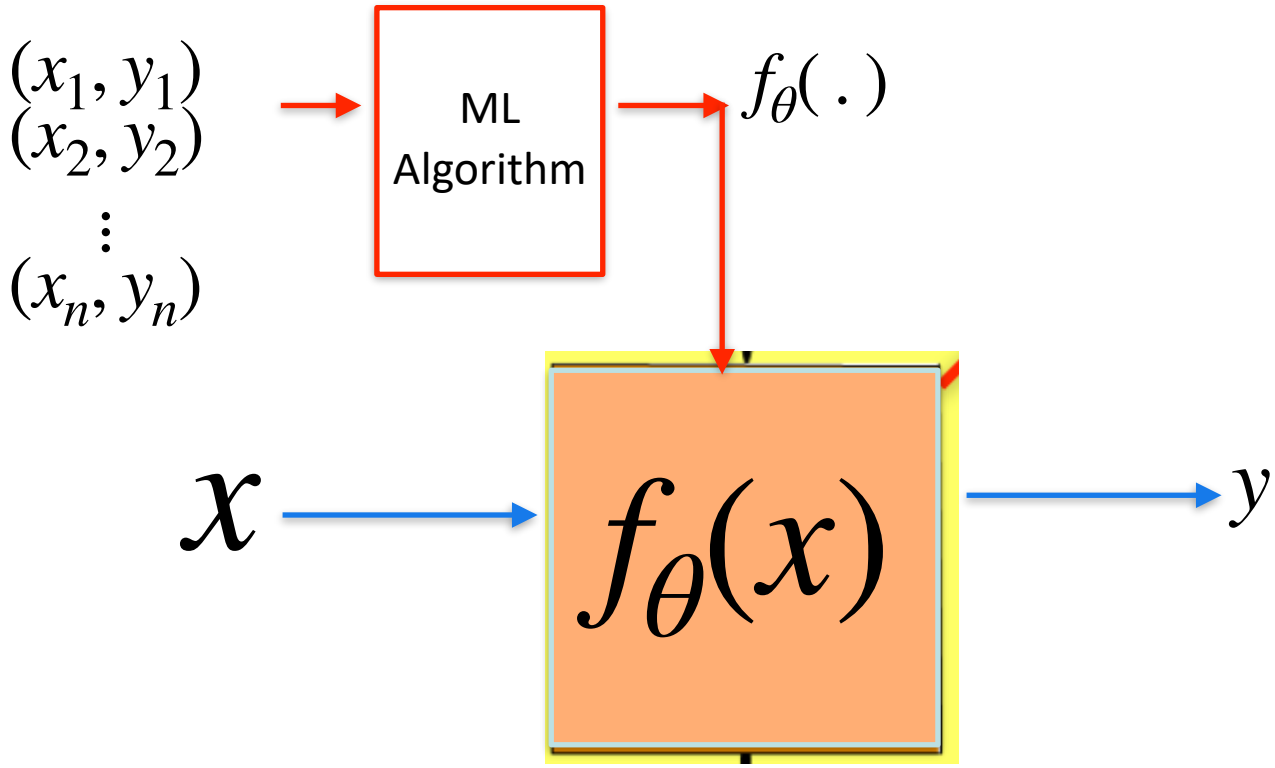
Course
Knowledge-based Systems

Lecture 11 – Deep Learning

Dr. Mohsen Mesar

Universität Duisburg-Essen

Recall



Recall

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- In ML, we conduct the following steps:
  - Find/Collect a **dataset**
  - Define **features** to encode data samples into numerical vectors that are understandable for machines
  - Identify **the type of task**: classification, regression, clustering, …
  - Identify the what **ML model** to use: SVM, Decision tree, kNN,…
  - **Evaluate** your model using **a concrete metric** (e.g., accuracy or F-measure,…) using **k-fold cross validation** or **an unseen test set**
  - **Compare** the performance of **your model with intuitive baselines** and **state-of-the-art models** for your task
  - **Improve** your model
    - **Better features** or **representations** of data samples
    - Better **ML model (either existing ones or introduce new algorithms)**

Any other open questions?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

In this lecture, you learn …

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

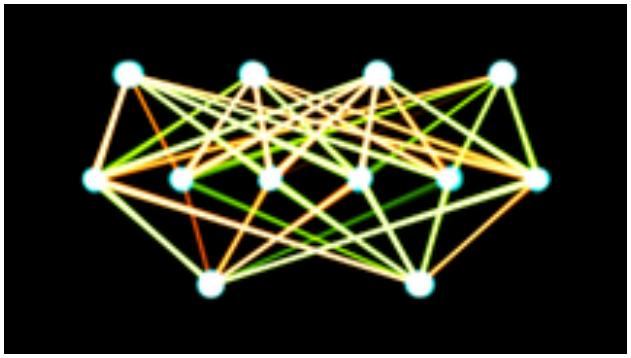
- Deep learning (DL) as a subfield of ML
- History of DL
- Basics of DL models
- How to train DL models

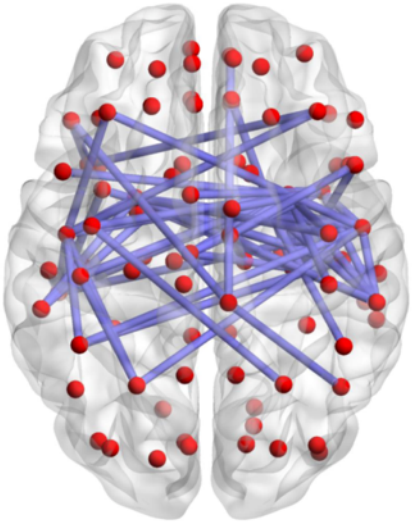
Deep Learning (DL)

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Subfield of machine learning
- Neural networks: a brain-inspired metaphor for a computational model





https://upload.wikimedia.org/wikipedia/commons/0/0e/Brain_network.png
https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/Single-layer_feedforward_artificial_neural_network.png/214px-Single-layer_feedforward_artificial_neural_network.png

Classical vs. Deep Learning

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Analyze the data
- Develop good features
  → requires domain knowledge
  → very tedious and time-consuming
  → very often, features cannot be reused for another task

Classical vs. Deep Learning

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

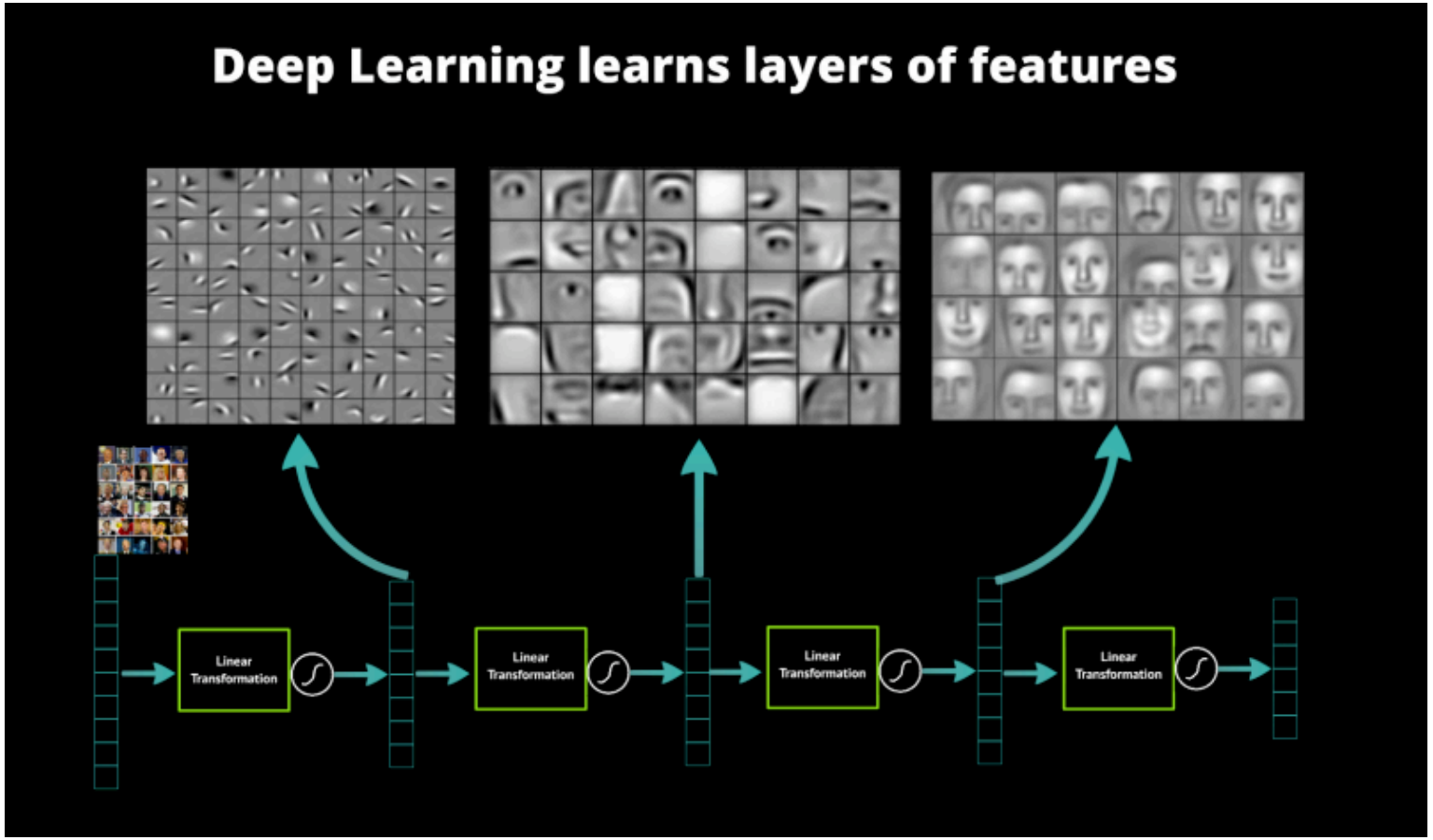
- Analyze the data
- Develop good features
  → requires domain knowledge
  → very tedious and time-consuming
  → very often, features cannot be reused for another task

- **Idea:** automate the feature development
  →start with very basic low-level representations
  →identify relevant patterns in the data

Learned Representations

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken



Source: https://becominghuman.ai/what-is-deep-learning-and-its-advantages-16b74bc541a1 and Lee et al. (ICML 2009)

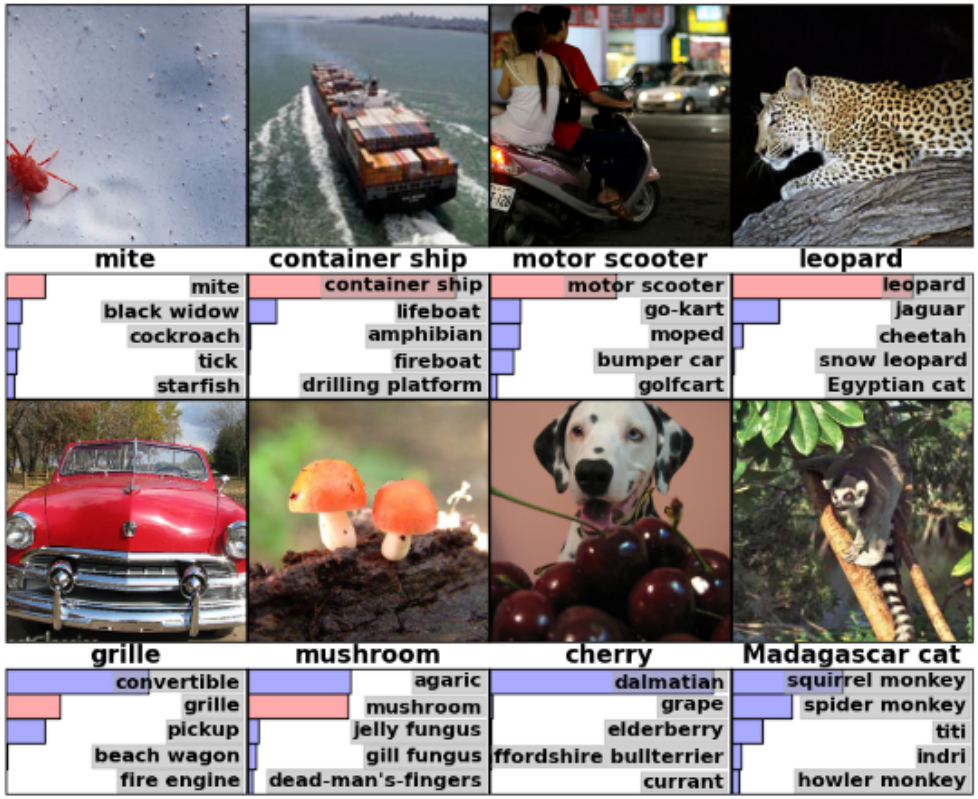
DL Example: Object Recognition

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2012): *ImageNetClassification With Deep Convolutional Neural Networks*

Figure 4:



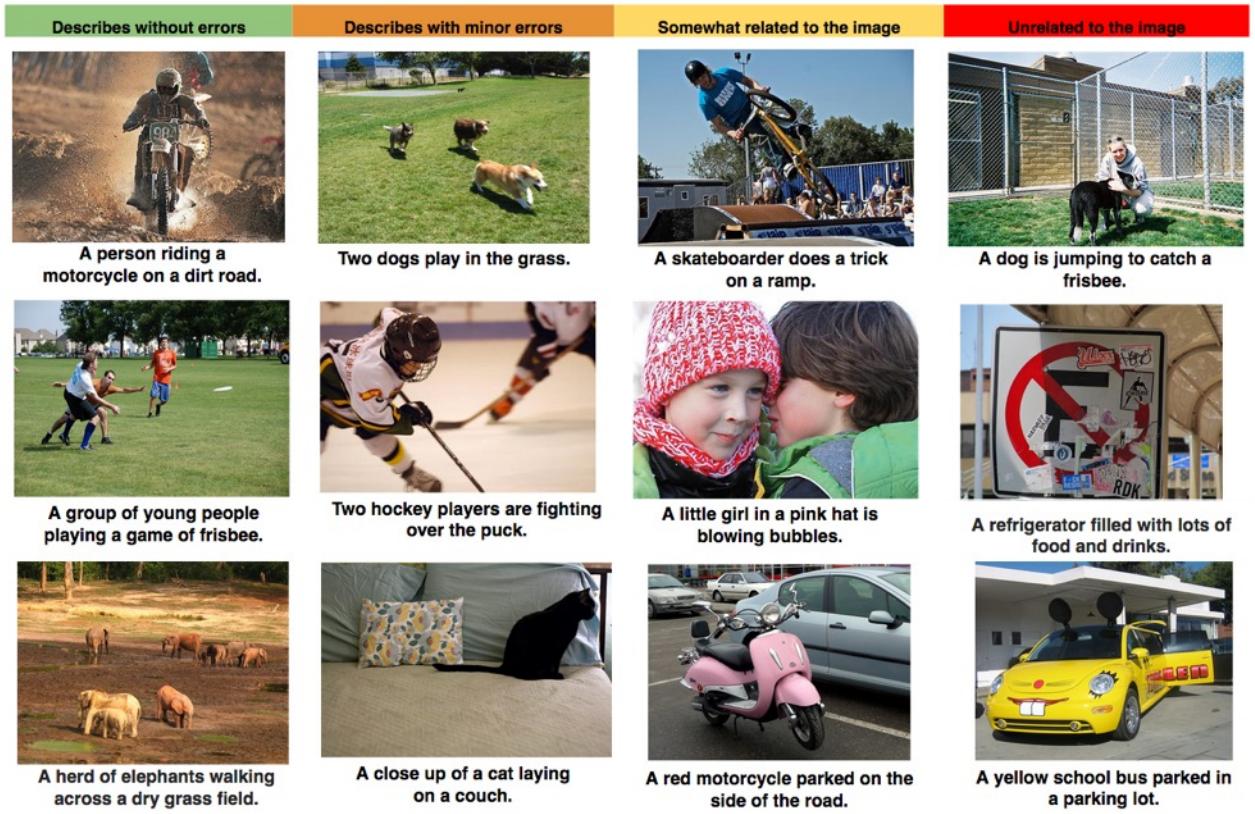
DL Example: Image Description

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

GoogLeNet 2014: http://googleresearch.blogspot.de/2014/09/

Building deeper understanding of images

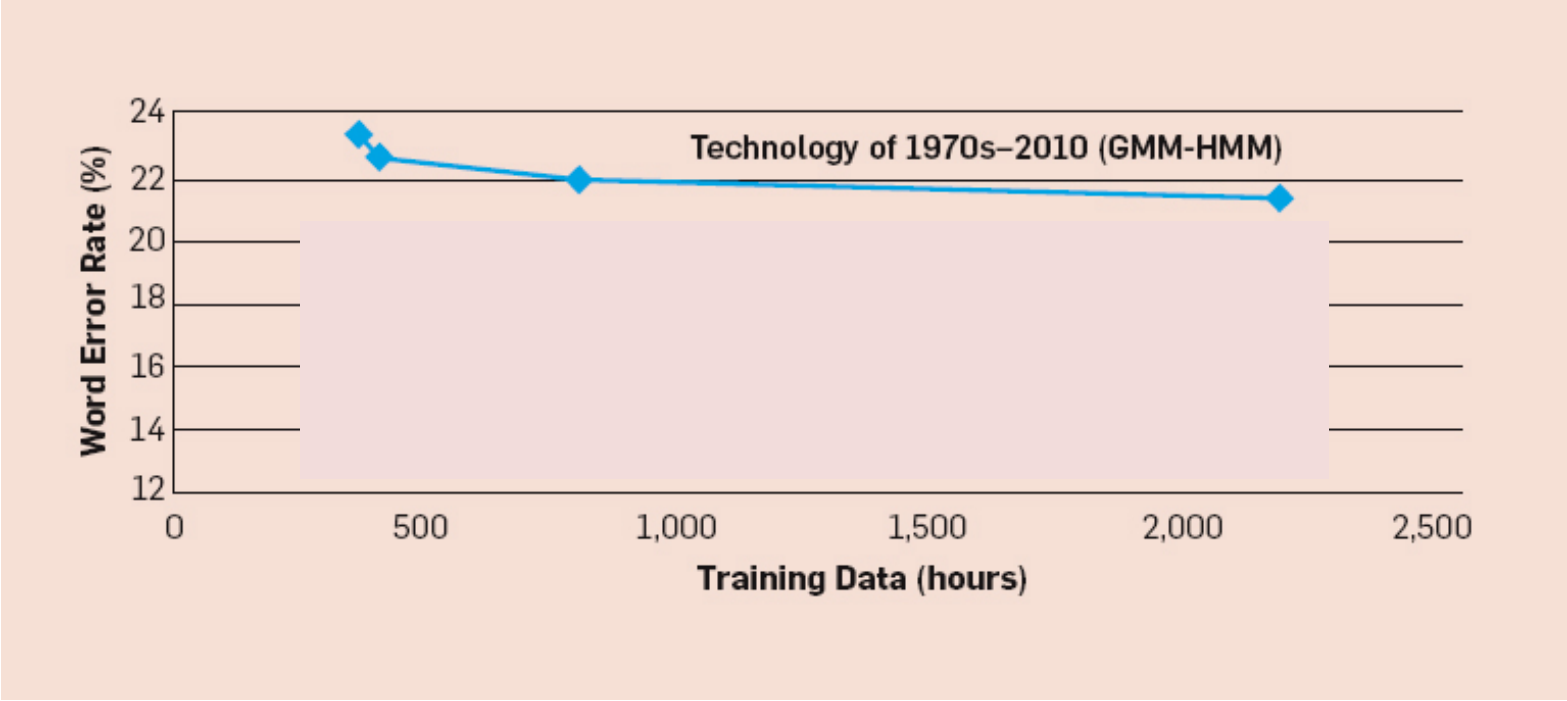


Speech Recognition

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Xuedong Huang, James Baker, Raj Reddy:
  A Historical Perspective of Speech Recognition

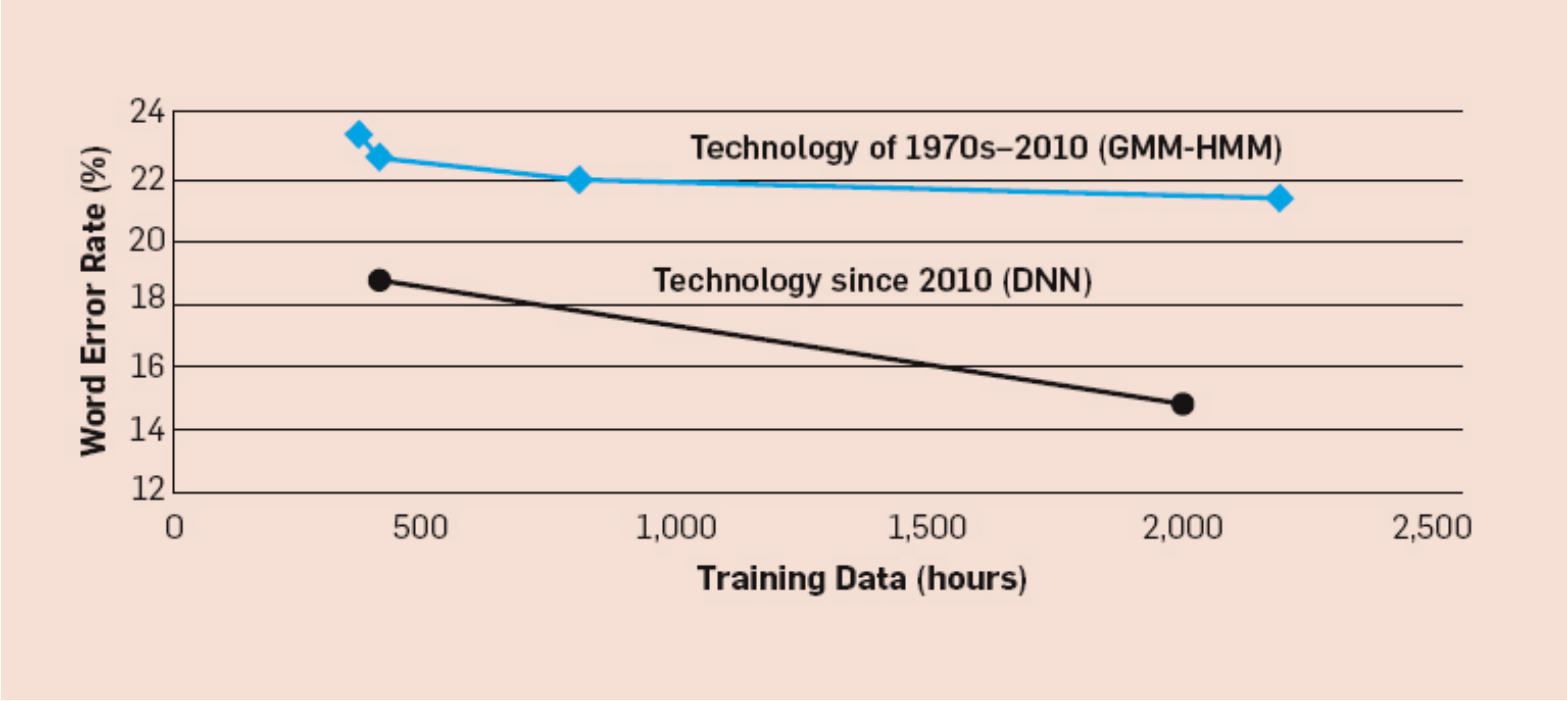


What happened in 2010?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Abdel-rahman Mohamed, Dong Yu, Li Deng (2010): *Investigation of Full-Sequence Training of Deep Belief Networks for Speech Recognition*

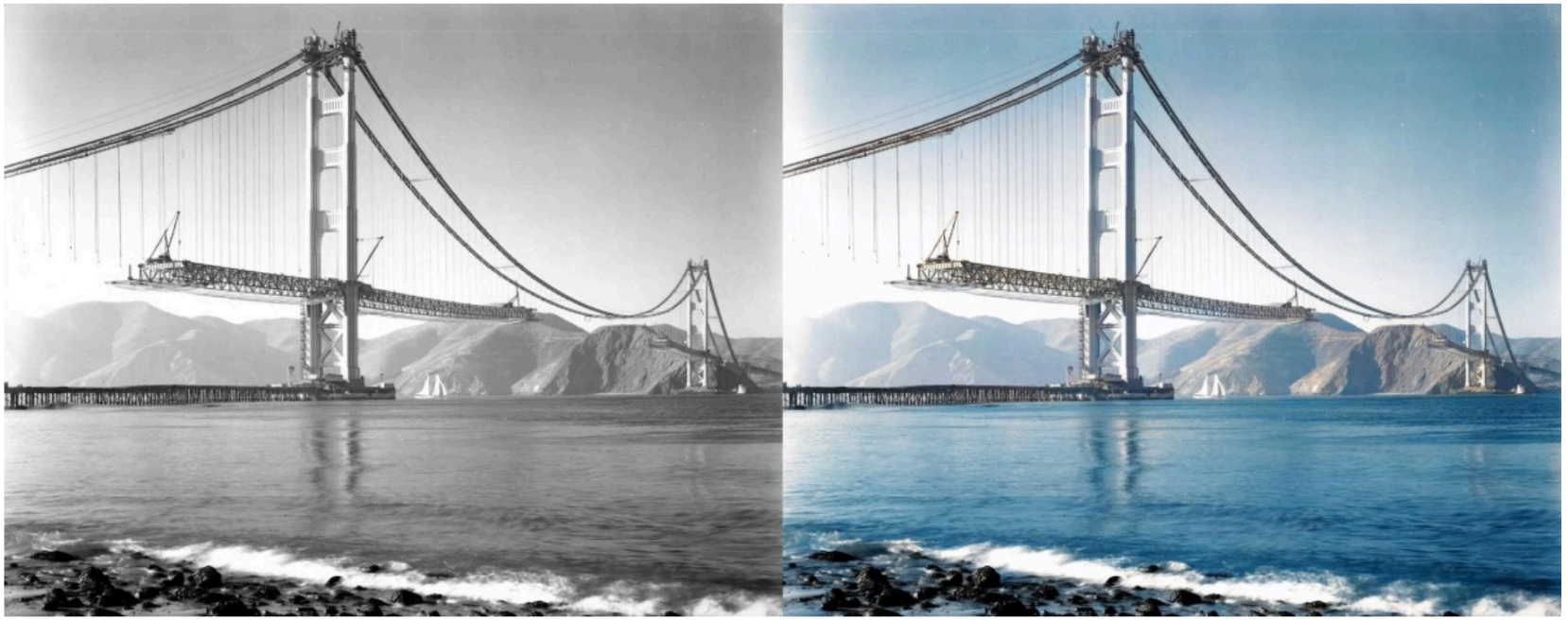


Cool Projects

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- https://github.com/jantic/DeOldify
- Colorize Images



Cool Projects

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- https://github.com/lengstrom/fast-style-transfer

Add styles from famous paintings to any photo in a fraction of a second

Why are neural networks so powerful?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Non-linear models

- Most problems do not follow linear relationships

Deep structure (hidden layers)

- Reduce **feature engineering**
- **Learn** from **low-level input**
- **Extract patterns** in the data that humans were not able to anticipate

Powerful architectures

- Recurrent neural networks,
- LSTMs,
- convolutional networks,
- etc.

In return, …

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

You need

- **a lot of data and** a lot of **computing power**
- Understanding of **which network architecture works** best for your problem
- **a lot of time/money to optimize your networks**

As you don’t design the features, the outcome **is really hard to interpret.** What did the network learn?

History of DL

History of Deep Learning

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Early booming (1950s – early 1960s)

- Rosenblatt (1958) - Perceptron
- Widrow and Hoff (1960, 1962) - Learning was rule-based on gradient descent

Setback (mid 1960s – late 1970s)

- Serious problems with perceptron model (Minsky's book 1969)
  → It can't even represent simple functions

Renewed enthusiasm (1980s)

- New techniques (backpropagation for "deep nets")

Deep Learning

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Out-of-fashion again (1990s – mid 2000s)

- Other techniques were considered superior and more understandable
  - e.g., Support Vector Machines (SVMs)
- Neural networks were so out of fashion some good CS journals immediately rejected papers on neural networks without review (as reported by Geoffrey Hinton, and others)
- Played no role in top NLP conferences

Since mid 2000: huge progress for “deep learning”

- Hinton and Salakhutdinov (2006): one can actually train deep nets
- Since 2013: Veritable hype in NLP, mostly due to word embeddings

Deep Learning

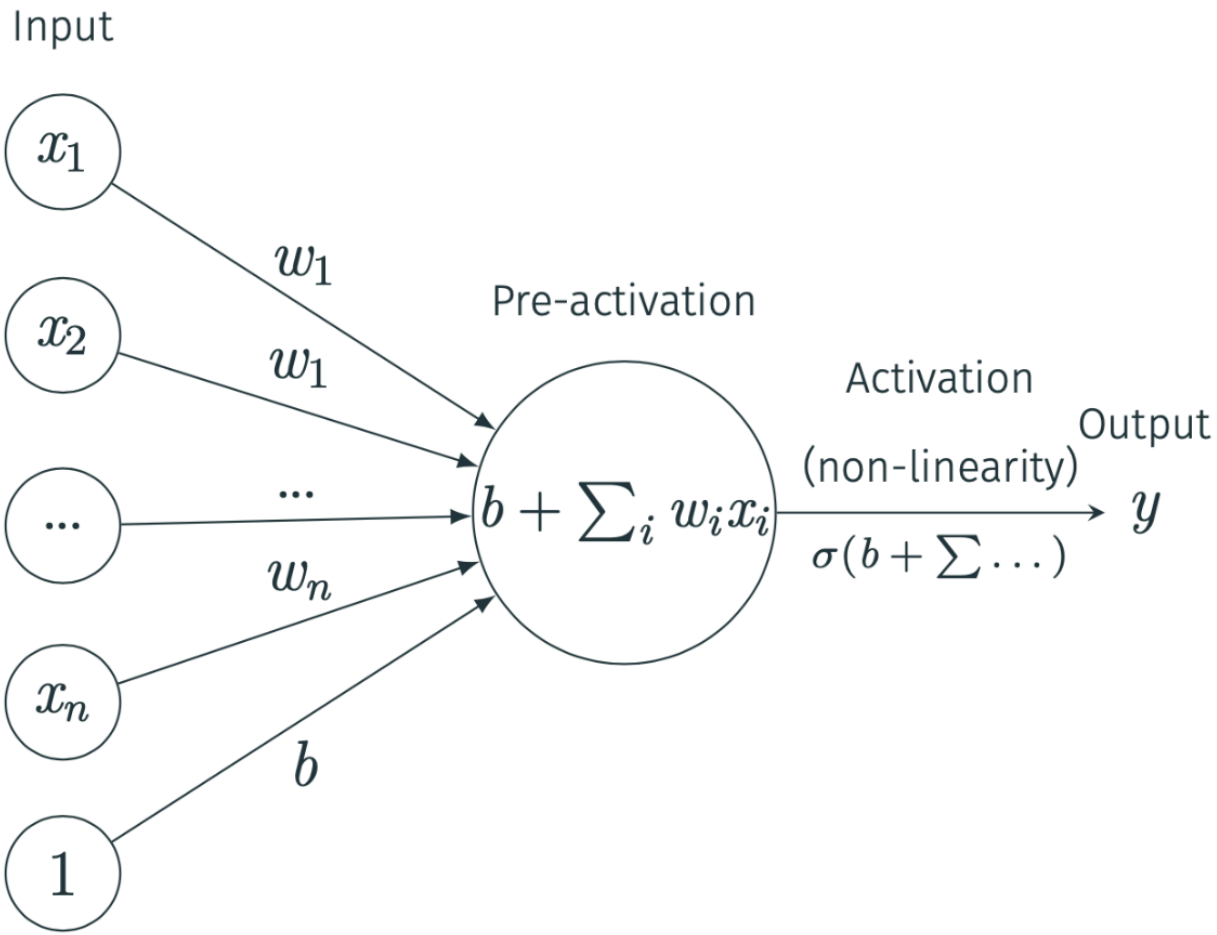
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Since mid 2000: huge progress for “deep learning”
  - Why?
    - More data
    - Faster compute power, and more suitable hardware (GPU)
    - Better optimization techniques

Basics of DL models

A computational neuron



Formally

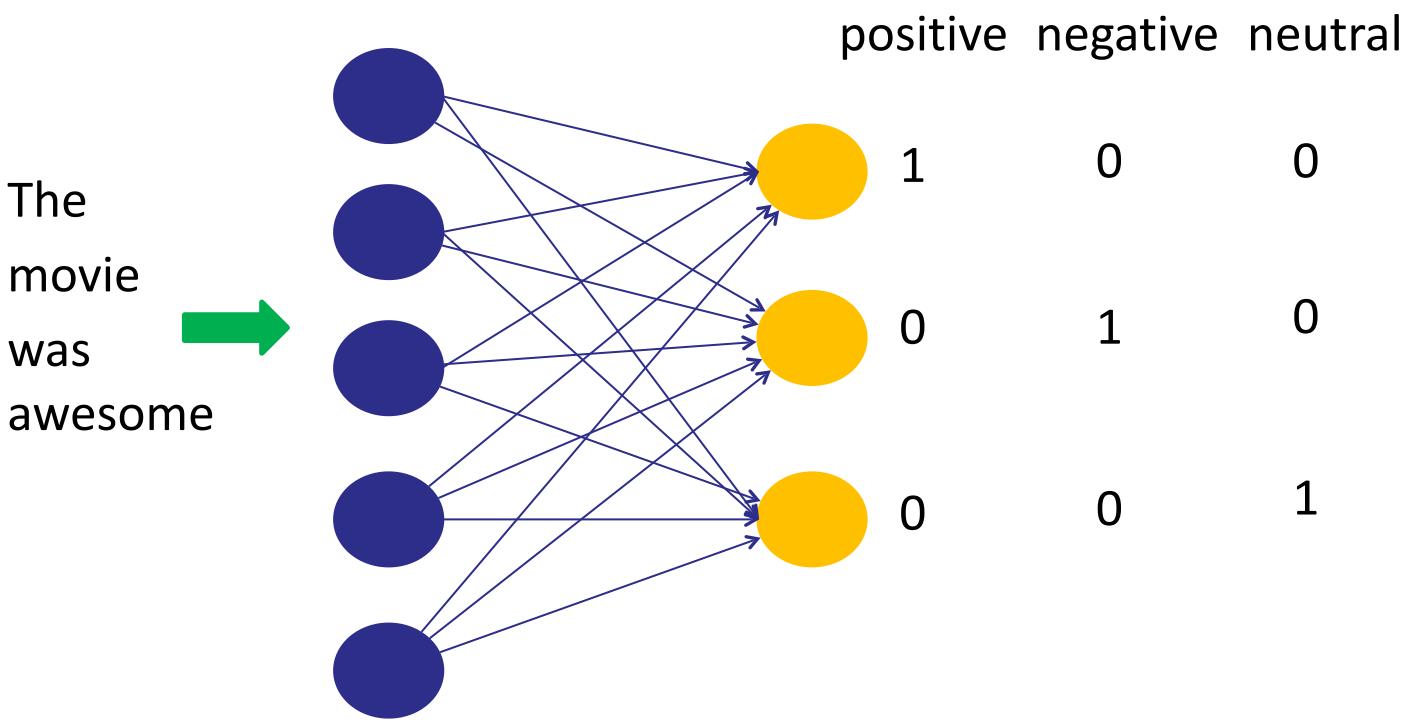
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- For simplicity, we add the constant bias $\mathbf{b}$ simply as an additional entry to the input vector $\mathbf{x}$
- Input: $\mathbf{x}$
- Parameters: $\mathbf{w}$
- Activation function: σ
- Output: $\sigma(w \bullet \mathbf{x})$

- $\mathbf{x}$ is determined by our data
- $\mathbf{w}$is a parameter of the model (often also denoted as θ)
- σ is a hyper-parameter

(Simple) Neural network



More complex neural network

UNIVERSITÄT DUISBURG ESSEN

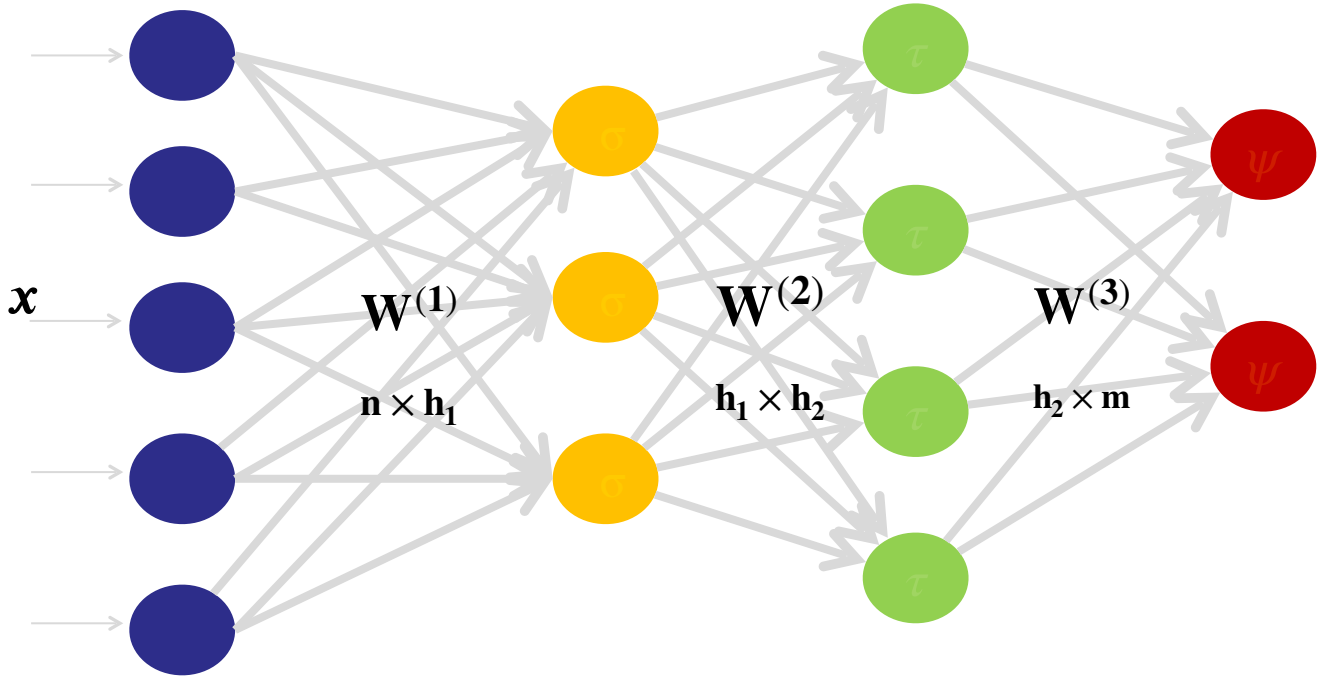
Offen im Denken

Multiple hidden layers (multilayer perceptron)

The output of the previous layer serves as the input of the next layer.

→ forward propagation

Many more parameters and choices.

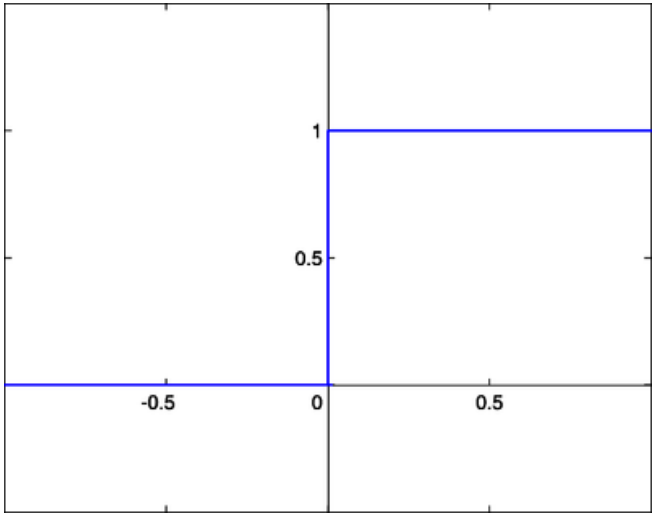


Activation function

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- The activation function is crucial.
- Given the cumulated weighted input, it computes the output.
- In a binary function: *"firing or not firing"*

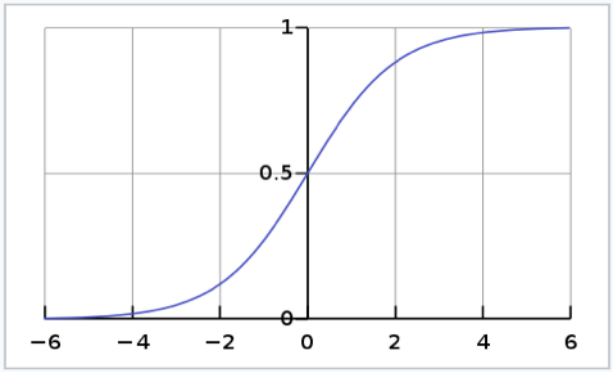


Sigmoid & Tanh

UNIVERSITÄT DUISBURG ESSEN

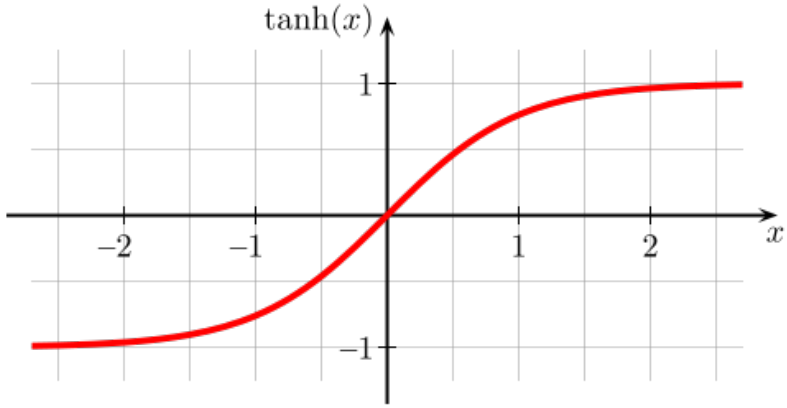
Offen im Denken

- Sigmoid function (scale: 0 to 1)



$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}$$

- Hyperbolic tangent (scale: -1 to 1)



$$\tanh x = \frac{\sinh x}{\cosh x}$$

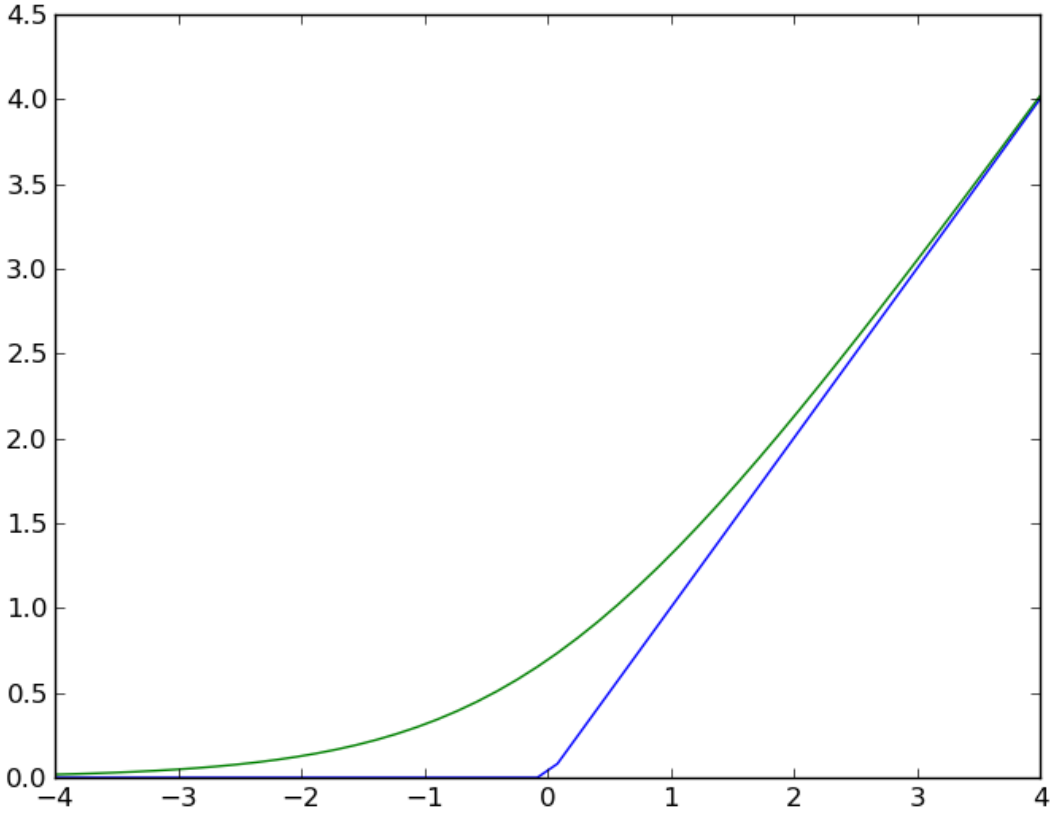
https://en.wikipedia.org/wiki/Sigmoid_function
https://de.wikipedia.org/wiki/Tangens_hyperbolicus_und_Kotangens_hyperbolicus

Rectified linear units (ReLu)

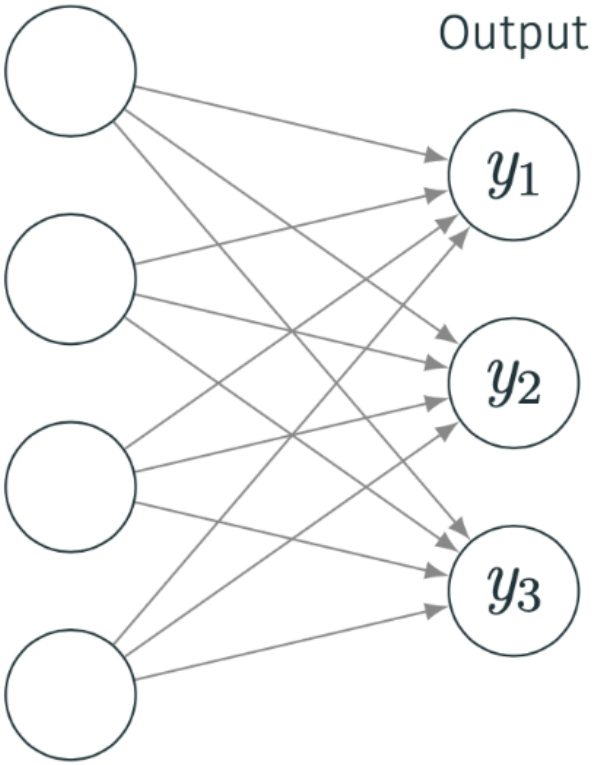
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Rectifier (blue): $f(x) = \max(0,x)$
- Smooth approximation (green): $f(x) = \ln(1 + e^x)$



Softmax



Classification into multiple output classes (e.g., „Positive“, „Negative“, „Neutral“), we’d often like our outputs to represent a probability distribution over these classes

$$y_1 + y_2 + y_3 = 1; \quad y_j \geq 0$$

Let z_1 be pre-activation of output node y_1

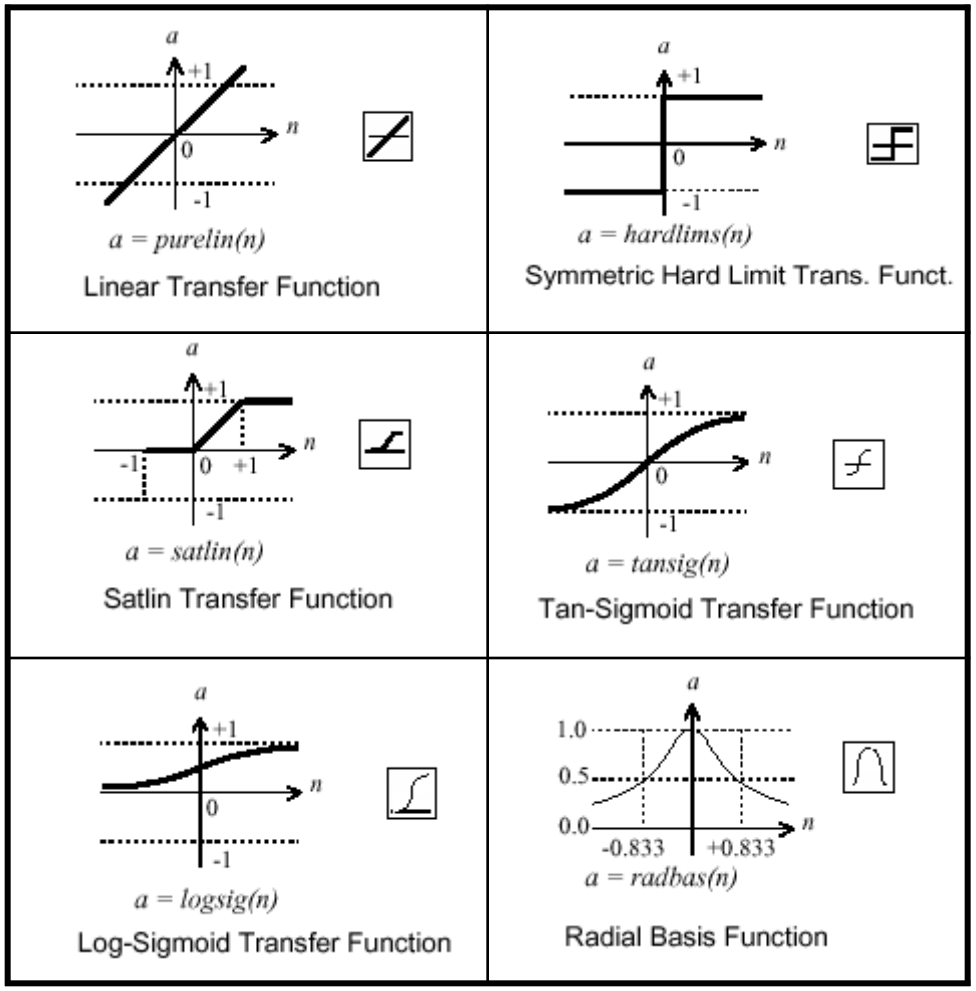
$$y_1 = \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2) + \exp(z_3)}$$

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{m} \exp(z_i)}$$

Many more…

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken



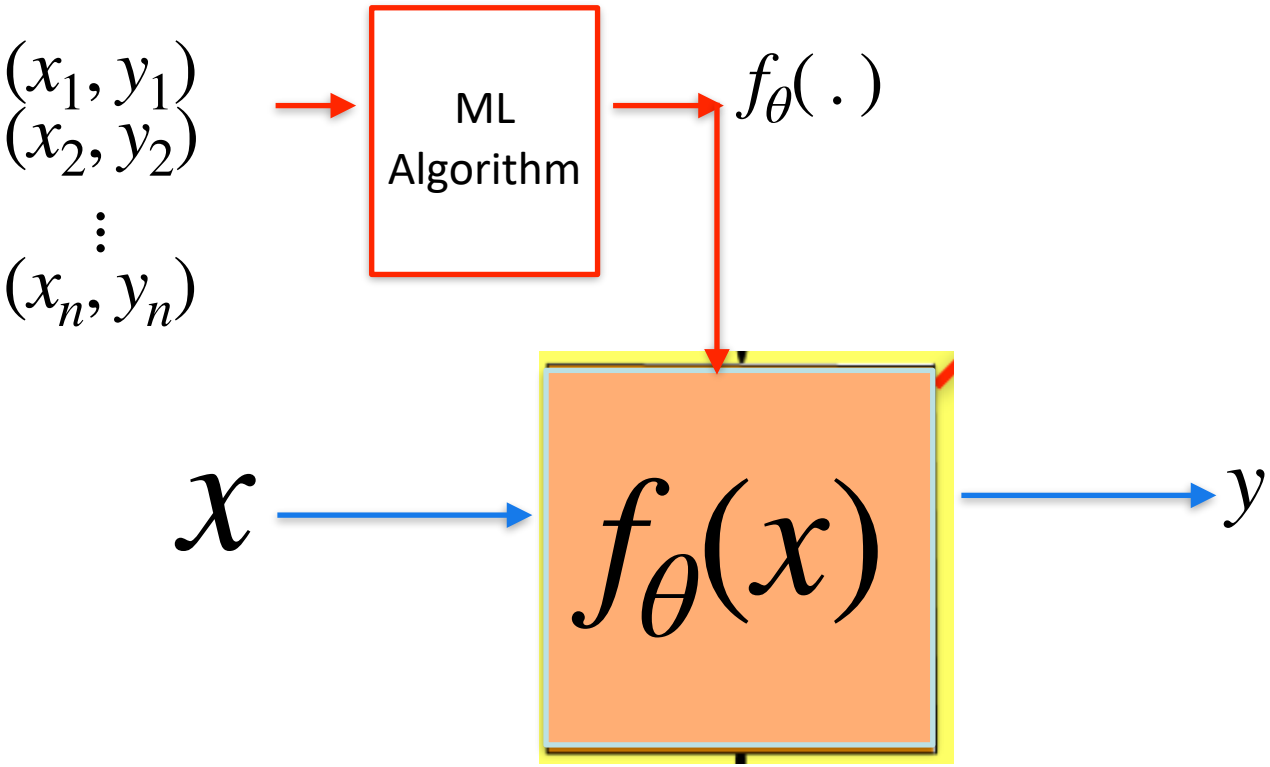
- Finding the ”best” activation function is still a question of tedious experimentation.

Recall Learning

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

How to use data samples in a dataset to learn function $f_\theta(\,.\,)$?



Supervised Learning (SL)

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- Training data samples x_i with reference labels y_i
- SL algorithms try to learn function $f_\theta(x)$ that approximates the data

$$
\begin{array}{ll}
x_1, y_1, & \hat{y}_1 = f_\theta(x_1) \\
x_2, y_2, & \hat{y}_2 = f_\theta(x_2) \\
\vdots & \\
x_n, y_n, & \hat{y}_n = f_\theta(x_n)
\end{array}
$$

Model's predictions

such that

The sum of the errors between $f_\theta(x_i)$ and y_i is minimum

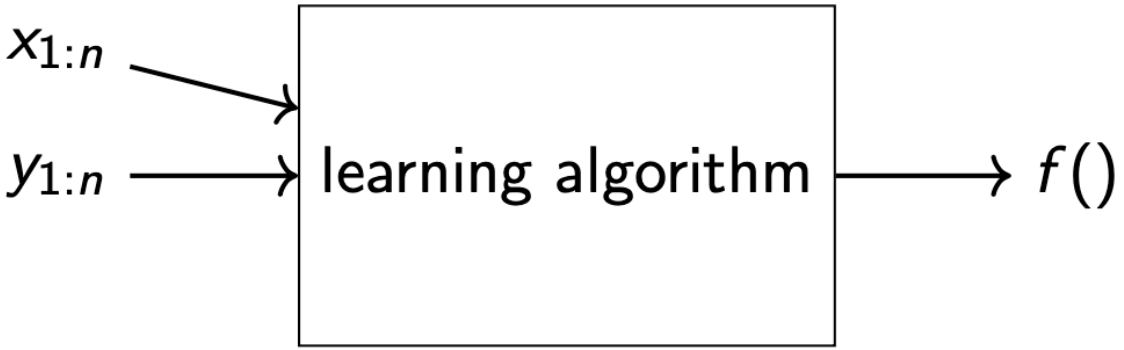
The learned function predict y for x which has not been seen in the training data.

SL

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels



- how to measure if $f()$ works accurately?

Loss Function

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- *Loss function* $L(y, \hat{y})$: It quantifies the loss suffered when predicting ŷ while the true label is y
- given a labeled training set $(x_{1:n}; y_{1:n})$, a per-instance loss function L and a parameterized function $f(x; \Theta)$, we define the corpus-wide loss with respect to the parameters as the average loss over all training examples:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(\hat{y}, y_i),$$

$$\hat{y} = f(x_i; \Theta)$$

Training as Optimization

- the goal of the training algorithm is then to set the values of the parameters such that the value of $\mathcal{L}$ is minimized

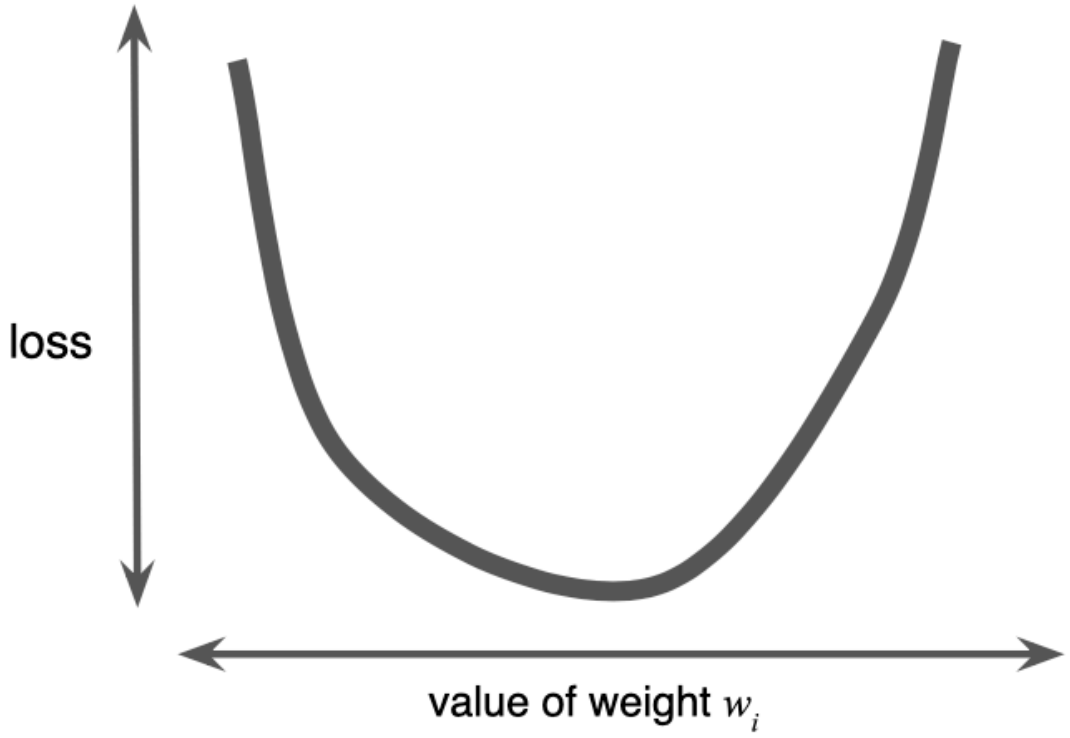
$$\hat{\Theta} = \text{argmin}_{\Theta}\mathcal{L}(\Theta) = \text{argmin}_{\Theta}\frac{1}{n}\sum_{i=1}^{n} L(\hat{y}, y_i)$$

- how to find parameter values that minimize loss?

Gradient-based Optimization

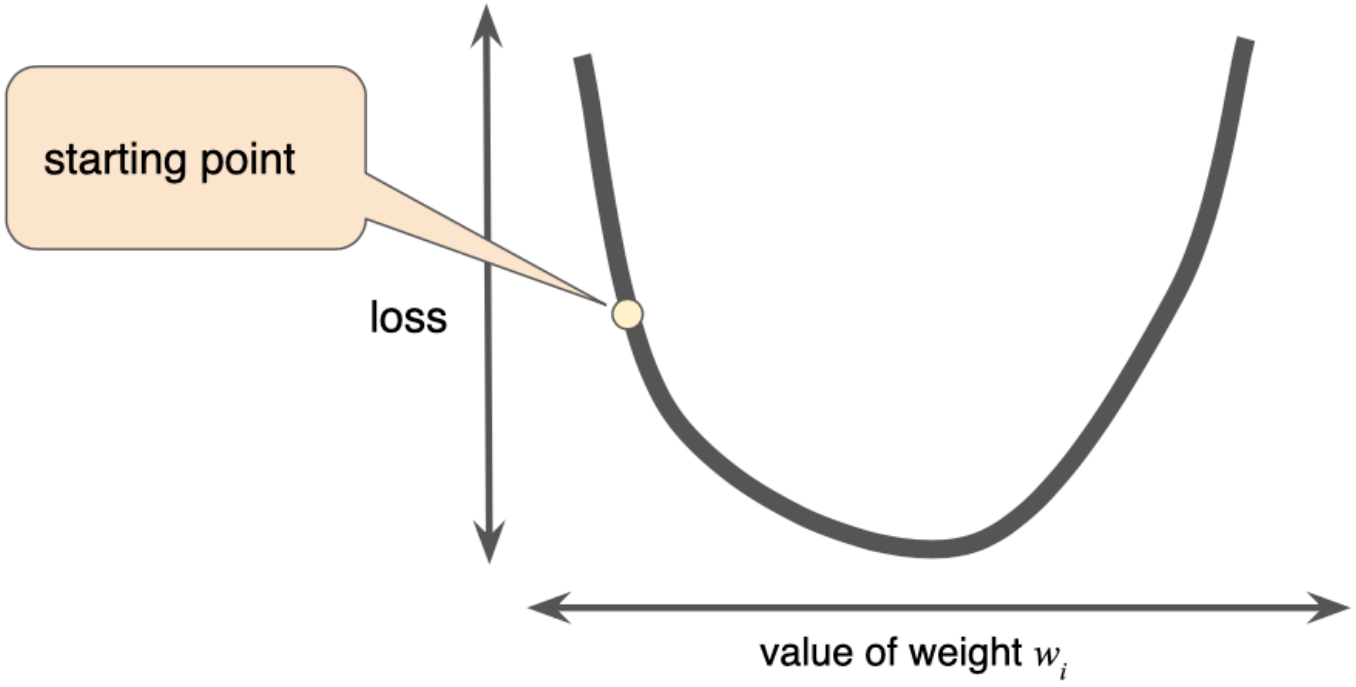
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

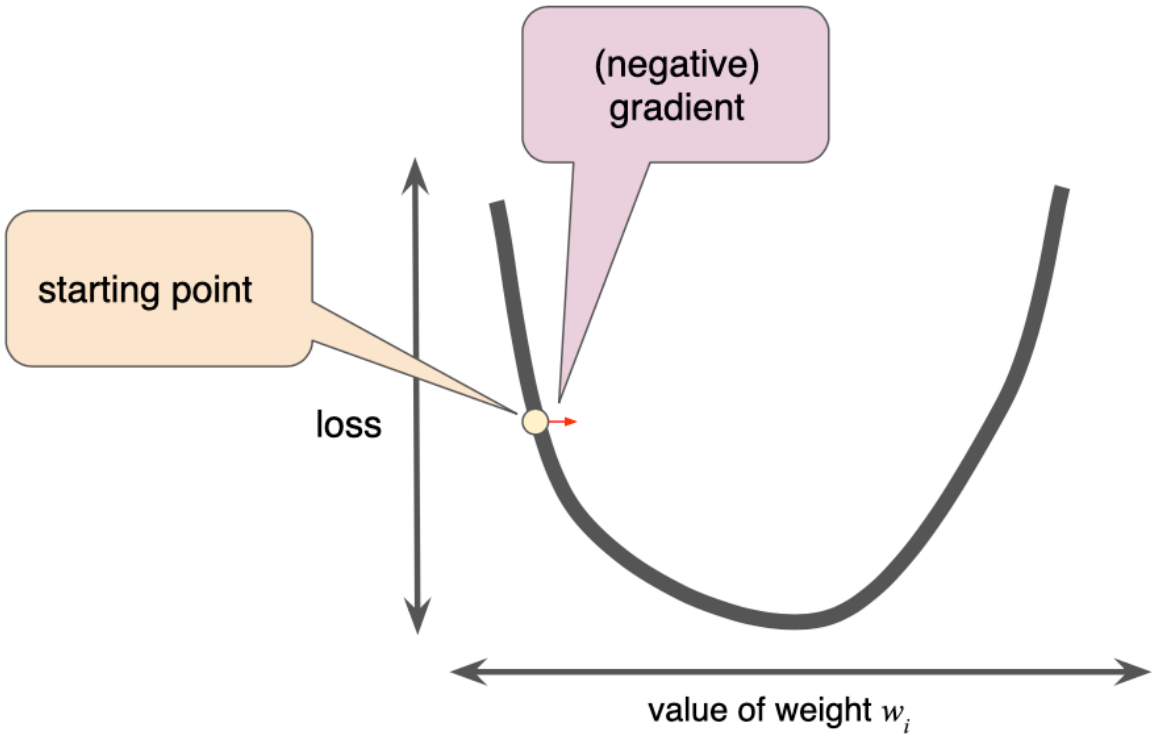


https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent

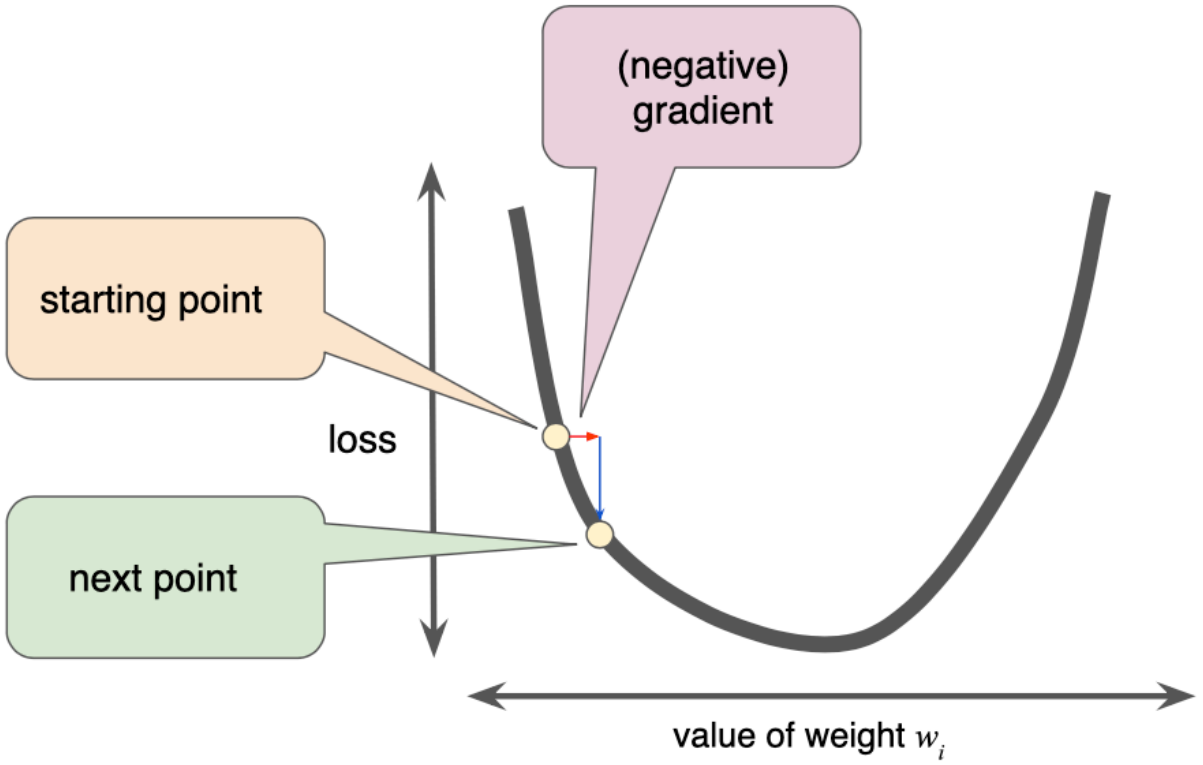
Gradient-based Optimization



Gradient-based Optimization



Gradient-based Optimization



Training as Optimization

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- we repeatedly compute an estimate of the loss over the training set
- we compute the gradients of the parameters with respect to the loss estimate
- we move the parameter values in the opposite directions of the gradient

Online SGD Training Algorithm

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Algorithm 2.1 Online stochastic gradient descent training.

Input:

- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$
3: Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
4: $\hat{\boldsymbol{g}} \leftarrow$ gradients of $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: **return** Θ

Online SGD Training Algorithm

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Algorithm 2.1 Online stochastic gradient descent training.

Input:

- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a training example $\boldsymbol{x_i}, \boldsymbol{y_i}$
3: Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
4: $\hat{\boldsymbol{g}} \leftarrow$ gradients of $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
6: **return** Θ

Learning rate

Minibatch SGD Training Algorithm

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:

- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a minibatch of m examples $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_m}, \boldsymbol{y_m})\}$
3: $\hat{\boldsymbol{g}} \leftarrow 0$
4: **for** $i = 1$ to m **do**
5: Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
6: $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ gradients of $\frac{1}{m} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
7: $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
8: **return** Θ

While using mini batch SGD?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

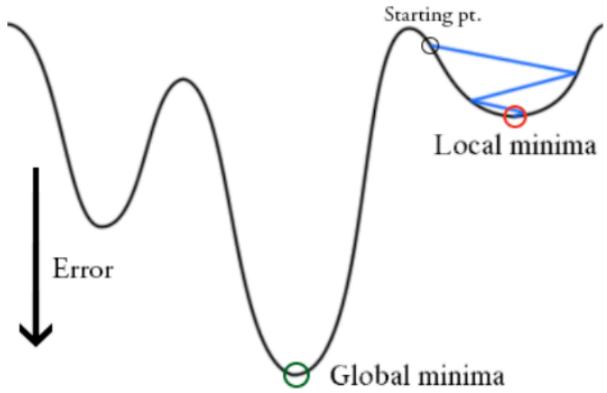
- it's not expensive: while computing loss function gradient on all training set can be computationally expensive
- it converges faster to a good solution than full-batch learning, in which we use all training set to compute gradient
- smaller mini-batch sizes lead often to better solutions (generalize better)

Risks of using SGD

UNIVERSITÄT DUISBURG ESSEN

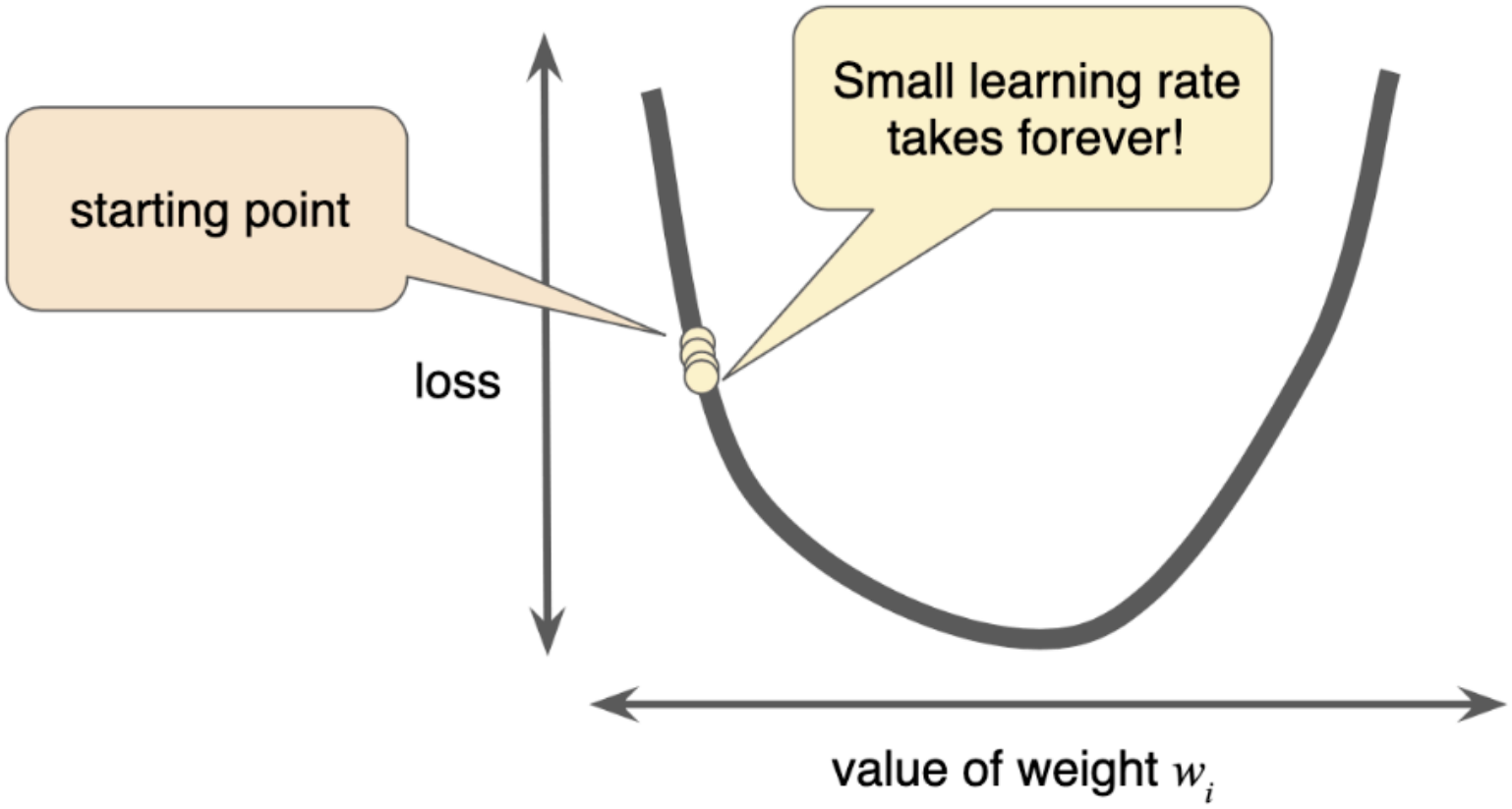
Offen im Denken

- gradient descent does not always lead to best solutions

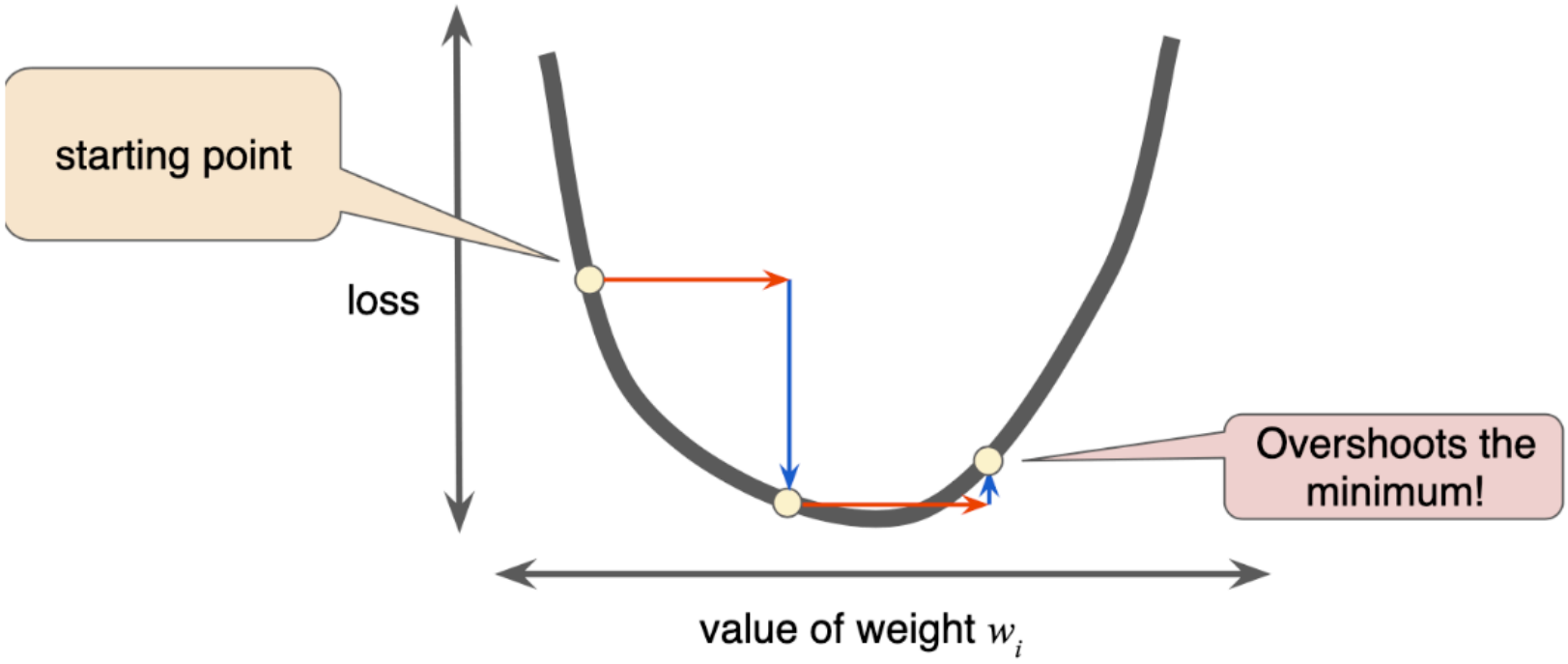


- why?
- SGD is sensitive to the learning rate and initial parameter values (starting point)

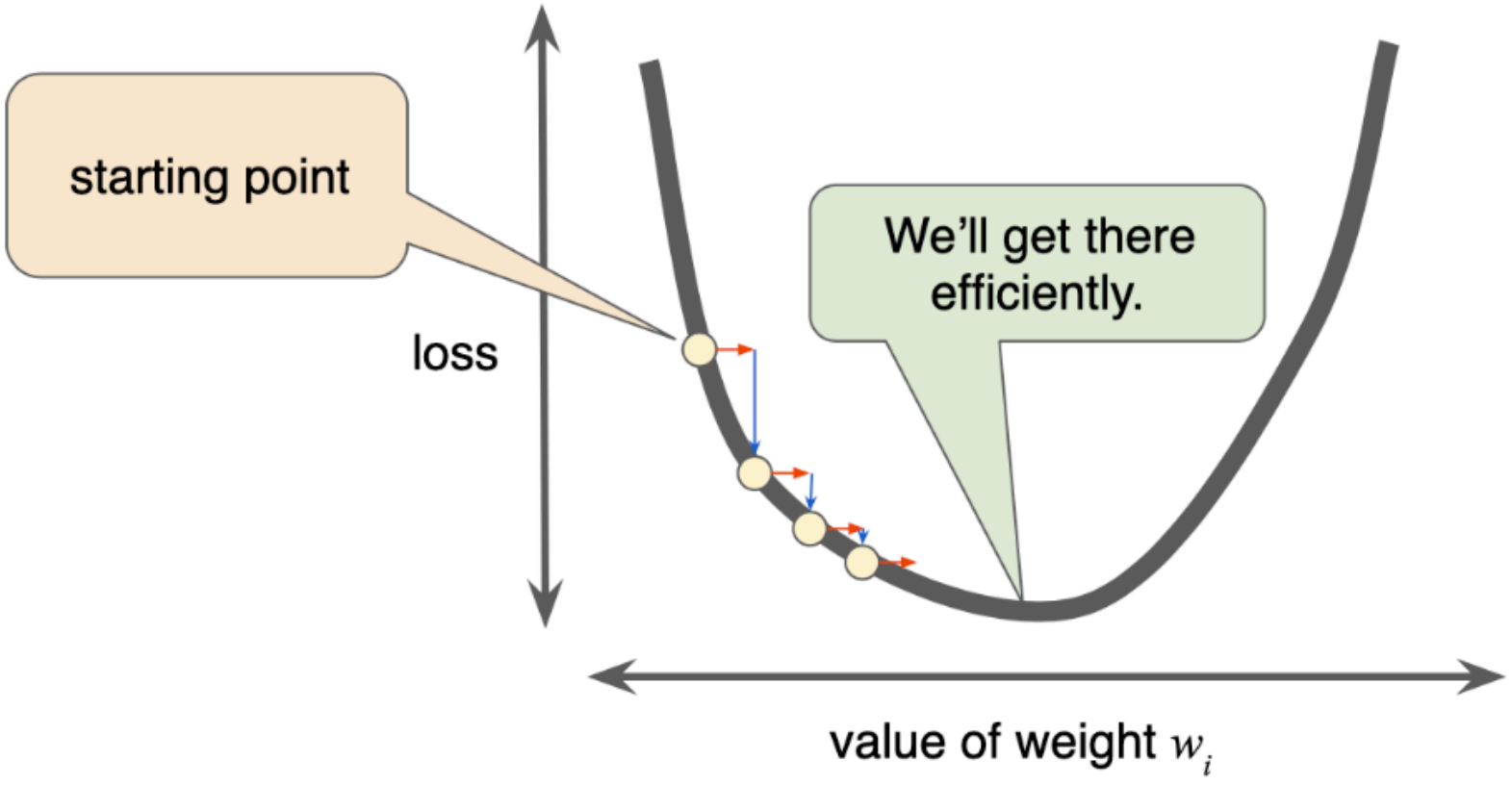
Small Learning Rate



Large Learning Rate



Adaptive Learning Rate



Common Adaptive SGD

- Use adaptive learning rate algorithms
  - AdaGrad [Duchi et al., 2011],
  - AdaDelta [Zeiler, 2012],
  - RMSProp [Tieleman and Hinton, 2012],
  - Adam [Kingma and Ba, 2014]

Common loss function

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- x is the input, f is our model (network structure), θ are the parameters that need to be optimized (weights), t is the true output value in the training data T.

- square loss for each predicted y:

$$\ell(t, y) = (t - y)^2$$

- If y and t are vectors → multi-dimensional square loss :

$$\ell(t, y) = \left|\left|t - y\right|\right|^2$$

Common loss function

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Cross-entropy loss

- $\ell(\mathbf{t}, \mathbf{y}) = -\sum_{j} \mathbf{t}_j \log(\mathbf{y}_j)$

Example: $\mathbf{t} = (0, 1, 0, 0),\ \mathbf{y} = (0.25, 0.3, 0.4, 0.05)$

Square loss:

- $0.25^2 + 0.7^2 + 0.4^2 + 0.05^2$

Cross-entropy loss:

- $-\log(0.3)$

How do we initialize the weights?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- All zero → bad idea
- Randomized → okay, but hard to optimize

- Low random values: Problem of vanishing gradients
  During training, each of the weights receives an update proportional to the partial derivative of the error function. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing the weight from changing its value.

- High random values: Problem of exploding gradients

How do we initialize the weights?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

- All zero → bad idea
- Randomized → okay, but hard to optimize
- Current state-of-the-art: Glorot initialization (or "Xavier" initialization)
- $$W_{ij} = U\left[-\frac{1}{\sqrt{n}}, +\frac{1}{\sqrt{n}}\right]$$
  - U is the uniform distribution
  - n is the size of the previous layer (the number of columns of W)
- Subject to a lot of experimentation

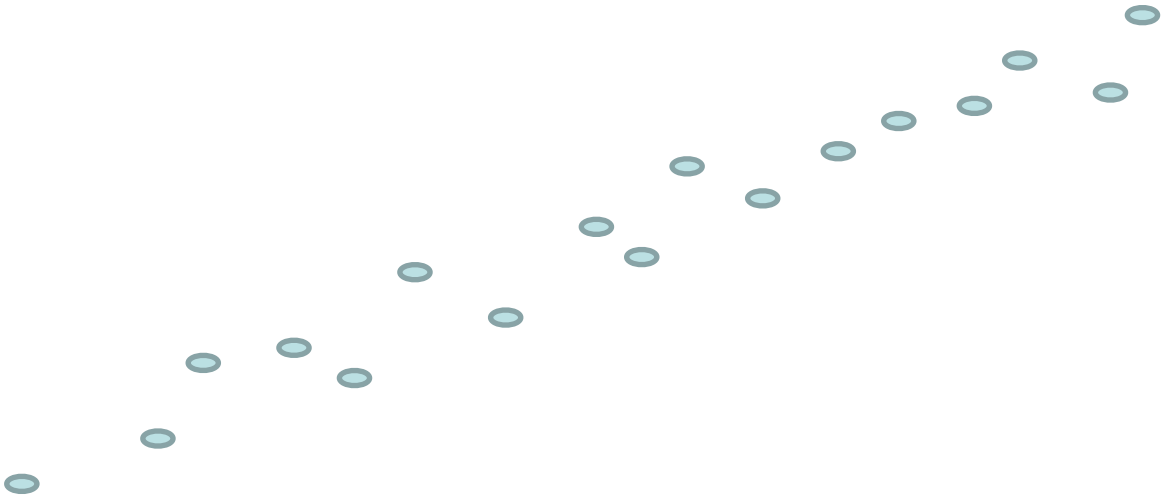
- Xavier Glorot & YoshuaBengio. (2010): *Understanding the difficulty of training deep feed forward neural networks* http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf

Regularization

Why regularization?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

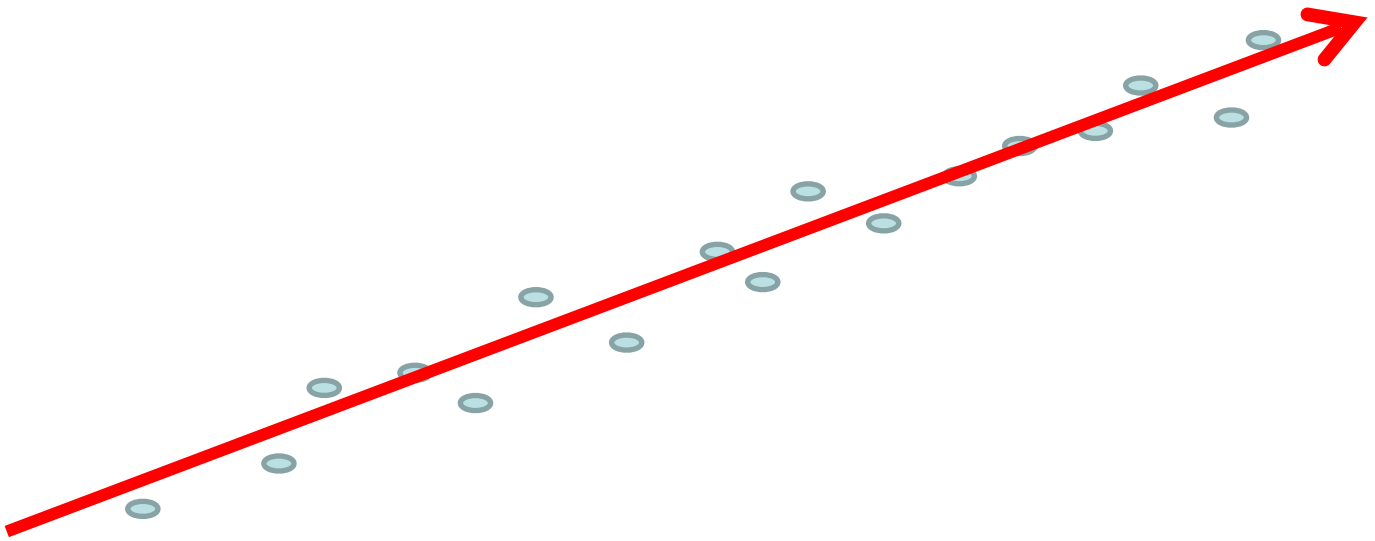


Why regularization?

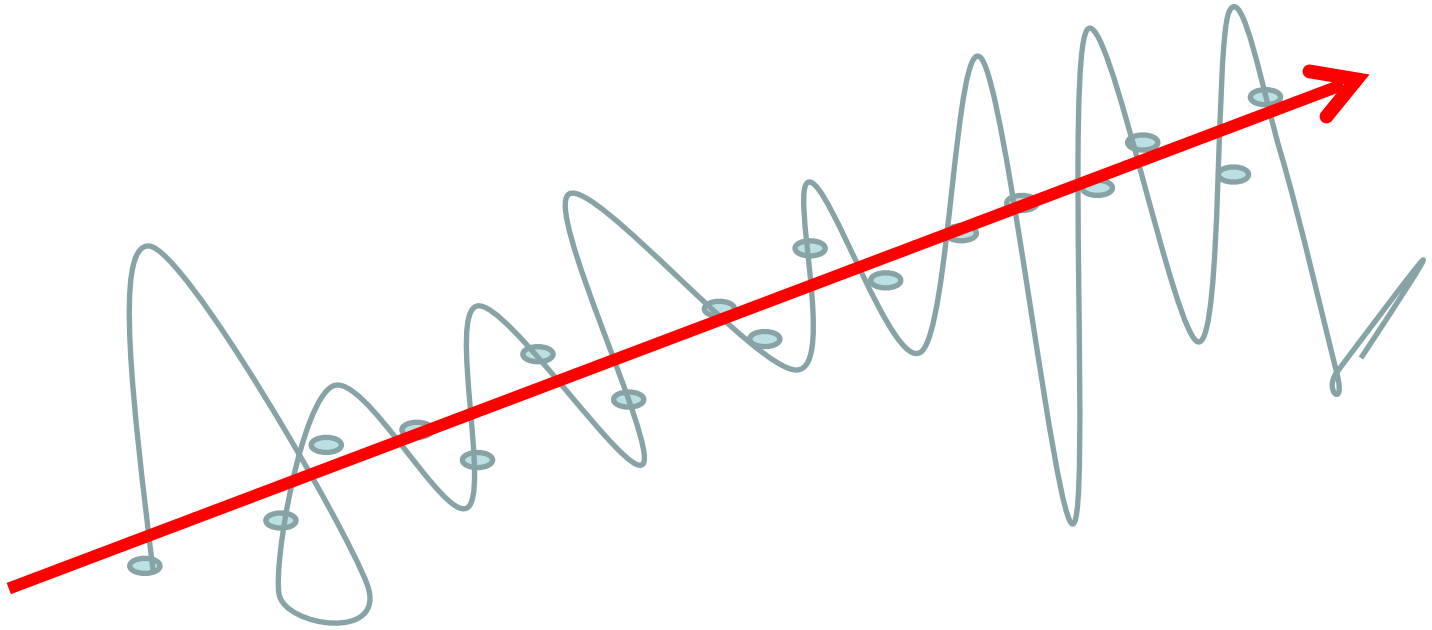
UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Train a model to fit the data.



Why regularization?



How can we apply regularization?

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

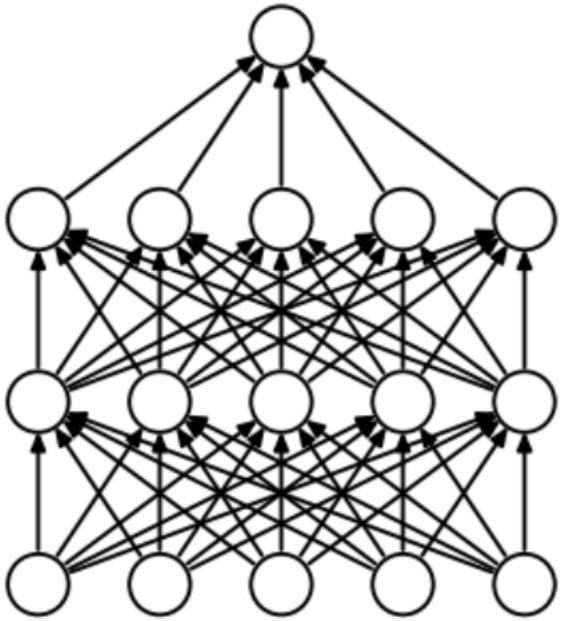
- Our goal was to minimize the error function by optimizing the weights.
- We want to avoid that the weights get too fitted to the data.
  → add a regularization term to the objective function

- minimize: $f(w) + \lambda \mathbf{R}(\mathbf{w})$
- λ is our regularization parameter
- R is our regularization function
- L2 regularization: $\mathbf{R}(\mathbf{w}) = ||\mathbf{w}||^2 = \sum_i \mathbf{w}_i^2$
- L1 regularization: $\mathbf{R}(\mathbf{w}) = |\mathbf{w}| = \sum_i |\mathbf{w}_i|$

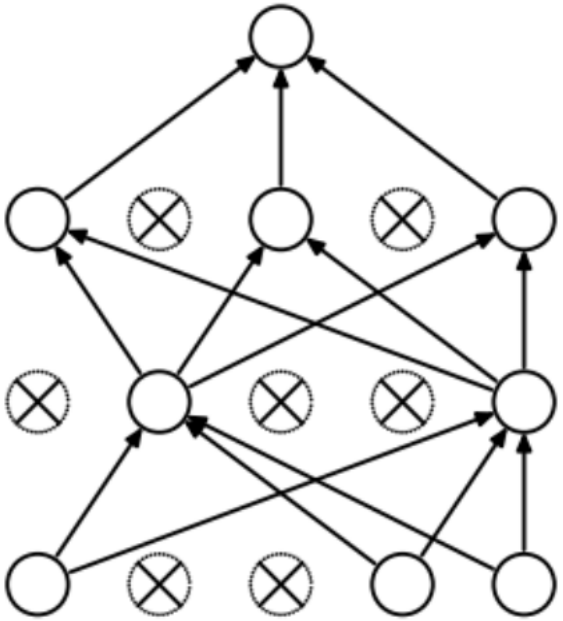
- If we want to minimize the whole thing, we need to minimize the error and the regularization function → keep the weights small

Dropout for regularization

Dropout



(a) Standard Neural Net



(b) After applying dropout.

Reading

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Mandatory:

- Ian Goodfellow, Yoshua Bengio, Aaron Courville:
- Deep Learning, *Introduction*, p.1-26
  http://www.deeplearningbook.org/contents/intro.html

Optional:

- Tariq Rashid: *Make your own neural network*
  https://www.amazon.com/Make-Your-Own-Neural-Network-ebook/dp/B01EER4Z4G/
- Very gentle step-by-step introduction

Thank You