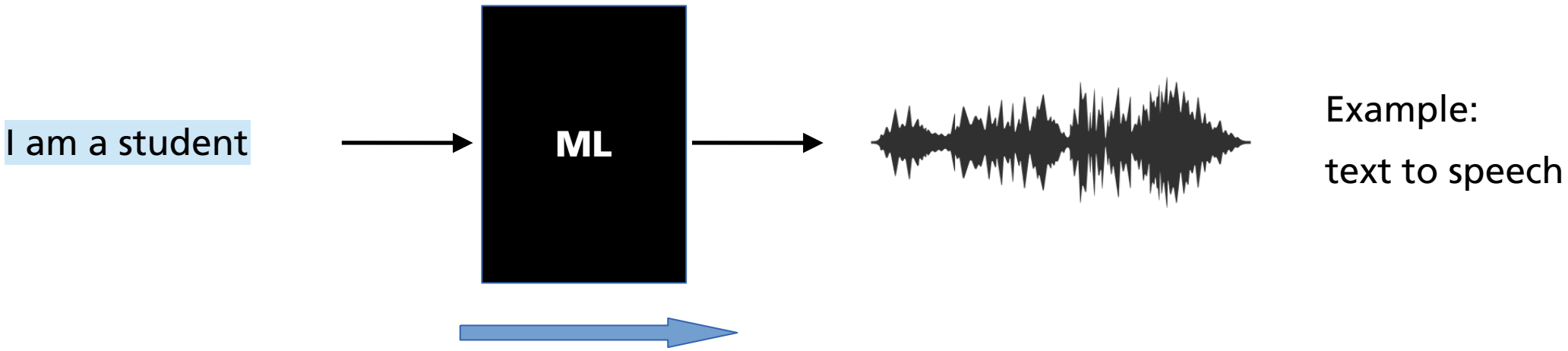
Example:

style transfer

- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing

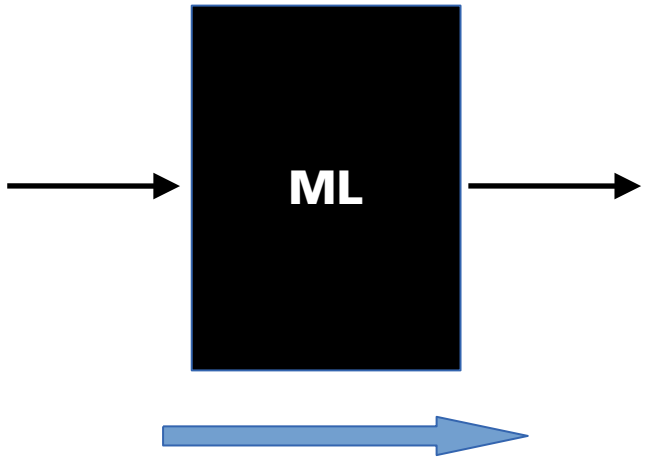


- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output:
  decision tree, SVM, …
- More complex problems need post-processing

Public domain



Public domain

Example:

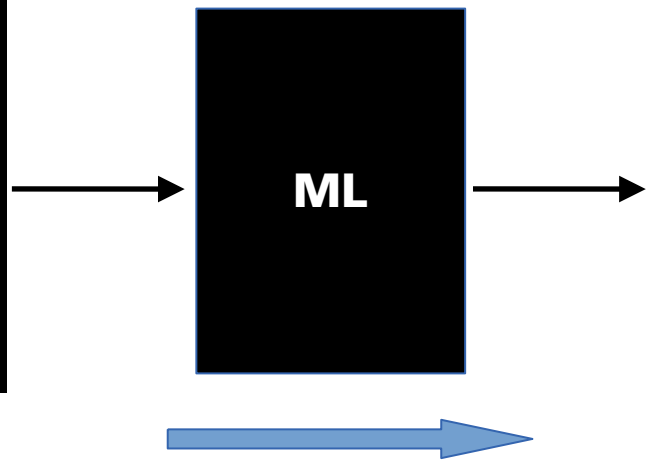
data generation

- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning can Produce High-dimensional Outputs

- Many ML models only generate a single output: decision tree, SVM, …
- More complex problems need post-processing

Screenshot from simulator



"move left"

Example:

game playing

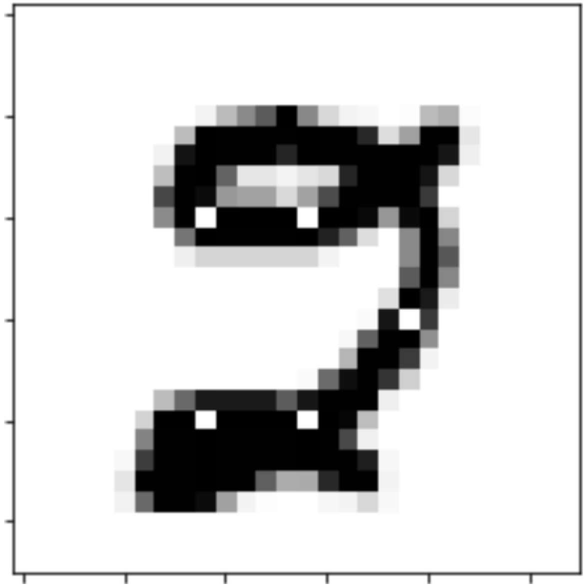
- Deep Learning generates high-dimensional outputs: tensors
  - may represent different type of media

Deep Learning Building Blocks
Agenda

1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Main Groups of Machine Learning Models
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

Data of an object is modified by Transformations

- Image data is encoded by a pixel matrix



Printed from MNIST

.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	0.1	0.4	0.5	0.7	1.0	0.5	0.3	0.1	0.1	.	0.0	0.4	0.4	0.0	.	.	.	.	.
.	.	.	.	.	.	.	.	0.4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.2	0.5	1.0	1.0	0.2	.	.	.	.	.
.	.	.	.	.	.	.	0.1	0.9	1.0	1.0	1.0	1.0	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.1	.	.	.	.	.
.	.	.	.	.	.	.	0.4	1.0	1.0	0.7	0.2	0.2	0.1	0.2	0.2	0.9	1.0	1.0	1.0	0.9	0.2	.	.	.	.	.	.
.	.	.	.	.	.	.	0.8	1.0	0.9	0.5	0.5	0.5	0.2	0.5	0.8	1.0	1.0	1.0	1.0	0.8	.	.	.	.	.	.	.
.	.	.	.	.	.	.	0.5	1.0	.	1.0	1.0	1.0	1.0	.	1.0	1.0	1.0	0.5	1.0	1.0	0.3	.	.	.	.	.	.
.	.	.	.	.	.	.	.	0.6	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.7	0.2	.	0.5	1.0	0.6	.	.	.	.	.	.
.	.	.	.	.	.	.	.	0.1	0.3	0.3	0.3	0.3	0.3	0.3	0.1	.	.	.	0.5	1.0	0.7	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.7	1.0	0.6	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.2	1.0	0.9	0.1	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.0	0.9	.	0.8	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.1	0.7	1.0	1.0	0.5	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.4	1.0	1.0	0.8	0.1	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	0.0	0.6	0.9	1.0	0.8	0.3	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	0.4	0.7	0.9	0.9	0.9	0.9	0.7	0.9	1.0	1.0	1.0	0.1	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	0.3	1.0	1.0	.	1.0	1.0	1.0	1.0	.	1.0	1.0	0.4	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	0.1	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	0.1	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.1	.	.	.	.	.	.	.	.	.
.	.	.	.	.	0.2	1.0	1.0	1.0	1.0	1.0	1.0	0.7	0.4	0.4	0.9	1.0	1.0	0.1	.	.	.	.	.	.	.	.	.
.	.	.	.	.	0.1	0.7	1.0	1.0	0.9	0.5	0.1	0.0	.	.	0.1	0.1	0.3	0.0	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

- If image is shifted to the right ➔ corresponding numbers appear in different places of matrix
- need image features which are invariant to this transformation

There are many types of image transformations

need representation invariant to these transformations

"Cats in the Spring" by zaimoku_woodpile / CC BY 2.0

illumination

"Cat" by Sean MacEntee / CC BY 2.0

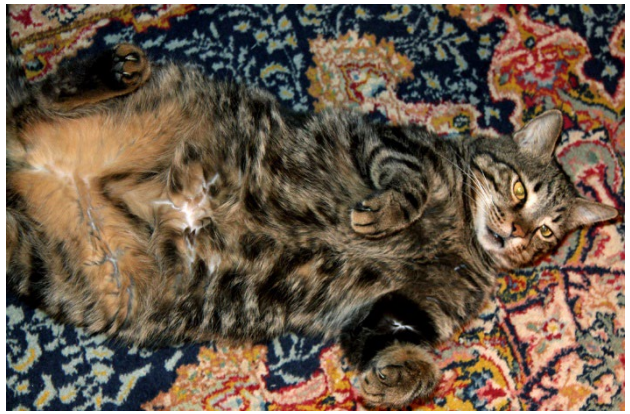
deformation

"Big Fat Cat" by In Memoriam: -Tripp- / CC BY 2.0

occlusion

"Funny cat posing" by be creator / CC BY 2.0

background clutter



"CAT" by Bahman Farzad / CC BY 2.0

variation within the class

"So many kittens!" by Clevergrrl / CC BY-SA 2.0

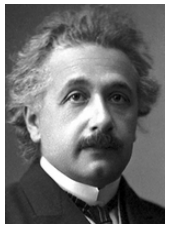
G. invariant mapping by Deep Neural Network

Finding invariant representation

Requirements

- Preserve information semantic classes have in common
- Ignore other information
- Is invariant against transformations and variations of objects in the semantic class
- reduces the dimensionality of the problem

- Traditional ML: manual feature definition



Public domain

X
original numerical representation

manual feature extraction

eye, mouth, nose ears, …

$\tilde{X}$
optimized feature representation

ML

Y
numeric output

"Einstein"

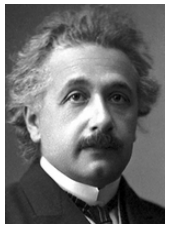
- Very difficult to find robust mathematical representations
- Human experts often are not successful

Finding invariant representation

Requirements

- Preserve information semantic classes have in common
- Ignore other information
- Is invariant against transformations and variations of objects in the semantic class
- reduces the dimensionality of the problem

- Deep learning with many layers: hidden vectors are **new optimized features**



Public domain

X
original numerical representation

automatic feature extraction

$\tilde{X}$
optimized feature representation

ML

Y
numeric output

"Einstein"

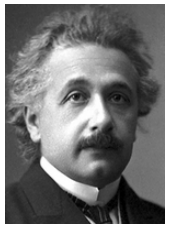
- Intermediate features are determined by optimization
- End-to-end learning

Finding invariant representation

Requirements

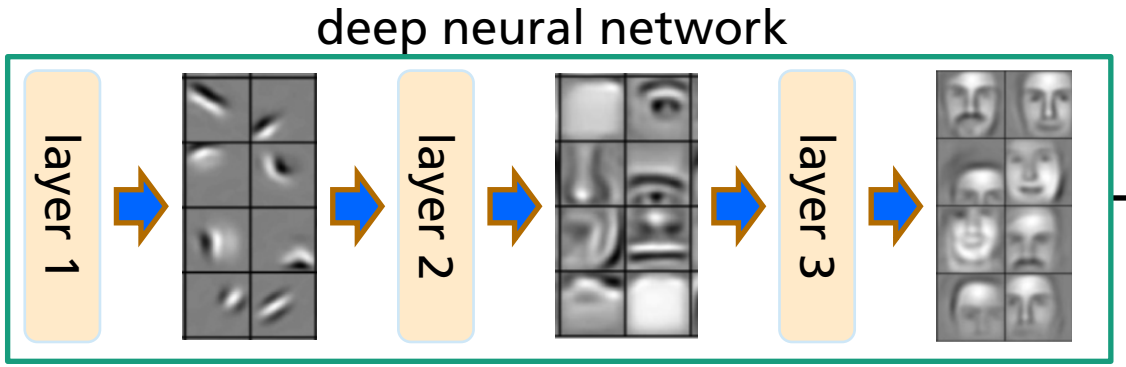
- Preserve information semantic classes have in common
- Ignore other information
- Is invariant against transformations and variations of objects in the semantic class
- reduces the dimensionality of the problem

- Deep learning with many layers: hidden vectors are new features



Public domain

X
original
numerical
representation



Y
numeric
output

"Einstein"

- Intermediate features are determined by optimization [Lee et al. 2011 p.101]
- End-to-end learning

03-a

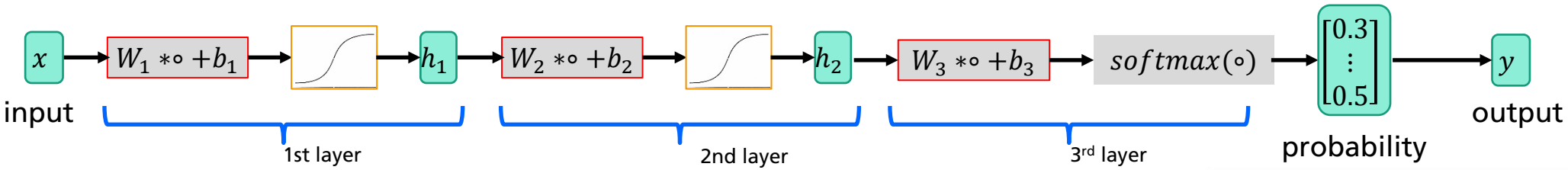
Deep Learning Building Blocks
Agenda

1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

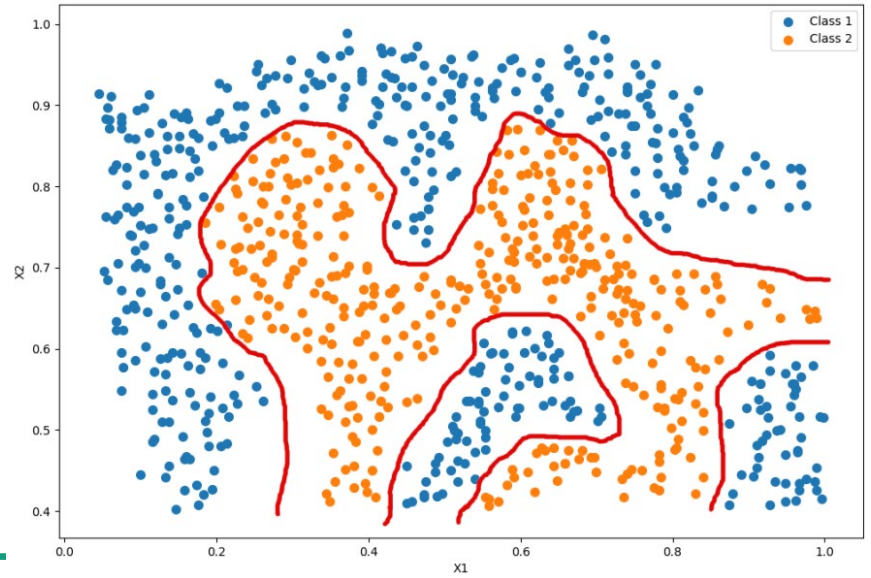
Main ingredients of deep neural networks

Basic structure: defined by the user

- layers, often with linear transformations $W * x + b$
- non-linear activation functions: required capture nonlinear decision boundaries
- Last layer generates desired prediction / classification



- Each layer usually has some parameters: $W_1, b_1, W_2, b_2, W_3, b_3, \ldots$
  - initialized randomly
  - determined by optimization procedure
    ➔ good prediction of training set outputs

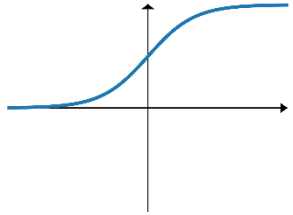


➔ **Fully connected layers, fully connected network**

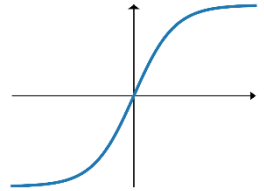
Fraunhofer BIG DATA AI

Activation functions

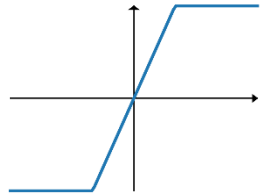
- sigmoid $1/(1+e^{-x})$



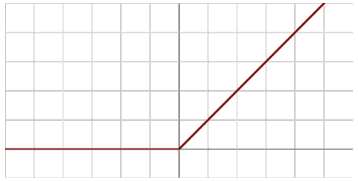
- hyperbolic tangent $(e^{x}-e^{-x})\,/\,(e^{x}+e^{-x})$



- hard tanh $\max(-1,\min(1,x))$

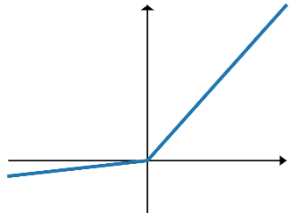


- rectified linear unit (ReLU) $\max(x,0)$

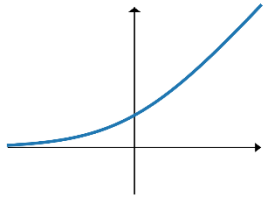


good default choice

- parametric ReLU $\max(x,0)+\alpha\min(0,x)$



- softplus $\log(1+e^{x})$



and many others

Optimization Criterion: Loss function

- training set instances $\text{Train} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - Change parameters $w = W, b$ such that most predicted outputs are correct
  - How to measure the accuracy / loss of the prediction?

IDEA:

- Neural network computes the probability of the output y given the input x: $\hat{p}(y|x; w)$
- compute the probability of the training set for the current parameter w

$$\hat{p}(TrainSet|w) = \hat{p}(y_1|x_1; w) * \cdots * \hat{p}(y_n|x_n; w)$$

Likelihood of w

"*" as observations in the training set are **independent**

- **Maximum likelihood principle**: modify w such that the likelihood $\hat{p}(TrainSet|w)$ gets maximal
  - take $\log(\cdot)$ and multiply by -1: better numerical properties, no underflow, same optimum
  - **loss function**: to be minimized

$$L(w) = -\log\big(\hat{p}(TrainSet|w)\big) = -[\log \hat{p}(y_1|x_1; w) + \cdots + \log \hat{p}(y_n|x_n; w)]$$

maximum likelihood
loss function

G. Probability of Training Set G. loss function G. likelihood G. maximum likelihood
G. maximum likelihood principle G. maximum likelihood loss function

Different Loss function may be used

- **continuous** output variable y
- Assumption: $\hat{p}(y|x;w) = N(f(x;w), \sigma^2)$
  - where $f(x;w)$ is the output of the DNN with parameter w
  - $N(\mu, \sigma^2)$ is a normal distribution with expectation μ and variance σ^2.

- **loss function**: to be minimized

$$L(w) = -\log\big(\hat{p}(TrainSet|w)\big) = \big(y_1 - f(x_1;w)\big)^2 + \cdots + \big(y_n - f(x_n;w)\big)^2$$ mean square error

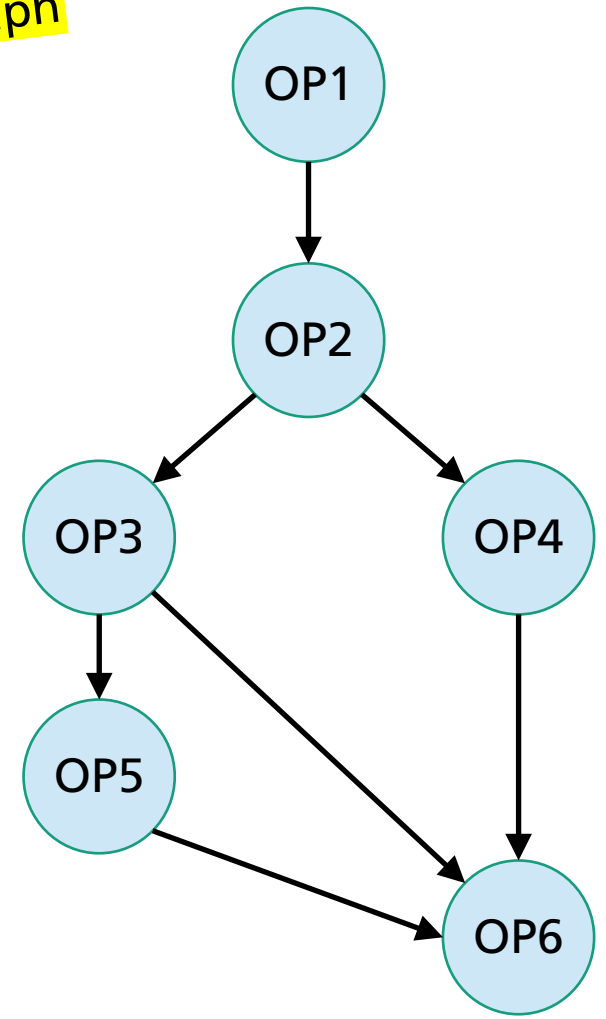
Other loss function

- training cases have different variances / measurement errors: weighted least squares
- 0-1 loss function: loss is 0 if $y = f(x;w)$ and 1 otherwise
- hinge loss: loss is $\max\big(0, 1 - y * f(x;w)\big)$, where $y \in \{-1, 1\}$
  used in SVM and maximum margin classification
- robust loss functions: less sensitive to outliers
- …

Neural Network structure may be a Graph

- A network is composed of operators (=layers)
- An operator takes one or more input tensors and computes one or more output tensors
- an operator may have parameters

directed acyclic graph



Input
- from files
- from databases
- …

Preprocessing
- downsampling
- normalization
- filtering
- …

Reshaping
- split
- merge
- flatten
- …

processing
- linear
- convolution
- pooling
- …

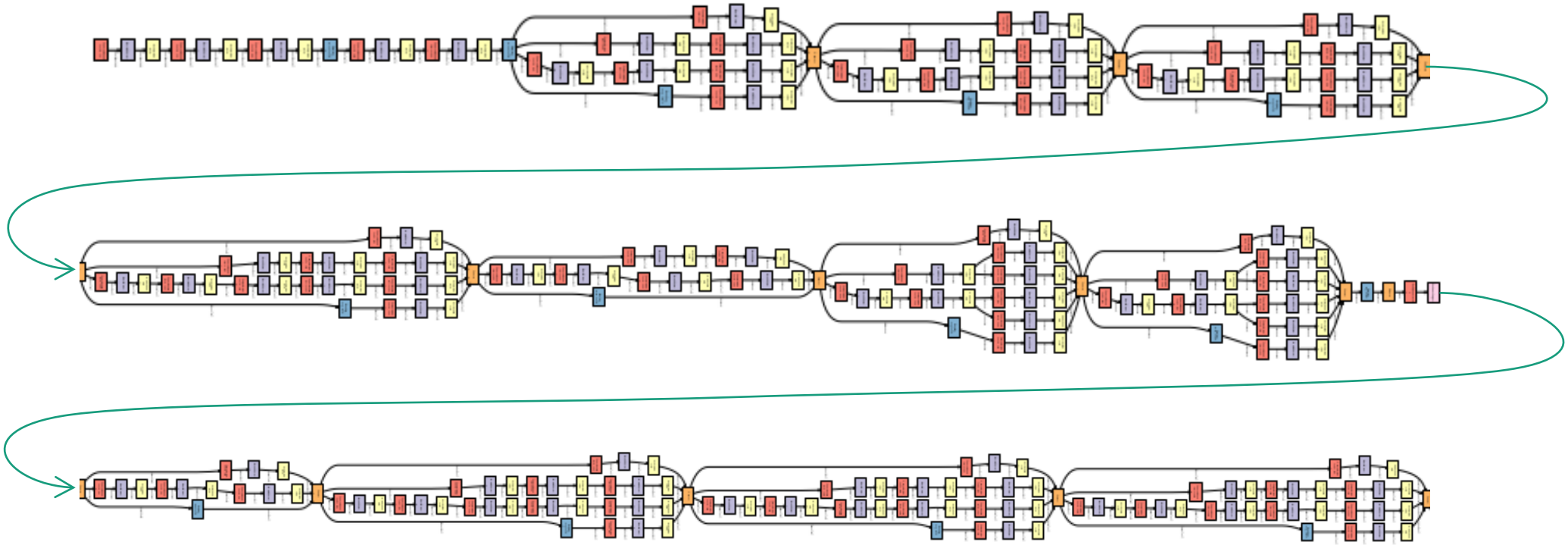
regularization
- dropout
- layer normalization
- …

loss function
- softmax
- square difference
- …

G. network operator G. directed graph

Network structure may be a graph

- A more complex example



Topology: InceptionNet_v3 for image classification

Deep Learning Building Blocks
Agenda

1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

Optimization by Gradient Descent

- want to go down into the valley
- very limited sight: you can only see close neighborhood
- similar to descent from the moutain in fog
- approach:
  until you reach the bottom:
  1. determine the direction of steepest descent
  2. go a step into that direction
  3. repeat step 1

alpine_fog_snow_new_zealand / public domain

Optimization by Gradient Descent

- observed: training data (x, y) model: $\hat{y} = f(x; w)$ loss: $L(y, \hat{y})$
- $L(w) = -\log[p(TrainSet|w)] = -[\log \hat{p}(y_1|x_1; w) + \cdots + \log \hat{p}(y_n|x_n; w)]$
- determine gradient

$$\frac{\partial L(w)}{\partial w} = \left(\frac{\partial L(w)}{\partial w_1}, \ldots, \frac{\partial L(w)}{\partial w_m}\right)$$

univariate derivative of $L(w)$ with respect w_m

$$\nabla_w L(w) := \frac{\partial L(w)}{\partial w}$$

alternative notation

- $\frac{\partial L(w)}{\partial w}$ is **direction of steepest ascent**
- gradient computation: some rules
  - sum rule $\frac{\partial (f(w)+g(w))}{\partial w} = \frac{\partial f(w)}{\partial w} + \frac{\partial g(w)}{\partial w}$
  - product rule $\frac{\partial (f(w)*g(w))}{\partial w} = \frac{\partial f(w)}{\partial w} * g(w) + \frac{\partial g(w)}{\partial w} * f(w)$
  - **chain rule** $\frac{\partial f(g(w))}{\partial w} = \frac{\partial f(u)}{\partial u} * \frac{\partial g(w)}{\partial w}$ where $u = g(w)$

multivariate chain rule

$$\left[\frac{\partial (f(g(w)))}{\partial w_j}\right] = \left[\frac{\partial f(u)}{\partial u_i}\right]_i * \left[\frac{\partial g_i(w)}{\partial w_j}\right]_{i,j}$$

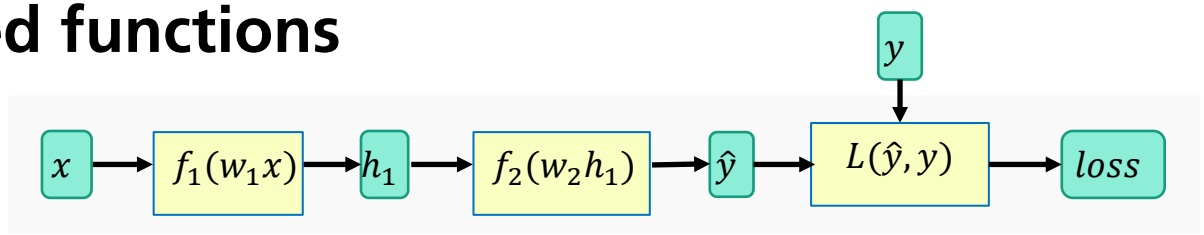
$g(w)$ maps a vector w to a vector u

$f(u)$ maps a vector u to a scalar

vector of derivatives

matrix of derivatives

How to compute gradients: nested functions



- very simple **univariate** model is a nested function
- Model $u_1 = w_1x;\ h_1 = f_1(u_1);\ u_2 = w_2h_1;\ \hat{y} = f_2(u_2);\ loss = L(\hat{y},y)$

$$loss = L(f_2(w_2f_1(w_1x)),y)$$

- derivatives by the chain rule

$$\frac{\partial L(f_2(w_2f_1(w_1x))),y)}{\partial w_1} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}}(\hat{y}) * \frac{\partial f_2(u_2)}{\partial u_2}(u_2) * \frac{\partial (w_2h_1)}{\partial h_1}(h_1) * \frac{\partial f_1(u_1)}{\partial u_1}(u_1) * \frac{\partial (w_1x)}{\partial w_1}(x)$$

$= w_2$ $= x$

$$\frac{\partial L(f_2(w_2f_1(w_1x))),y)}{\partial w_2} = \frac{\partial L(\hat{y},y)}{\partial \hat{y}}(\hat{y}) * \frac{\partial f_2(u_2)}{\partial u_2}(u_2) * \frac{\partial (w_2h_1)}{\partial w_2}(h_1)$$

$= h_1$

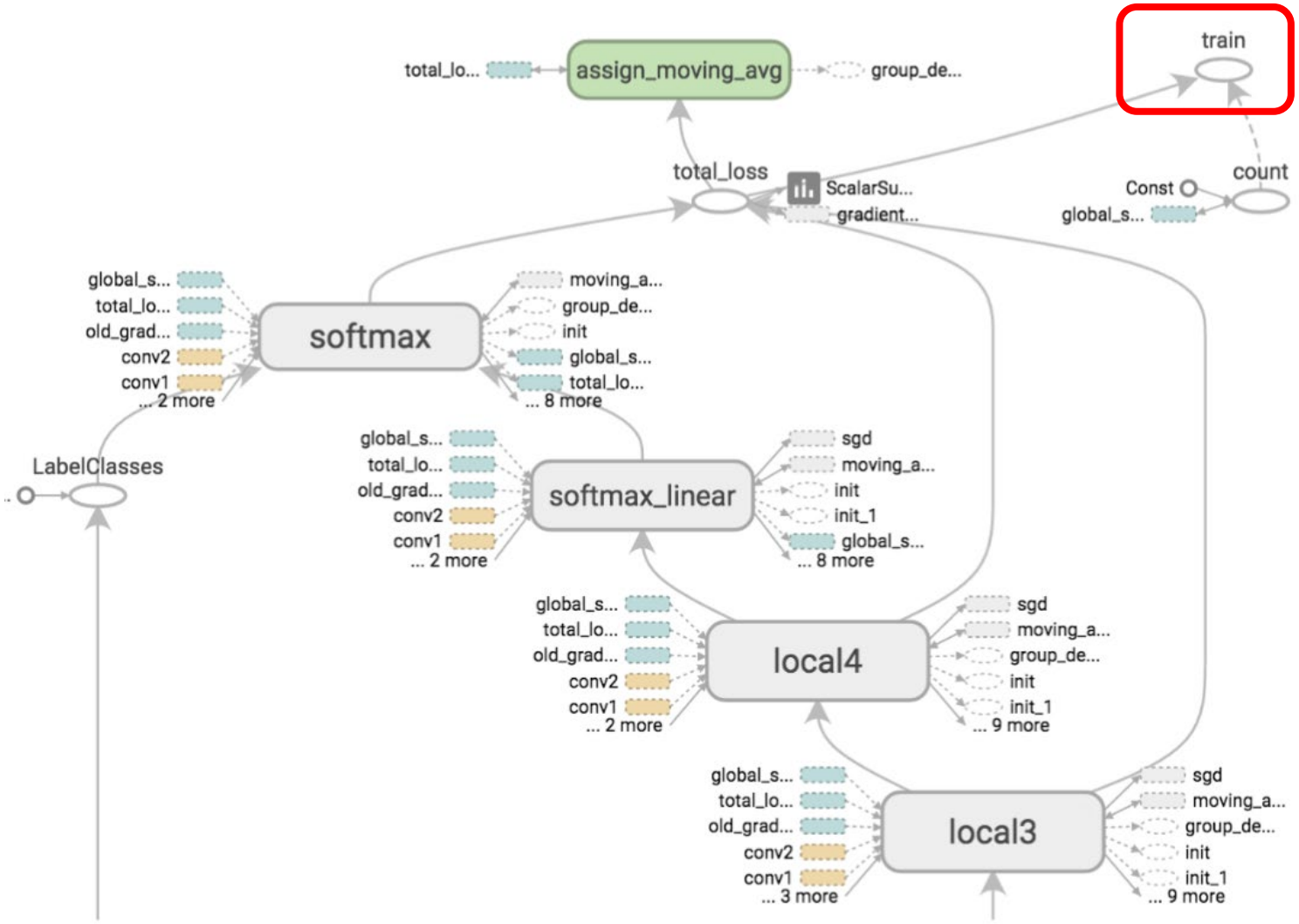
- you always need the prediction of quantities: $u_1, h_1, u_2, \hat{y}$ ➔ forward propagation
- compute derivatives starting from the outer (last) function ➔ backward propagation
  - use common terms: e.g. $\frac{\partial L(\hat{y},y)}{\partial \hat{y}}(\hat{y}) * \frac{\partial f_2(u_2)}{\partial u_2}(u_2) = \frac{\partial L(\hat{y},y)}{\partial u_2}(u_2)$ ➔ combine upstream derivatives
- works similar for functions with vector input

backpropagation

Automatic Gradient Computation

- gradient computation is tedious & error prone
- all mayor frameworks for neural networks have **automatic gradient** computation
  - specify the forward computations
  - define inputs and outputs
  - define parameters
- this defines the flow graph
  ➔ automatically compute gradients with respect to all parameters

TensorBoard graph visualization of a convolutional neural network model



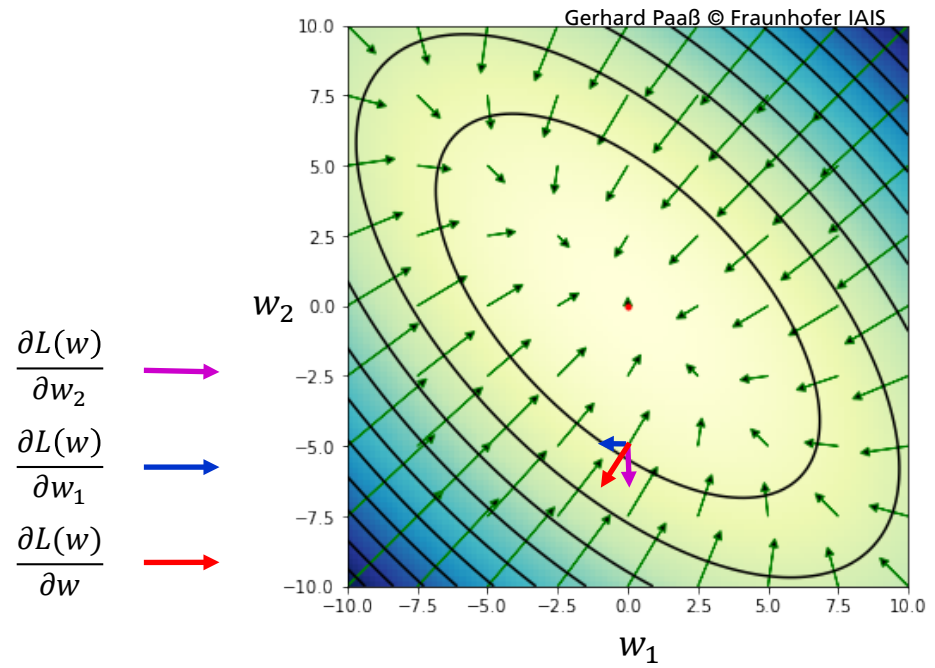
Tensorboard screenshot

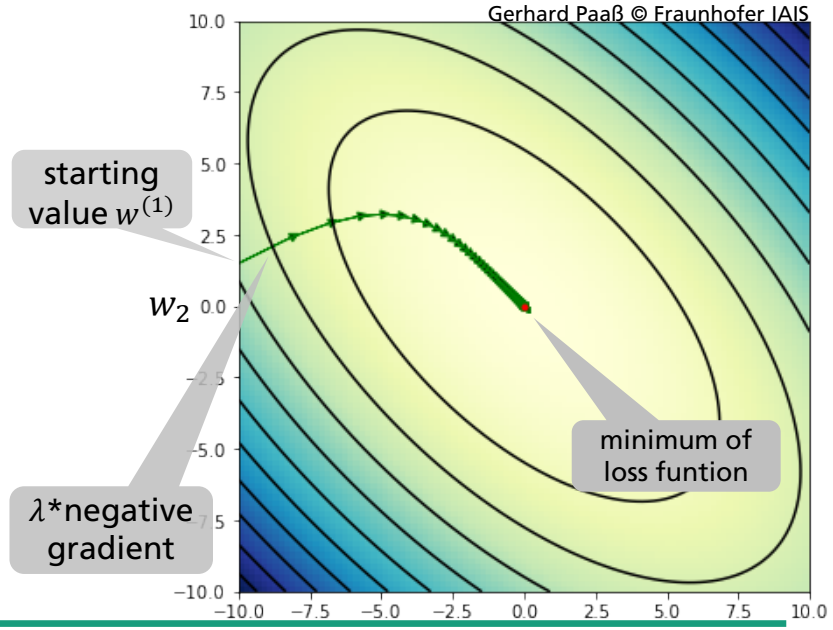
Gradient Descent Algorithm

- negative gradients $-\frac{\partial L(w)}{\partial w} = -\left(\frac{\partial L(w)}{\partial w_1}, \ldots, \frac{\partial L(w)}{\partial w_m}\right)$
  directions for steepest descent

$$L(w) = -\log[p(TrainSet|w)]$$
$$= -[\log \hat{p}(y_1|x_1; w) + \cdots + -\log \hat{p}(y_n|x_n; w)]$$

- **gradient descent algorithm**
  1. initialize parameter vector $w^{(1)}$ with random numbers
  2. compute gradient $\frac{\partial L(f(x,w),\hat{y})}{\partial w}$ of loss function in point $w^{(t)}$
  3. set $w^{(t+1)} = w^{(t)} - \lambda * \frac{\partial L(f(x,w),\hat{y})}{\partial w}$
  4. if change very small then stop, else continue with step 1
- step size $\lambda > 0$ is called **learning rate**
- gradient computed for all training set elements: **epoch**
- in general: non-convex optimization, difficult to solve





G. gradient descent optimization G. Learning rate G. Epoch

Stochastic Gradient Descent

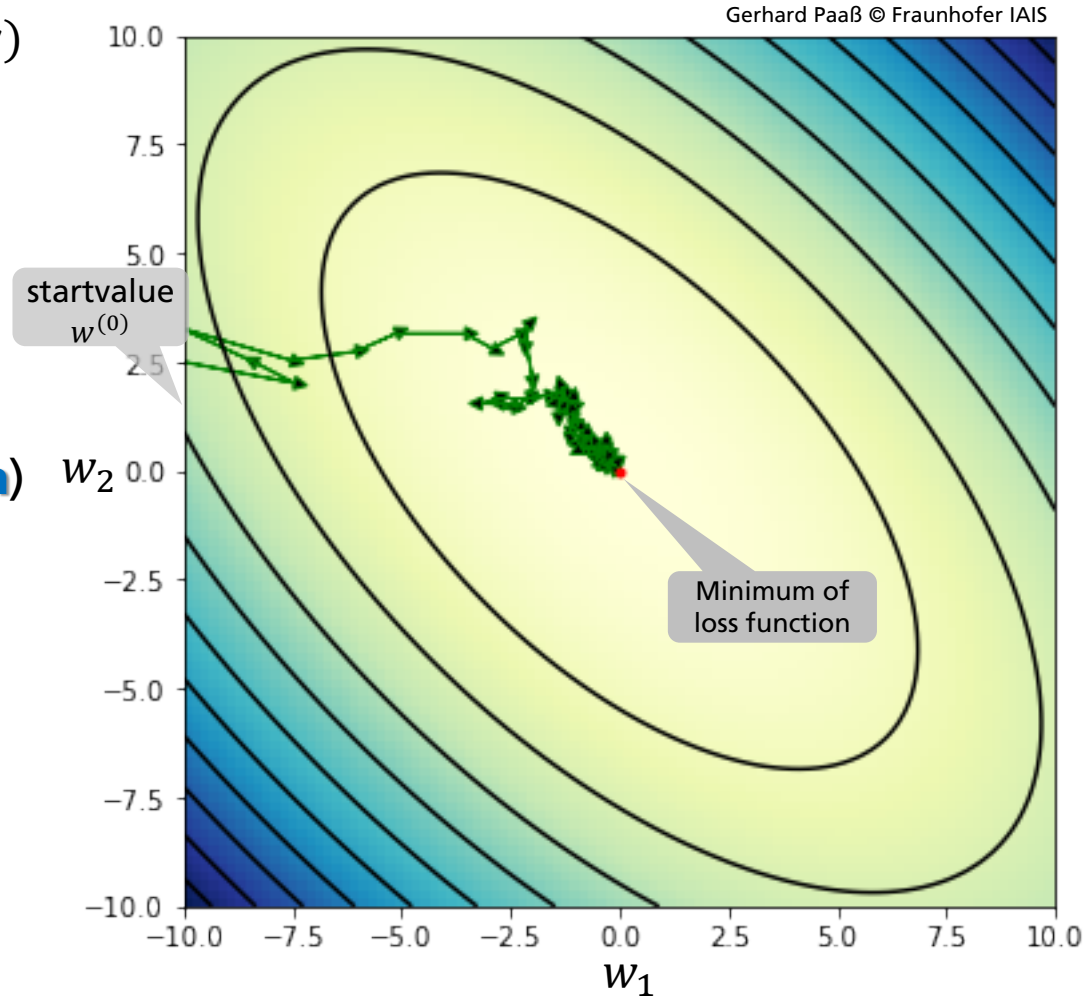
- loss function $L(w) = -\log \hat{p}(y_1|x_1; w) - \cdots - \log \hat{p}(y_n|x_n; w)$
- gradients $\frac{\partial L(w)}{\partial w} = -\frac{\partial \log \hat{p}(y_1|x_1; w)}{\partial w} - \cdots - \frac{\partial \log \hat{p}(y_n|x_n; w)}{\partial w}$
- high computational effort if training set is large

alternative:

- compute gradient for small random samples (**minibatch**)
  batchsize: number of instances in minibatch
- perform gradient descent
- gradient different from gradient for full training data
- but on average gradient is correct

properties:

- much less computational effort
- can escape local minima because of random variation



G. minibatch G. batchsize G. stochastic gradient descent

Stochastic Gradient Descent Variants

- stochastic gradient descent with momentum
- basic idea:
  - use previous gradients to speed up
  - moving average of gradients

$$V^{(t+1)} = \eta V^{(t)} + (1-\eta)\frac{\partial L(w^{(t)})}{\partial w}$$

  - gradient descent

$$w^{(t+1)} = w^{(t)} - \lambda V^{(t+1)}$$

Alec Radford's animations for optimization algorithms

- many more methods: adapt learning rates to function
  Adagrad, AdaDelta, Adam, RMSprop
- no method uniformly superior
  - Adam is a good default choice in many cases
  - SGD+Momentum can outperform Adam,
    but needs learning rate tuning

Comparison of different optimization algorithms

Core technology of neural nets

? 03-b

Deep Learning Building Blocks
Agenda

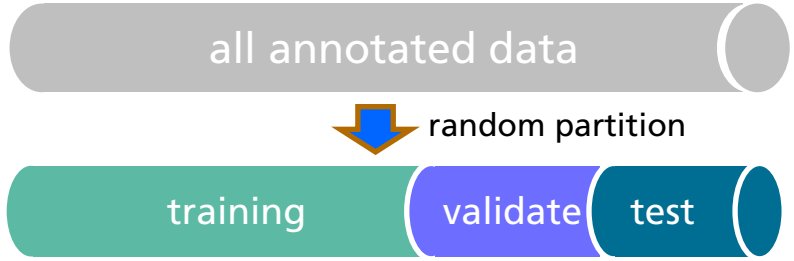
1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

Testing Performance of Neural Network

- Train parameters using the training set
- test parameters using a separate test set

- you may **not improve model** using results from the test set
  ➔ otherwise test set performance too optimistic

- introduce a new **validation set**

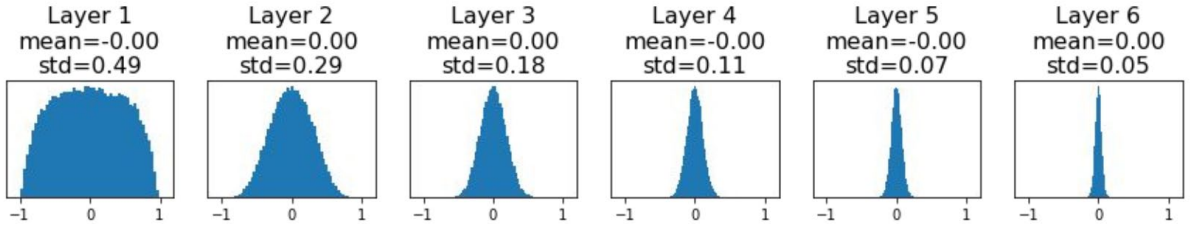


- validation set may be used for model improvement
  - selecting model architecture
  - select hyperparameters, e.g. learning rate
- use test set only for final estimation of model accuracy
  - unbiased estimate of performance on new data with same data distribution

Weight initialization

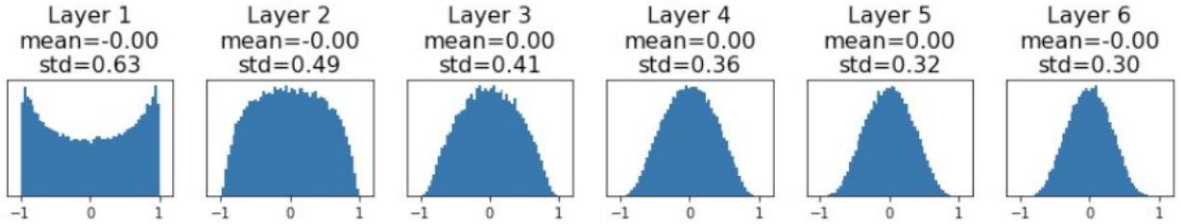
- weights have to be initialized randomly
  - ➔ symmetry breaking, otherwise derivative = 0
  - exception: bias weights b in $Ax + b$

- initialize by small random numbers
  - distribution of activations tend to zero



distribution of derivatives

- Initialize such that activation values on average have prescribed variance
  - normalize by $\sqrt{number\ of\ inputs}$
  - distribution of derivaties ok

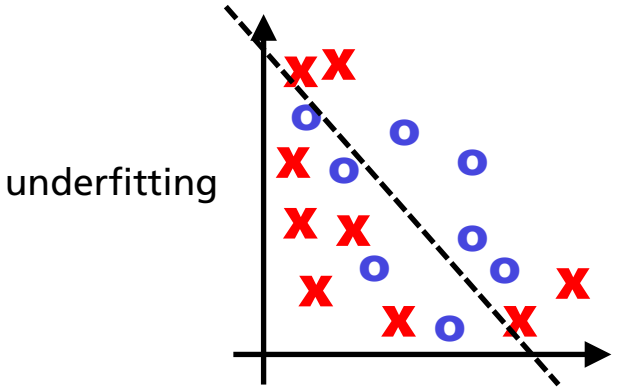


distribution of derivatives

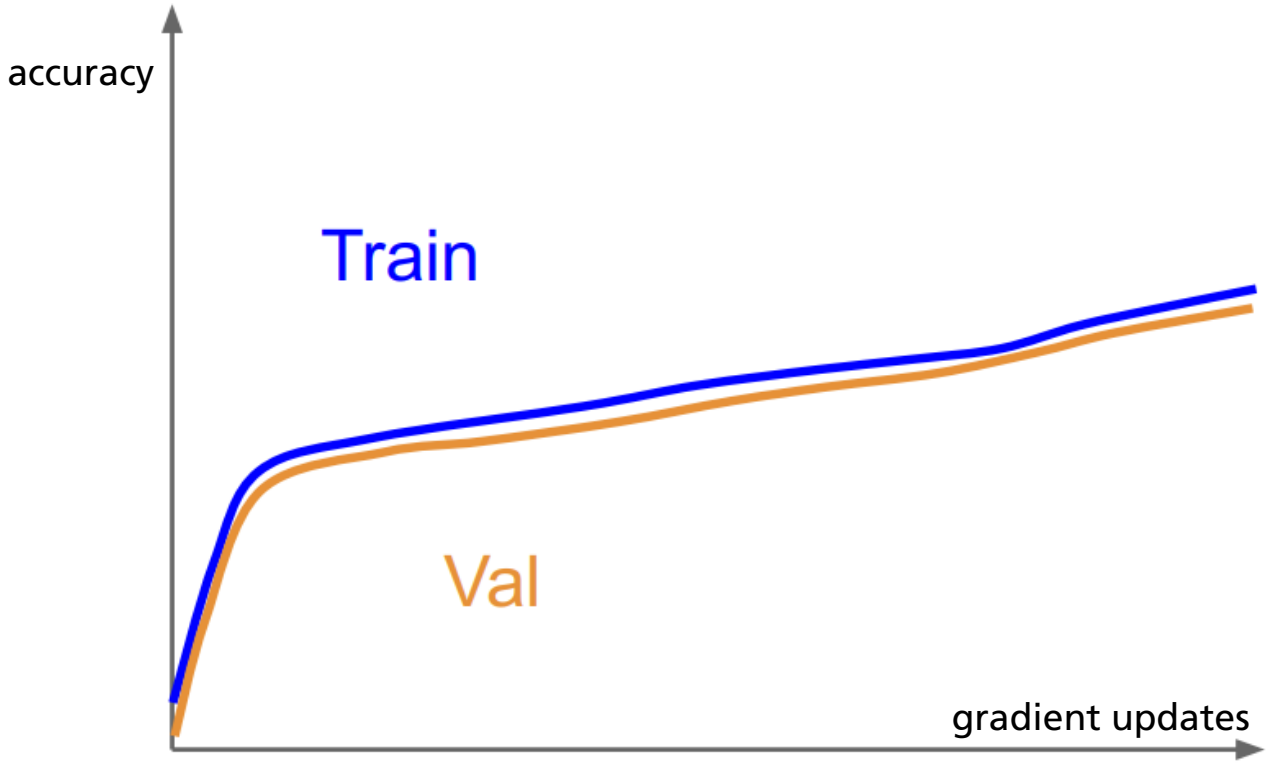
see also: https://www.deeplearning.ai/ai-notes/initialization/

Underfitting

- neural networks $y = f(x; w)$
- may not capture the structural relation between input x and output y
  - high error on training and test data: **underfitting**

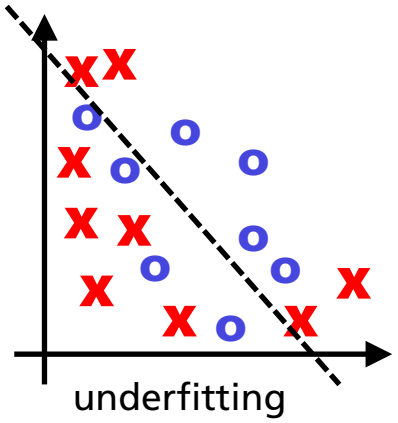


- improvements
  - more iterations
  - more hidden units (wider),
  - more layers (deeper),
  - more data
  - change hyperparameters:
    learning rate, minibatch size, …

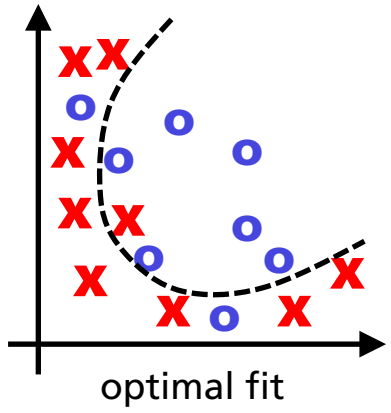


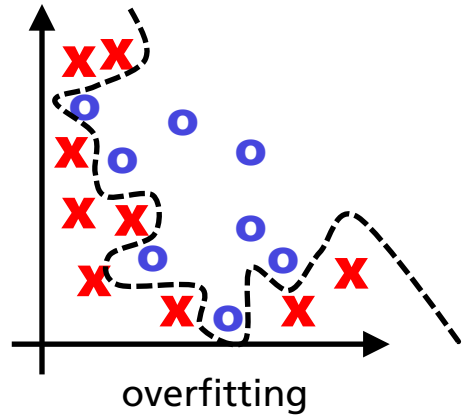
Overfitting

- large number of parameters
- represent non-systematic variations in training data
  ➔ **overfitting**
- large gap between error on training data and error on validation data
- improvements
  - fewer iterations: early stopping
  - reduce number of parameters
  - change hyperparameters: learning rate, minibatch size, …
  - **regularization**: change network operation / architecture to reduce network capacity
- Early stopping: stop if validation error grows

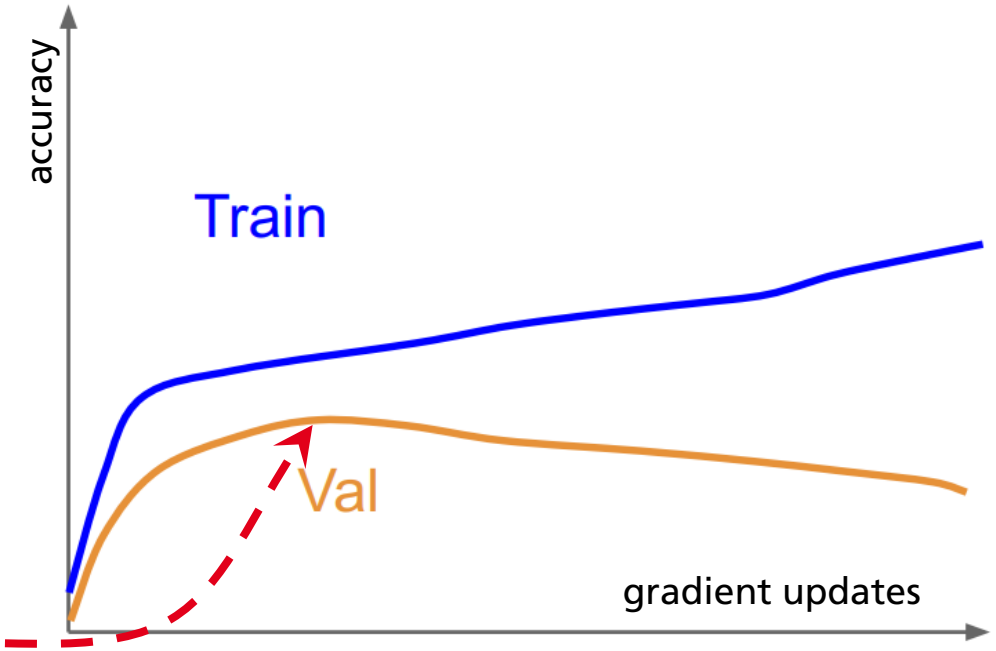


Model does not have capcity to fully learn the data





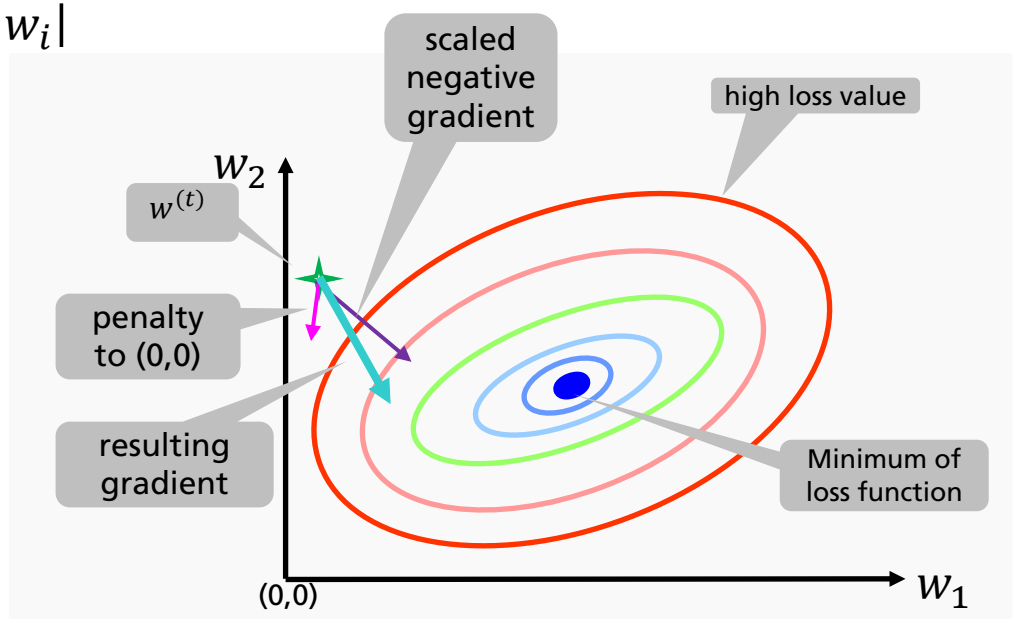
Too complex, extra parameters, fits random fluctuations



Regularization: add penalty to loss

- loss function
  $L(w) = -\log \hat{p}(y_1|x_1; w) - \cdots - \log \hat{p}(y_n|x_n; w) + \rho R(w)$
- penalty term $R(w)$ penalizes large parameter values
  ➔ reduce variability of the model
- L2-regularization: quadratic penalty $R(w) = \sum_{i=1}^{k} w_i^2$
  ➔ weight decay
- L1-regularization: absolute value penalty $R(w) = \sum_{i=1}^{k} |w_i|$
  ➔ sets some parameters to 0.0

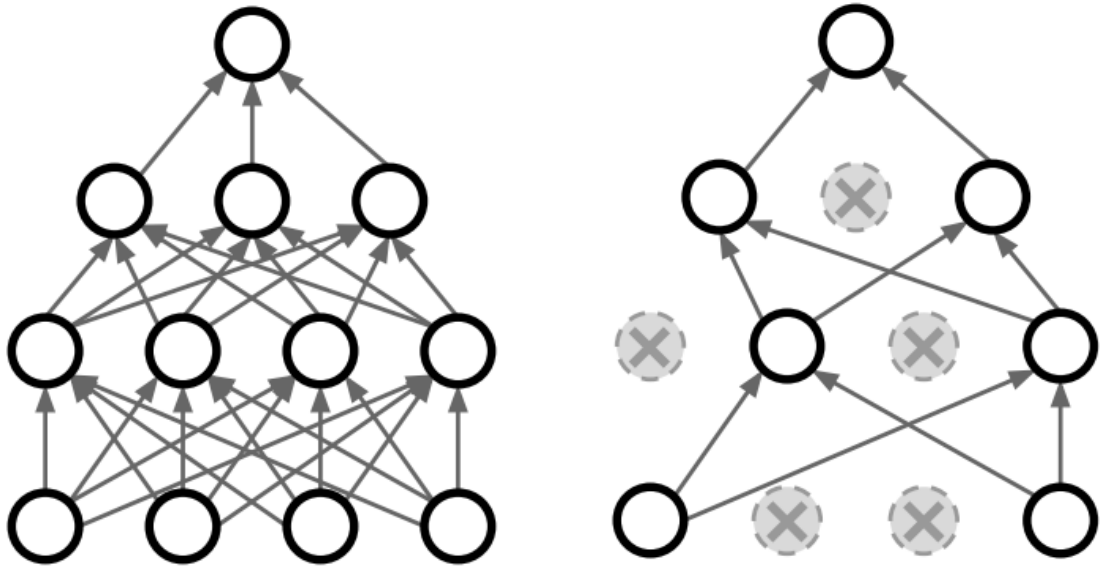
L2-Regularization usually better



Gerhard Paaß © Fraunhofer IAIS

Regularization: Dropout

- In each forward pass, randomly set some components to zero
  - Probability p_{drop} is a hyperparameter;
  - $p_{drop} = 0.5$ is common
- Why does this work?
  - Forces the network to have a redundant representation ;
  - Prevents co-adaptation of features
- Another interpretation:
  - Dropout is training a large **ensemble** of models (that share parameters).
  - Each binary mask is one model
  - allows computation of prediction uncertainty



- Dropout at test time
  - use dropout $p_{drop} = 0.0$
  - on average: activations too high
    scale the activations so that for each component:
    output at test time = expected output at training time

$G_{dropout}$

Regularization: Ensemble of Models

- combine the predictions of several base estimators
  ➔ improve generalizability / robustness over a single estimator.

- **Bagging**:
  - build **several estimators independently**
    e.g. by modifying training data, random parameter initialization
  - **average** their predictions
    ➔ On average, the combined estimator is better than any of the single base estimator because its variance is reduced.

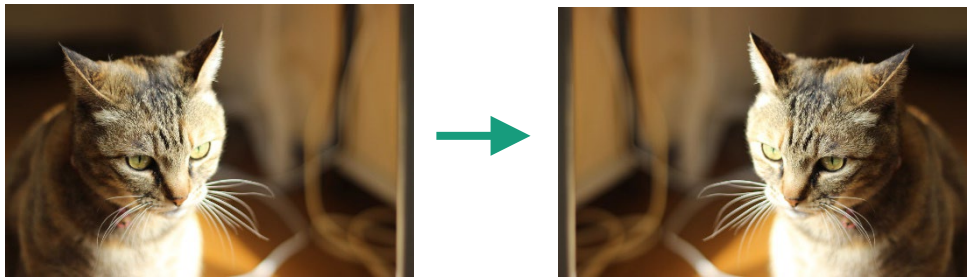
    [Hastie et al. 2009: Elements of Statistical Learning]

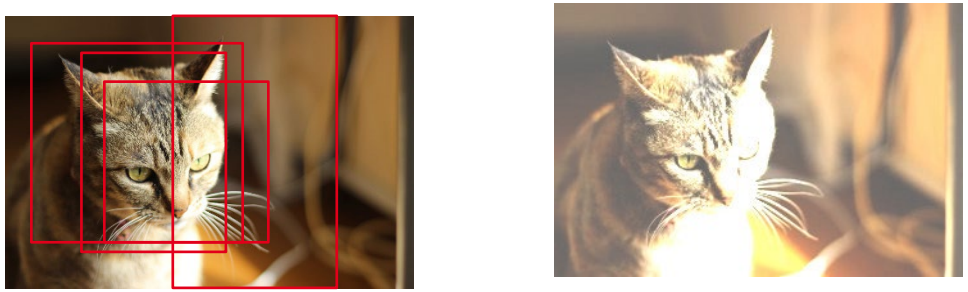
- **Boosting**:
  - base estimators are built sequentially
    give higher weight to **misclassified instances**
  - final estimator: weighted average of all estimators
    ➔ usually better performance than single estimators

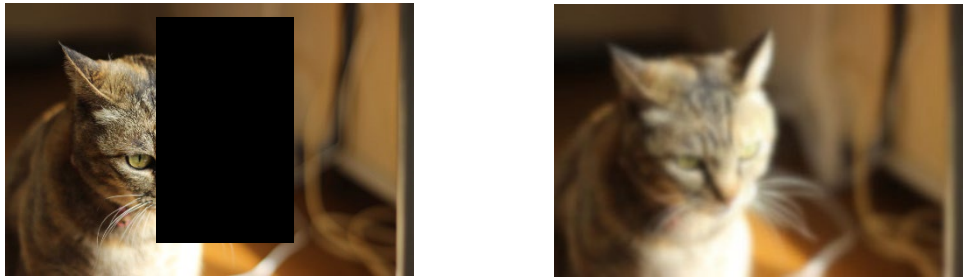
better mean prediction

Regularization: Data Augmentation

- transform input in a way which may occur in reality
- horizontal flips
- random crops and scales
- change contrast and brightness
- set random image regions to zero
- add synthetic observation noise: blur, rain, …







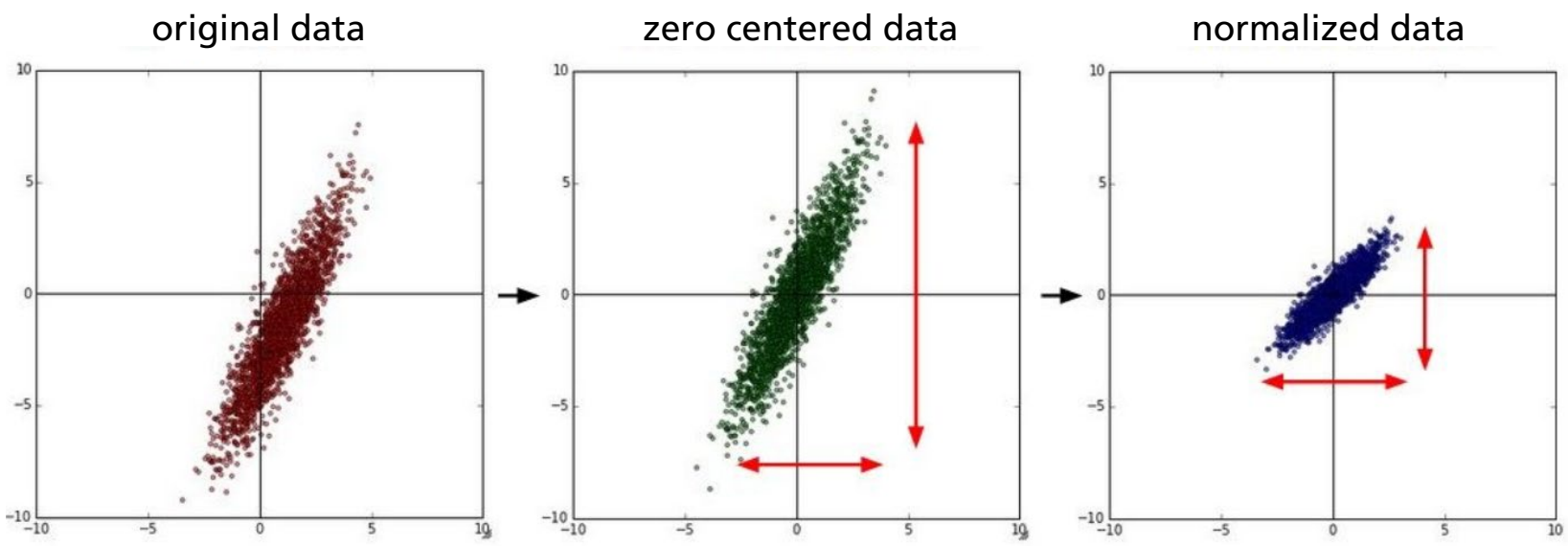
more robust predictions

Regularization: Batch Normalization

- aim: activations with mean 0.0 and unit variance are best
  - no saturation of activations
  - derivatives are in a good range
- transform inputs of a minibatch M
  - $\mu_j = \frac{1}{|M|}\sum_{x_i \in M} x_{i,j}$ mean for each component of $x_i \in M$
  - variance for $x_{*,j} \in M$
    $\sigma_j^2 = \frac{1}{|M|}\sum_{x_i \in M}\left(x_{i,j} - \mu_j\right)^2$
  - normalized input
    $\tilde{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$
- testing
  - use fixed stats to normalize

less sensitive to small changes in weights
easier to optimize

? 03-c



C batch normalization

Deep Learning Building Blocks
Agenda

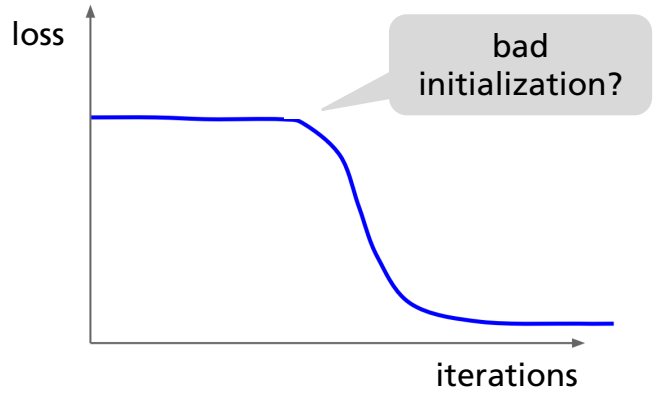
1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

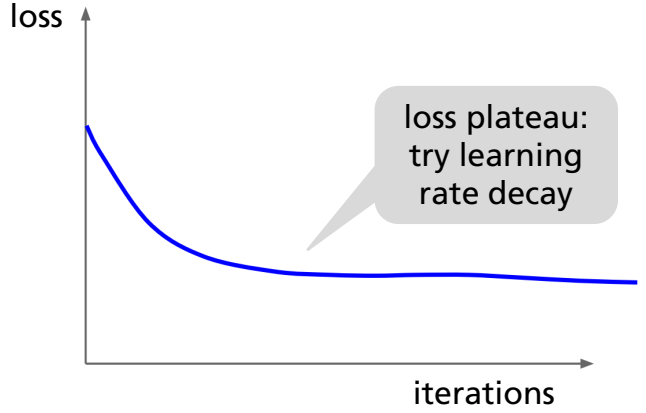
Model Tuning

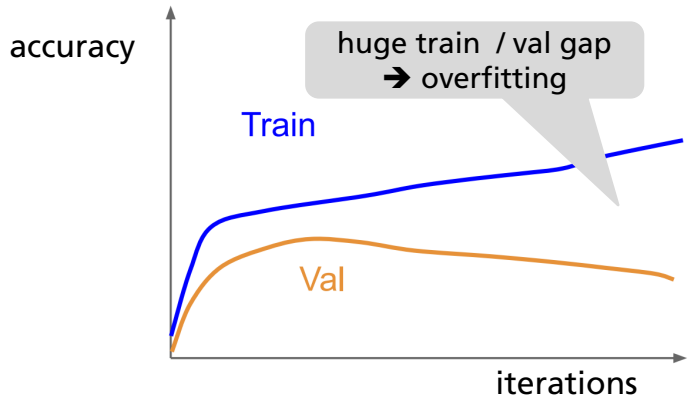
- Step 1: Overfit a small sample
  - Try to train to 100% training accuracy on a small sample of training data (~5-10 minibatches); fiddle with architecture, learning rate, weight initialization
  - Possible problems: Loss not going down? ➔ Learning rate too low, bad initialization
    Loss explodes to Inf or NaN? ➔ Learning too high, bad initialization
- Step 2: Find learning rate that makes loss go down
  - Use all training data, turn on small weight decay, find a learning rate that makes the loss drop significantly within ~100 iterations
  - Good learning rates to try: 0.1, 0.01, 0.001, 0.0001
- Step 3: Joint search for learning rate and regularization parameter – train for 1-5 epochs
  - e.g. weight decay rates 0.00001, 0.000001, 0
- Step 4: Refine grid, train longer, no learning rate decay – train for 10-20 epochs

Model Tuning

- Step 5: Look at learning curves
  - no immediate decrease of loss: change initialization
  - loss stops to decrease: reduce learning rate
  - loss stops to go down after learning rate reduction: learning rate reduction too early
  - both training and validation accuracy are still increasing: train longer
  - training accuracy goes up while validation accuracy goes down: overfitting ➔ more regularization, use more data (if possible)







Hyperparameters

- Many hyperparameters have to be set
  - start with recommended values from github / publications
  - perform optimization if necessary

structure hyperparameters

- number of layers
- size of hidden vectors
- type of activation function
- type of regularization
- initialization parameters
- …

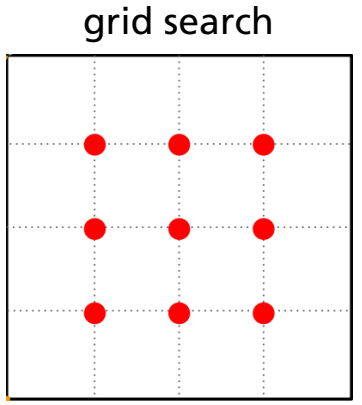
optimization hyperparameters

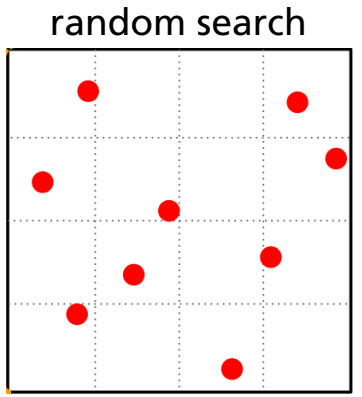
- learning rate
- LR decay, momentum, …
- batch size
- regularization parameters, e.g. dropout probability
- number of epochs
- …

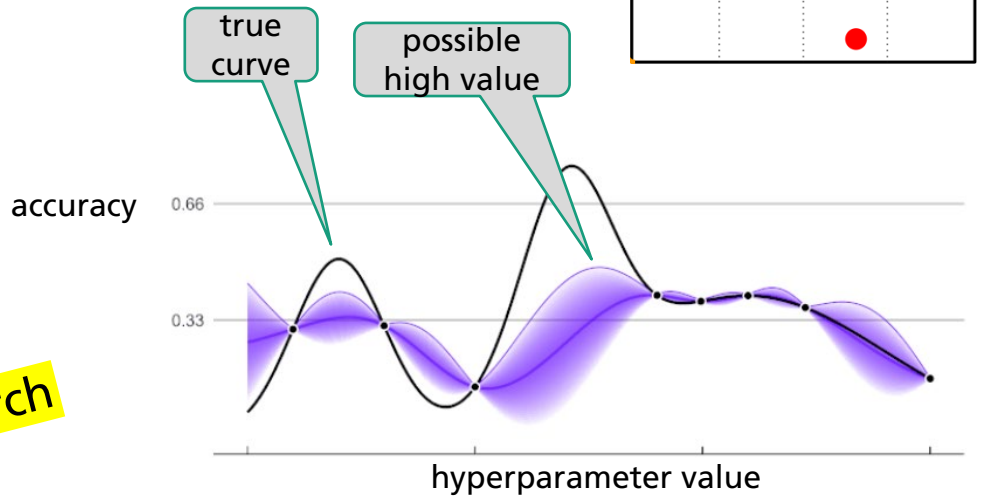
C hyperparameter

Hyperparameter Optimization

- May use algorithmic approach
- Grid search: systematically search grid of hyperparameter value combinations
  ➔ inefficient for more than 2 parameters
- Random search: select parameters independently at random
  ➔ much more efficient for several parameters
- Model-based search:
  create a model which predicts the the uncertainty of performance estimate (e.g. Bayes Regression)
  ➔ select new parameter at points where value may be high
- **Autokeras** uses this approach for continuous and discrete hyperparameters







Neural Architecture Search

C. hyperparameter optimization

Design Guide

- Examine data
  - What data is available: no. of observations
  - data quality, measurement errors
  - determine a cost criterion
  - preprocessing, scaling, error checking
- If possible, do not start from scratch:
  use model architectures from github, etc.
- Model tuning
  - as discussed before
- Parallelization and deployment
  - if initial experiments are successful

Number of Parameters vs. Training Set Size

- Larger Deep Neural Networks yield a better performance with enough data.
  - Number of parameters for text models: BERT large 1.5B, GPT-2 1.7B, T5 11B, GPT-3 170B
- [Kaplan et al. 2020] and [Hoffmann et al. 2022] empirically evaluated dependency between number of parameters N, size of training data D, amount of compute effort C

- If **N and D are increased at the same rate**, the model accuracy grows reliably.
  - for every doubling of model size N the number of training tokens D should also be doubled.

- **Large models** are better able to extract information from data than small models

  ➔ reach the same level of accuracy with fewer optimization steps and using fewer data points.
  - If **compute time** is fixed, but no restrictions on size or data,
    ➔ use very large models and stop before convergence.
  - The optimal **batch size** depends on the gradient noise,
    which is easy to measure during training and is larger than assumed before.

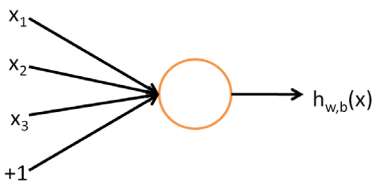
? 03-d

Deep Learning Building Blocks
Agenda

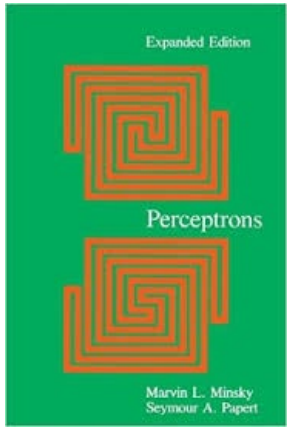
1. Neural Networks correspond to a Mapping
2. Generating High-dimensional Outputs
3. Construct better Features
4. Basic Architecture of Deep Neural Networks
5. Optimization by Gradient Descent
6. Overfitting and Regularization
7. Hyperparameter Tuning and Design Guide
8. Summary

History of Neural Networks

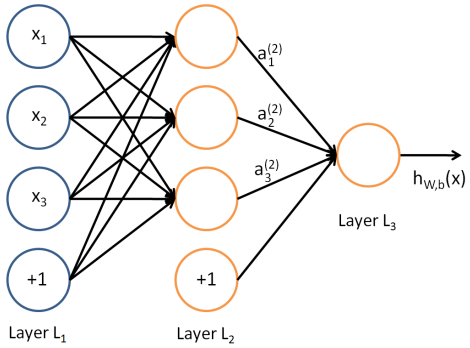
Rosenblatt's
perceptron



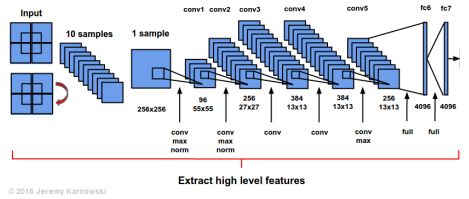
Minsky causes
'AI Winter'



Rumelhart & Hinton
Multi-layer perceptron & backpropagation

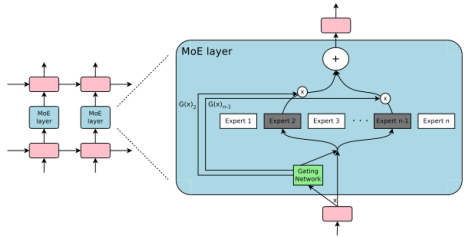


Kernel Methods (SVM) take over



NN comeback 2

AlexNet wins 2012 ImageNet competition



Very large + complex architectures:

- *Networks of Networks*
- *Generative Models*
- *Transformers*
- *...*

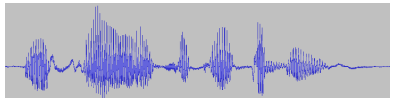
1957 — 1969 — 1986 — ~1995 — ~2010+ — Today

Summary

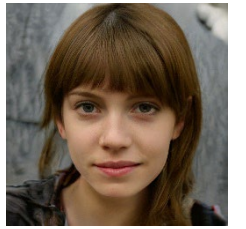
- Deep neural networks are functions $y = f(x, w)$ with parameter w
  - different types if inputs x and ouputs y (images, sound, text) converted to tensors
  - often many layers: generate features which are better for prediction / classification
- many types of layers
  - linear layers, nonlinear activation function
  - loss functions measure difference between prediction and observed output
- Optimization by stochastic gradient descent
  - compute derivative by backpropagation
  - select minibatch and change w by negatitive gradient multiplied with learning rate
- Handle overfitting and underfitting
  - compute learning diagnostics: validation accuracy & loss
  - use regularization: weight decay, dropout, layer normalization, …
  - adapt architecture and hyperparameters

Lioness at the Louisville Zoo by Ltshears / CC BY-SA 3.0

Public domain



Grafik als Beispiel für eine kontinuierliche Phrase by Mwka / CC BY-SA 3.0



Public domain

References

Blick 17, http://www.blick.ch/auto/news_n_trends/audi-q7-deep-learning-concept-das-auto-lernt-vom-fahrer-id6004071.html. Download on 06.01.2017

Castelvechii, D. "Deep Learning boosts Google translate Tool." Nature, 27.Sep. 2016

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. EMNLP 2014

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1026-1034).

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

Martin, J. "Why you might want to hold off buying Google Home" http://www.cio.com/. 7.Nov.16

Mikolov, T., and J. Dean. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems (2013).

Mirończuk, M., (2017): The BigGrams: the semi-supervised information extraction system from HTML: an improvement in the wrapper induction. Knowl Inf Syst (2018) 54: 711. https://doi.org/10.1007/s10115-017-1097-2

Microsoft NLP-group (2017): R-NET: Machine reading Comprehension with self-matching Networks

Reiley, C. "Deep Driving", MIT Technological Review, 18.Oct. 2016

Rumelhart, D.E; James McClelland (1986). Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Cambridge: MIT Press.

Schuster, M. "Zero-Shot Translation with Google's Multilingual Neural Machine Translation System", Google Research Blog , 22. Nov. 2016

Shazeer et al. 2017: Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.

Silver, D. et al. (2016): Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

Disclaimer

Responsible contact:
E-mail: