

Course

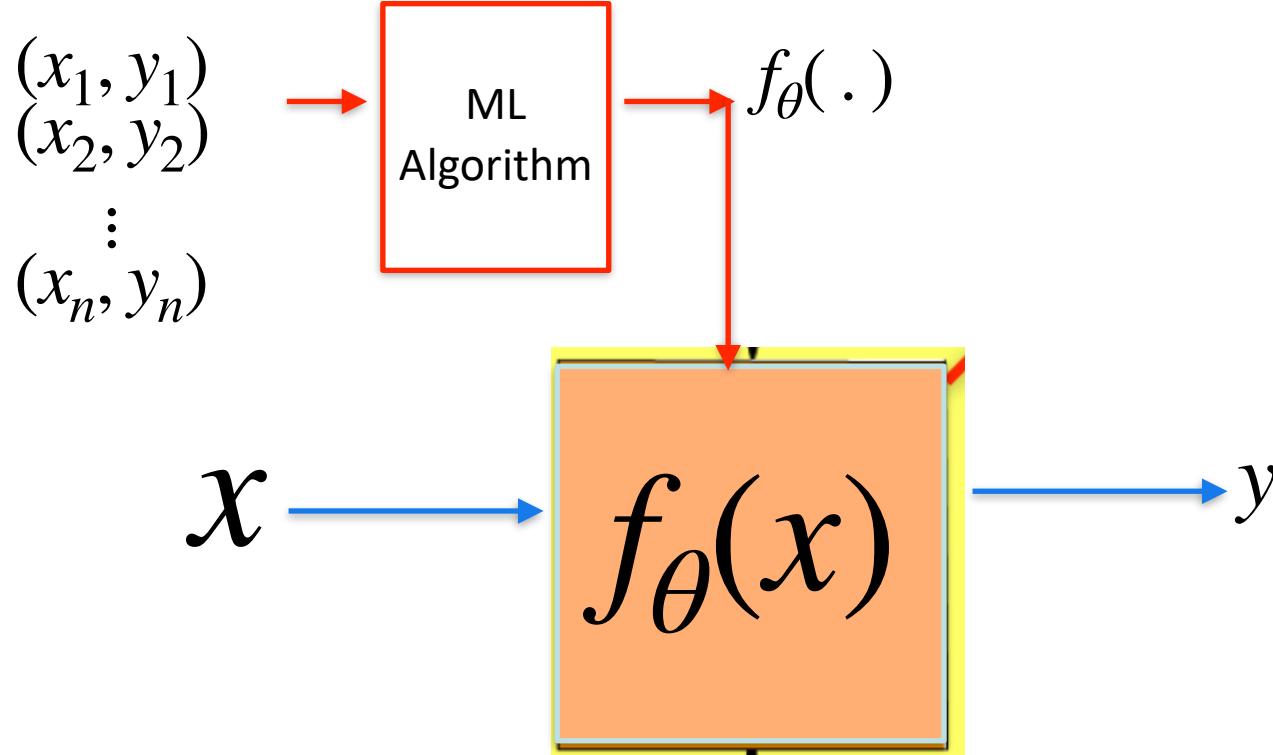
Foundations of Artificial Intelligence

Lecture 10 – Deep Learning

Dr. Mohsen Mesar

Universität Duisburg-Essen

Recall



Recall

- 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?

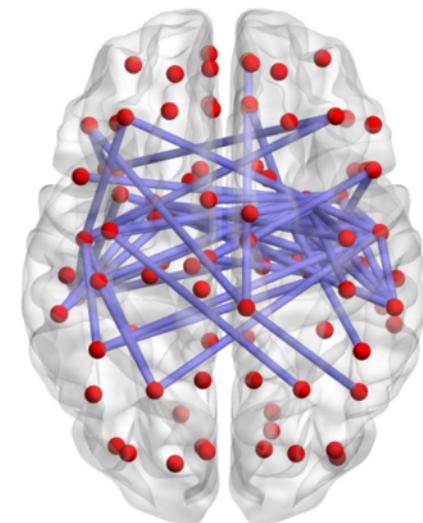
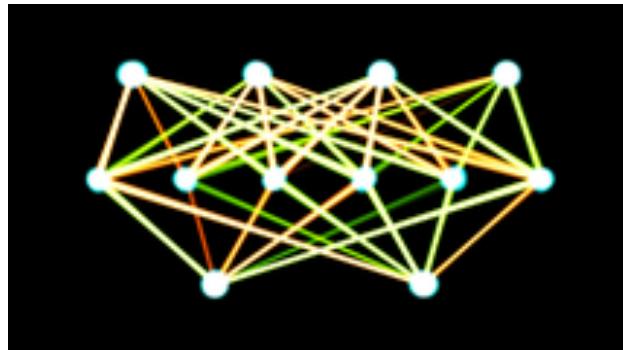


In this lecture, you learn ...

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

Deep Learning (DL)

- 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

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

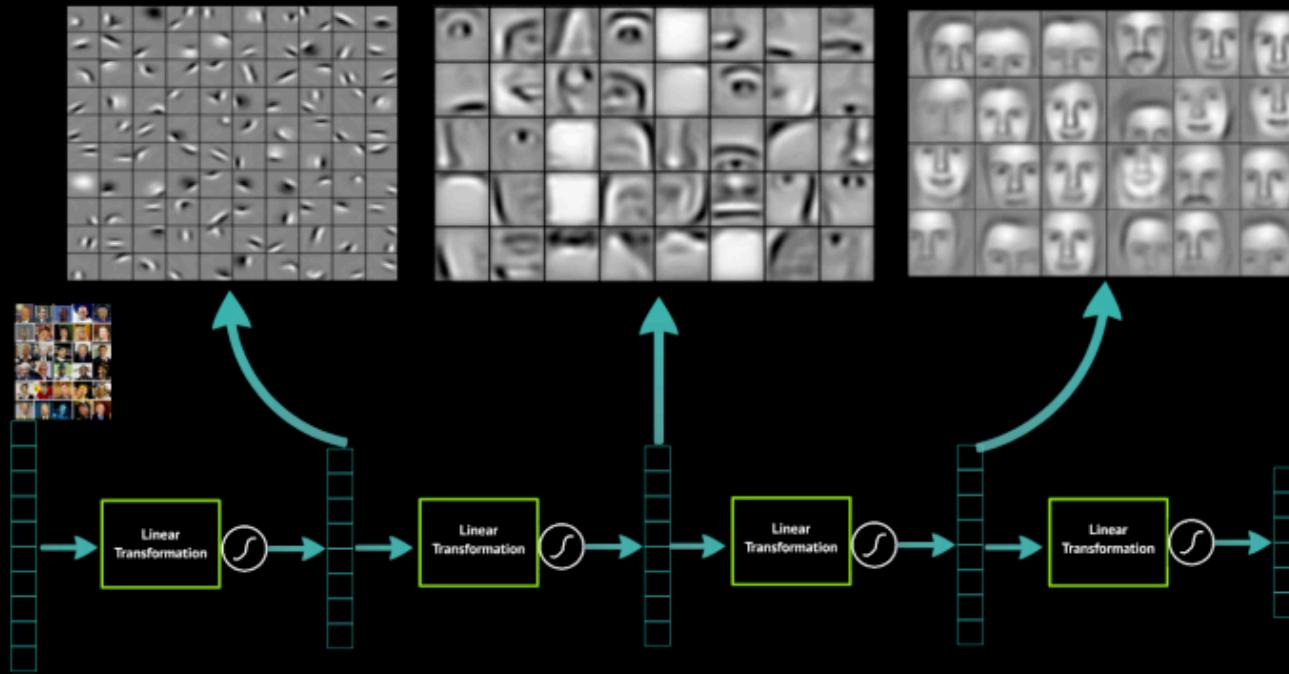


Classical vs. Deep Learning

- 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

Deep Learning learns layers of features

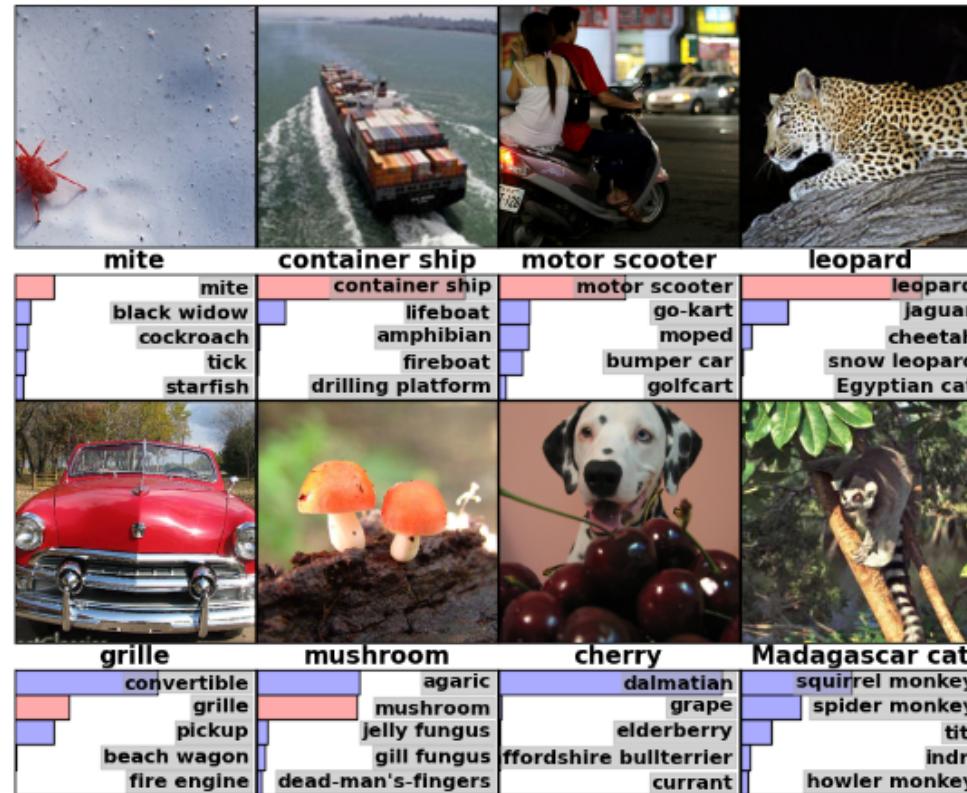


Source: <https://becominghuman.ai/what-is-deep-learning-and-its-advantages-16b74bc541a1> and Lee et al. (ICML 2009)

DL Example: Object Recognition

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

Figure 4:



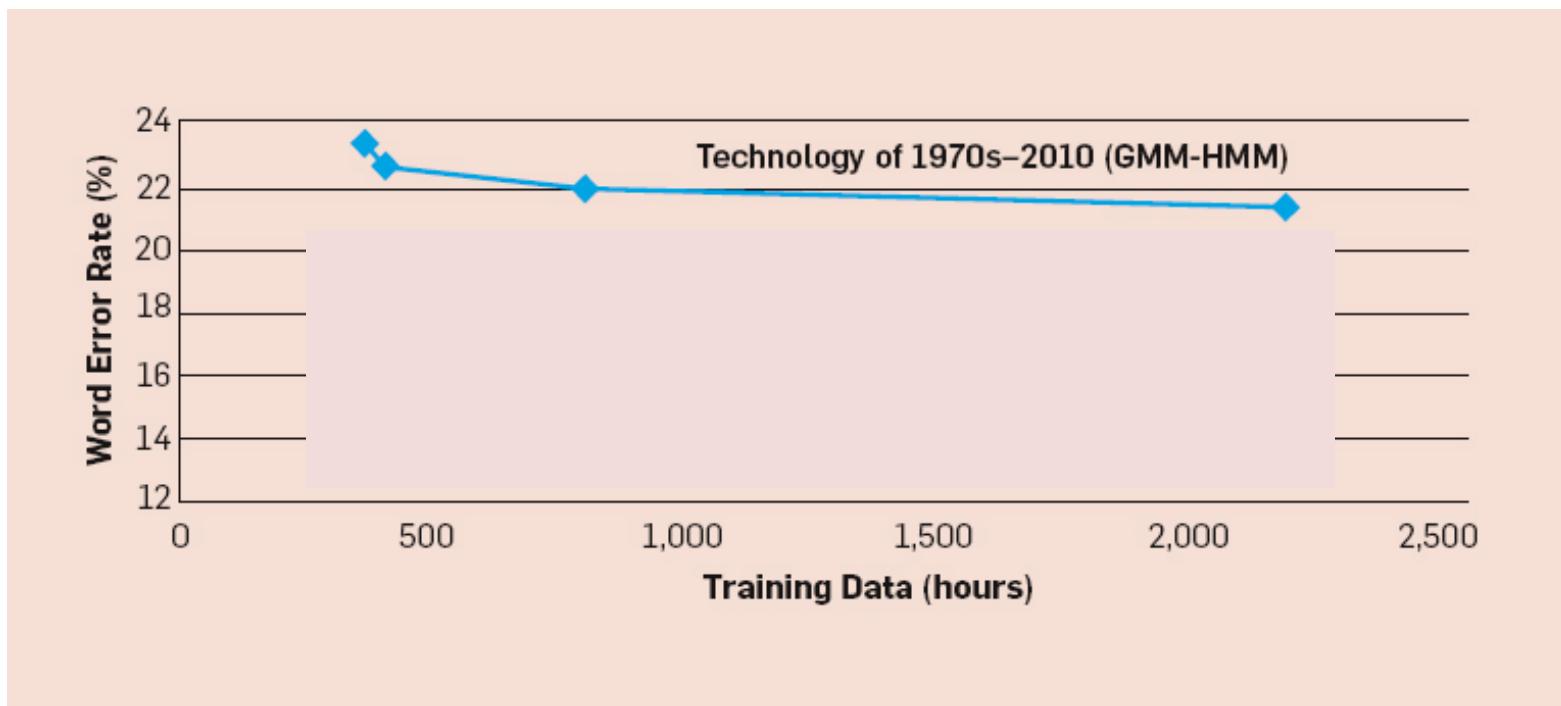
DL Example: Image Description

GoogLeNet 2014: <http://googleresearch.blogspot.de/2014/09/>
Building deeper understanding of images

Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
 A person riding a motorcycle on a dirt road.	 Two dogs play in the grass.	 A skateboarder does a trick on a ramp.	 A dog is jumping to catch a frisbee.
 A group of young people playing a game of frisbee.	 Two hockey players are fighting over the puck.	 A little girl in a pink hat is blowing bubbles.	 A refrigerator filled with lots of food and drinks.
 A herd of elephants walking across a dry grass field.	 A close up of a cat laying on a couch.	 A red motorcycle parked on the side of the road.	 A yellow school bus parked in a parking lot.

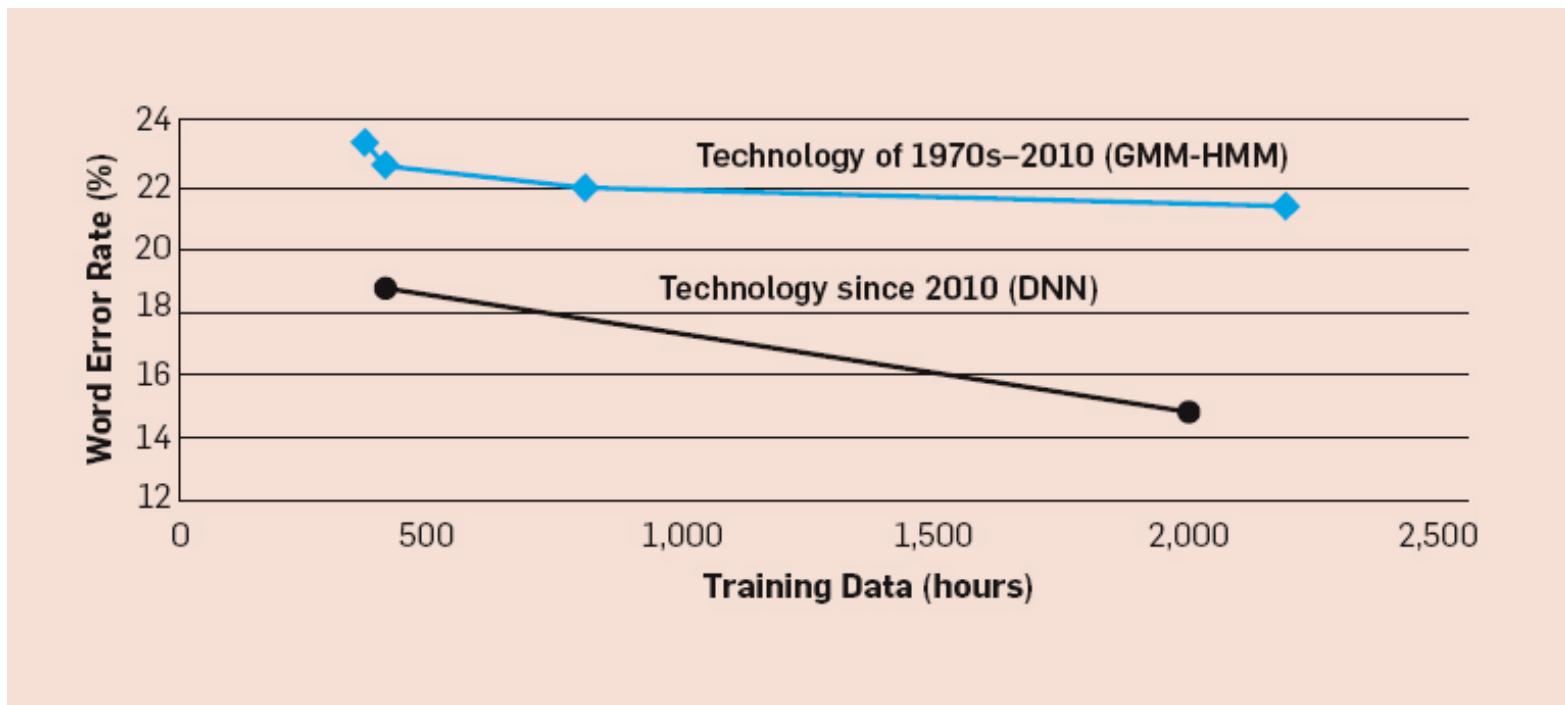
Speech Recognition

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



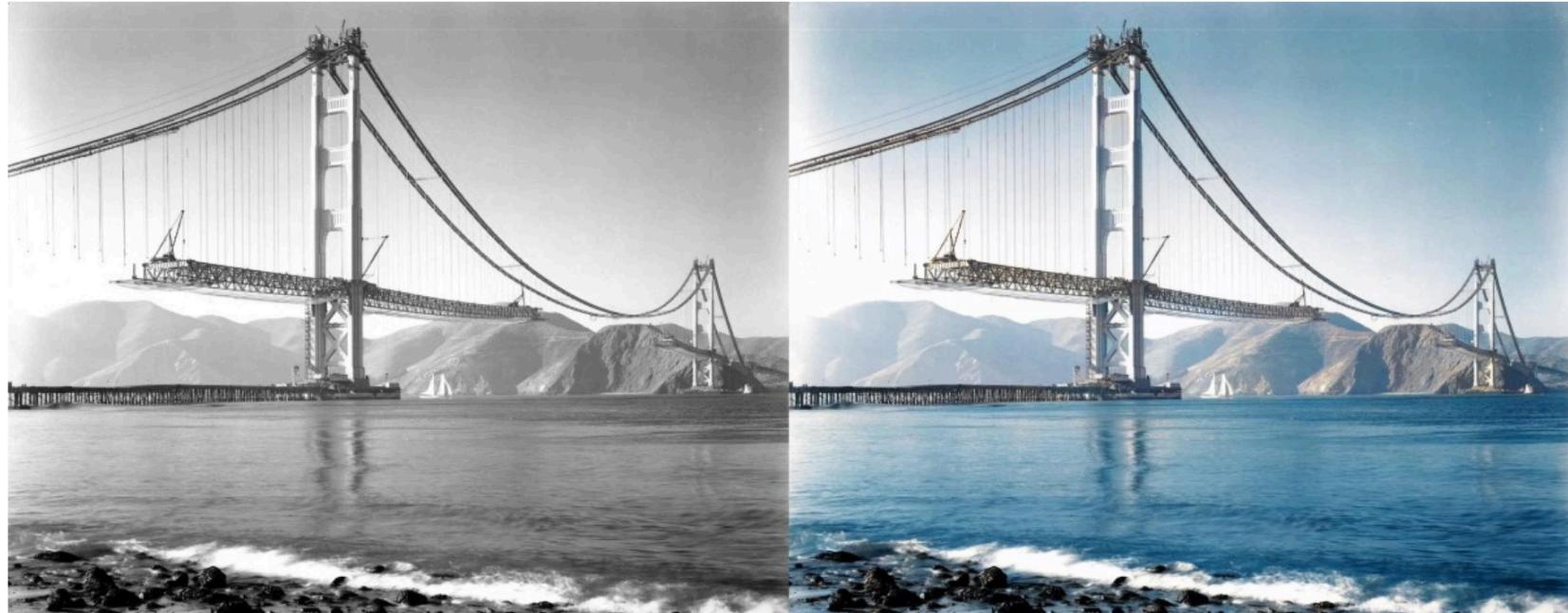
What happened in 2010?

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



Cool Projects

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



Cool Projects

- <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?

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, ...

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

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”)

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”

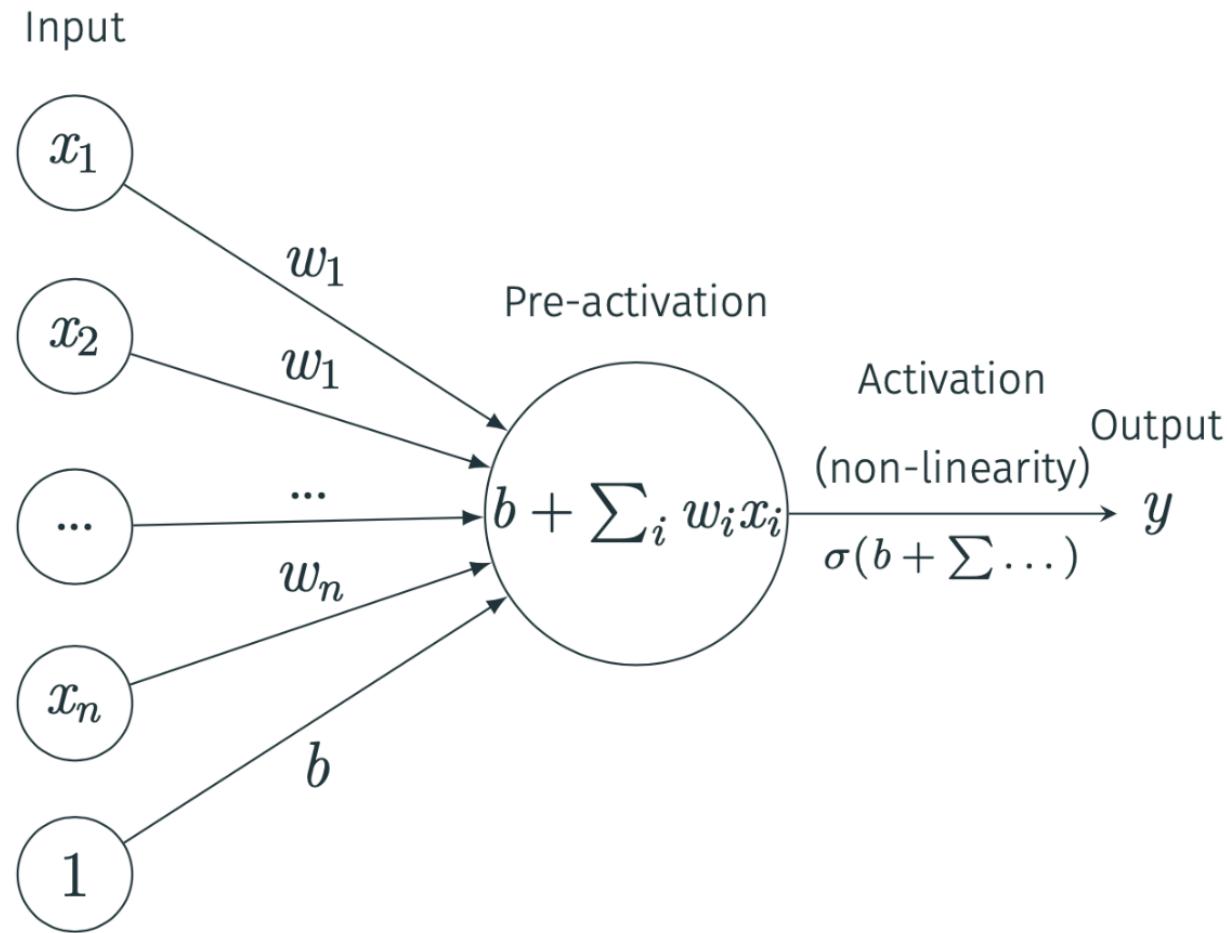
- Hinton and Salakhutdinov (2006): one can actually train deep nets
- Since 2013: Veritable hype in NLP, mostly due to word embeddings

Deep Learning

- 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

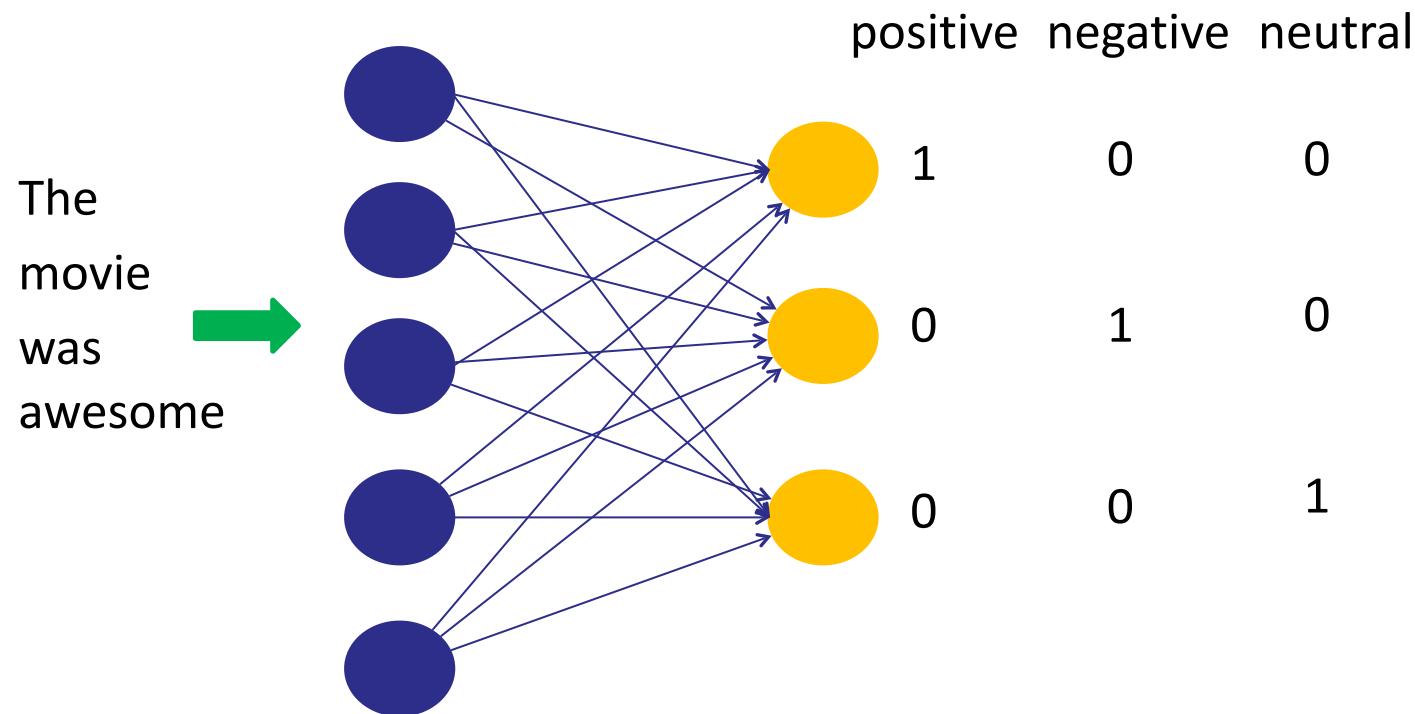


Formally

- 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: $\sigma(\mathbf{w} \cdot \mathbf{x})$

- \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



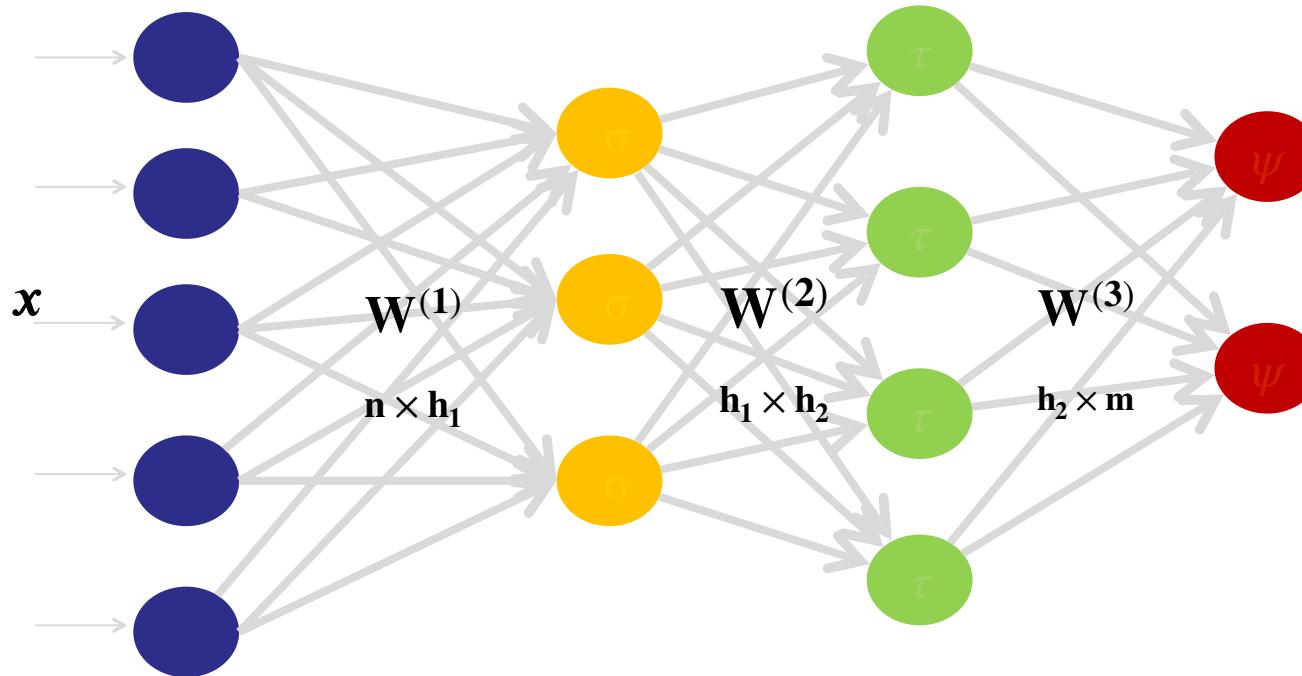
More complex neural network

Multiple hidden layers (multilayer perceptron)

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

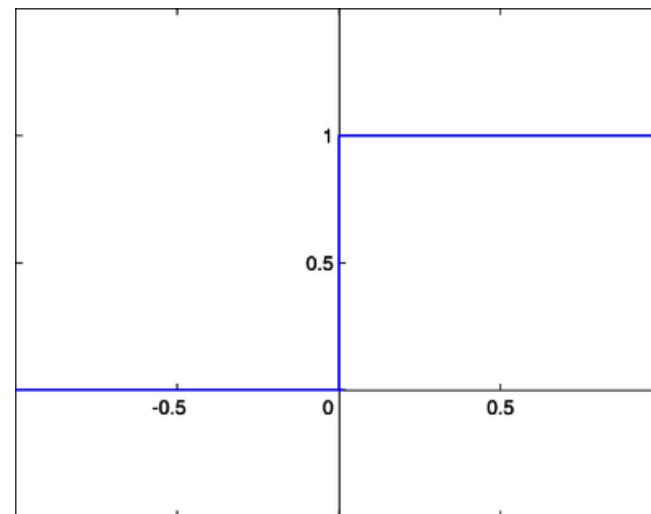
→ forward propagation

Many more parameters and choices.



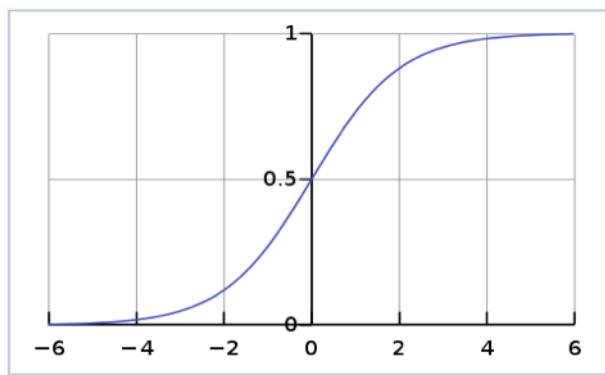
Activation function

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

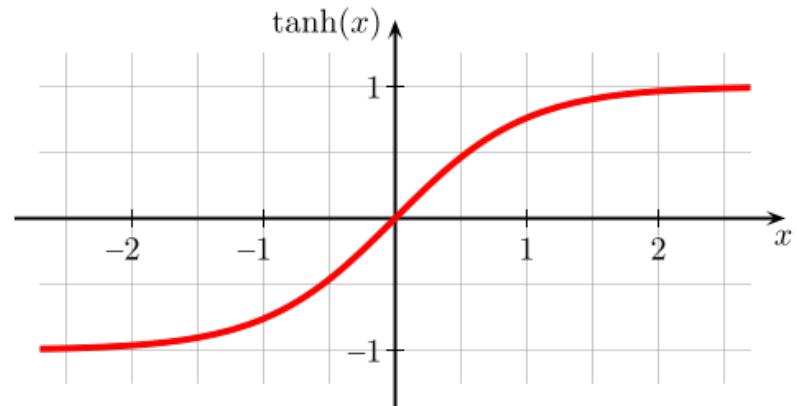


Sigmoid & Tanh

- Sigmoid function (scale: 0 to 1)
- Hyperbolic tangent (scale: -1 to 1)



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



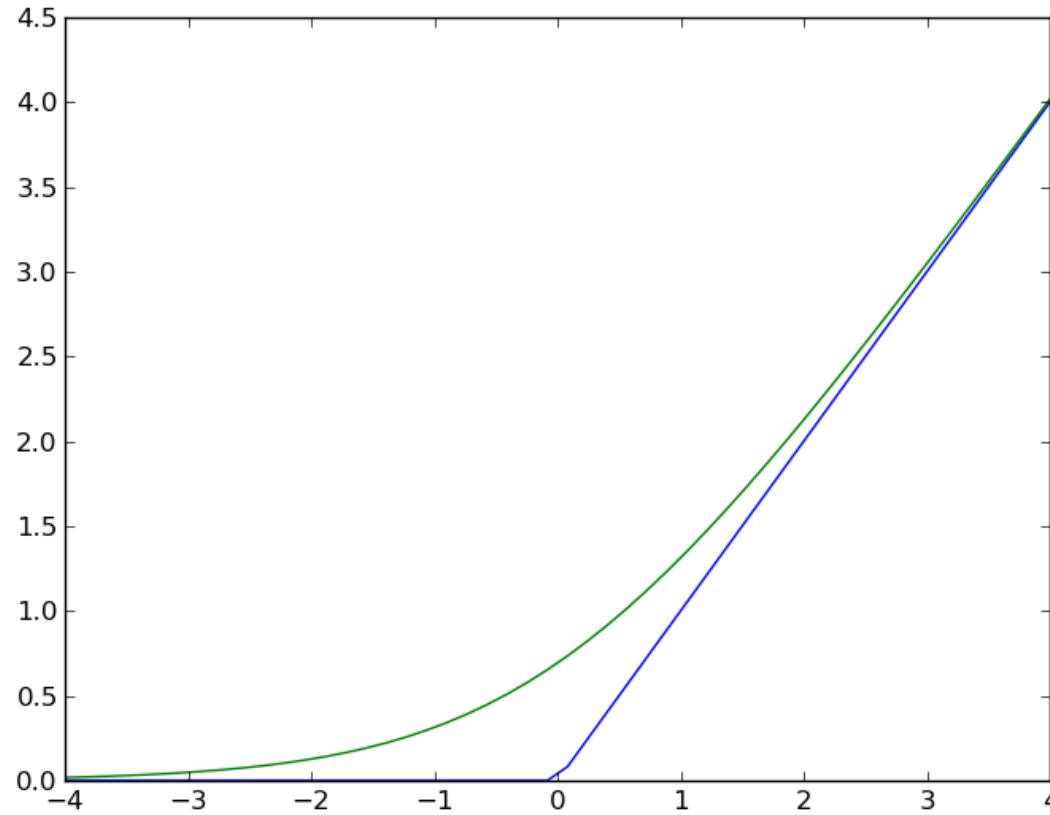
$$\tanh x = \frac{\sinh x}{\cosh x}$$

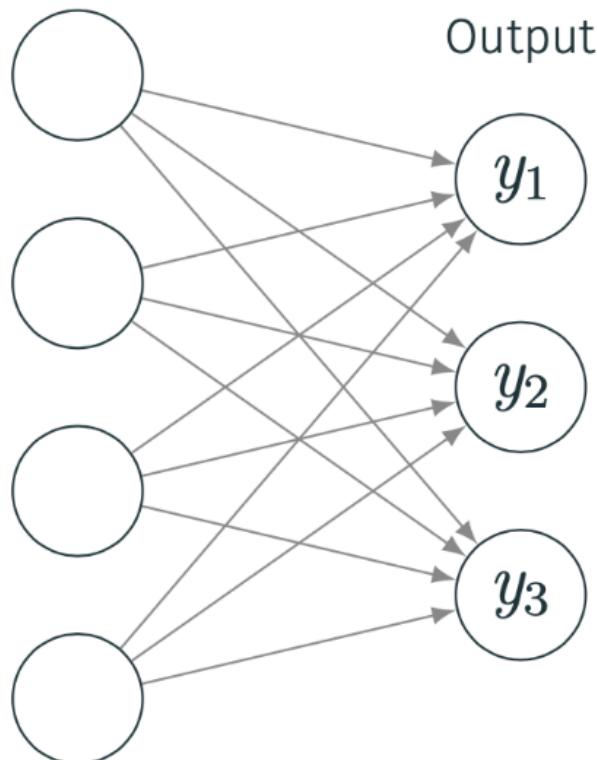
https://en.wikipedia.org/wiki/Sigmoid_function

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

Rectified linear units (ReLU)

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





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

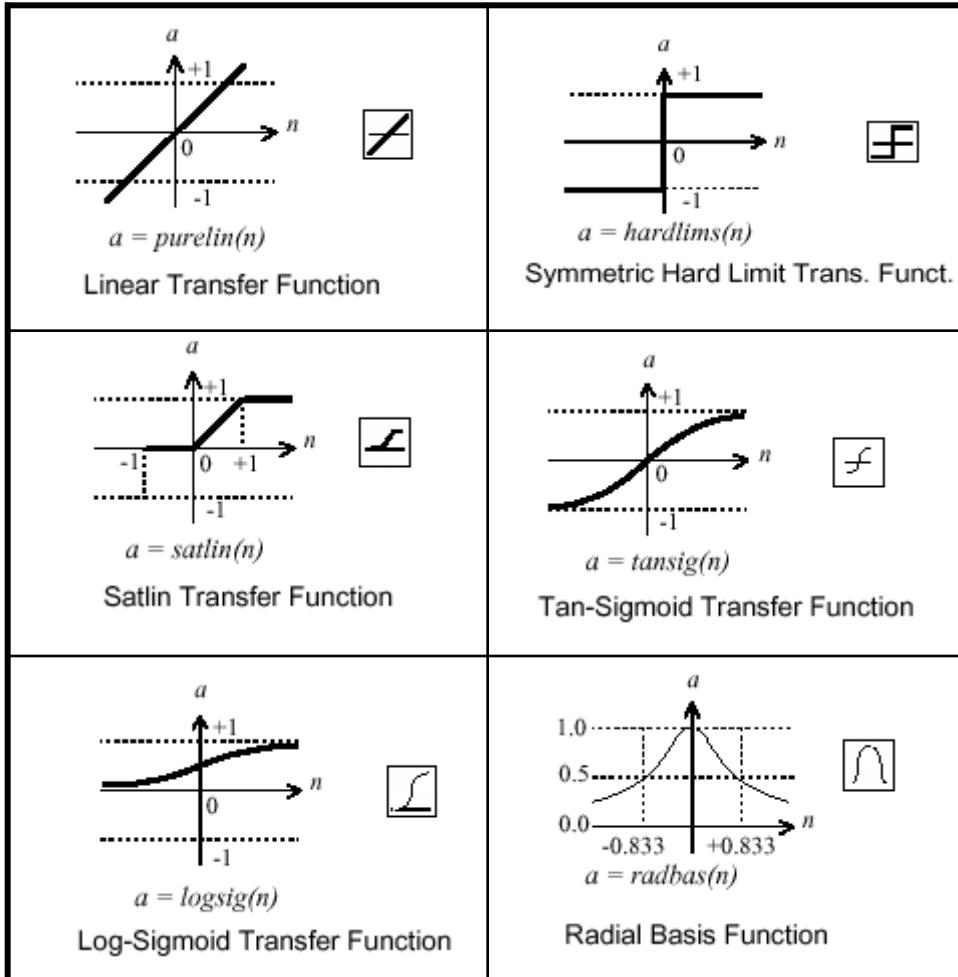
$$y_1 + y_2 + y_3 = 1; \quad y_j \geq 0$$

Let z_1 be pre-activation of output node y_1

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

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

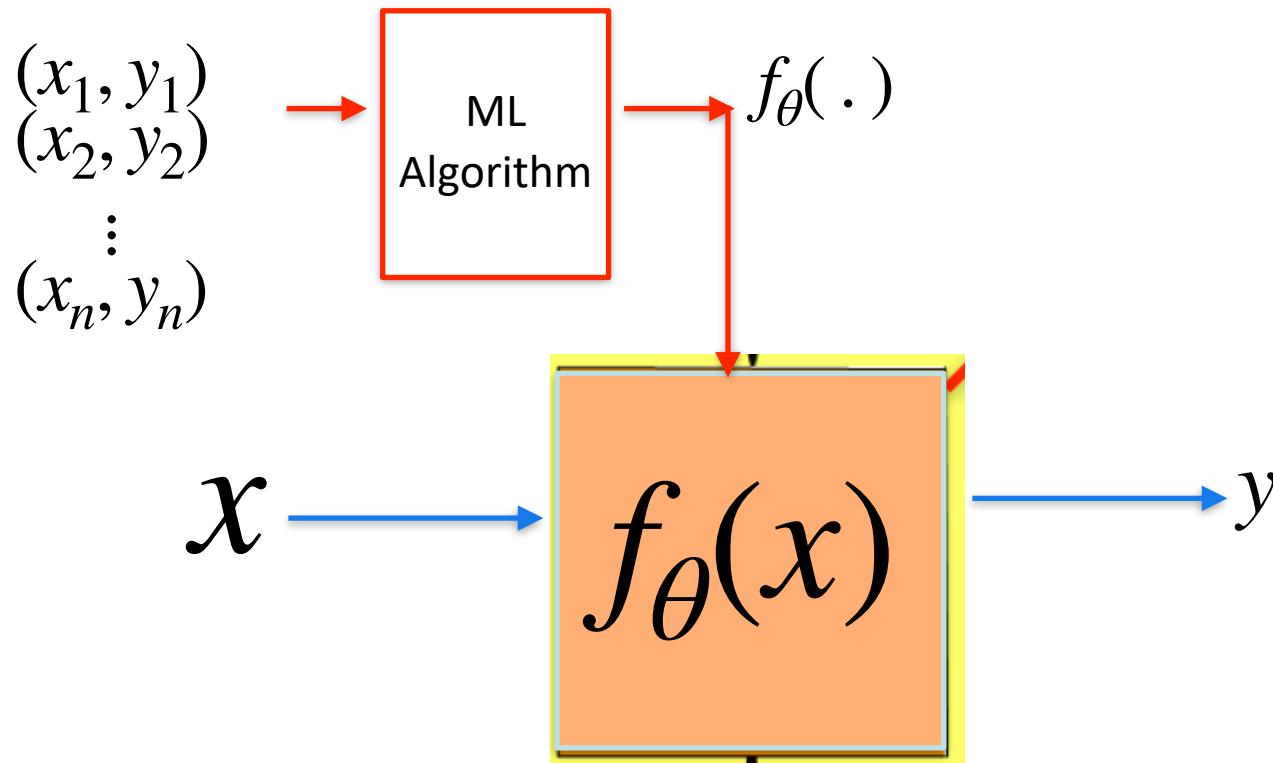
Many more...



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

Recall Learning

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



Supervised Learning (SL)

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

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

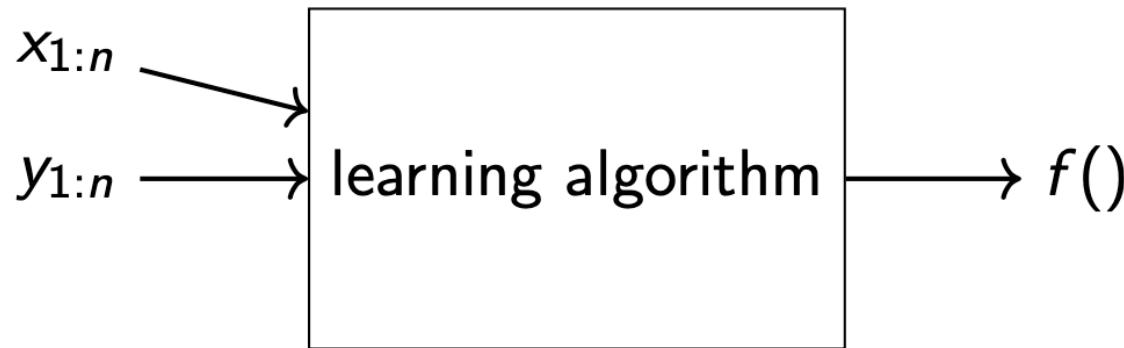
Model's predictions

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.

- ▶ the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
 - ▶ $x_{1:n} = x_1, x_2, \dots, x_n$ shows input examples
 - ▶ $y_{a:n} = y_1, y_2, \dots, y_n$ 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

- ▶ *Loss function $L(y, \hat{y})$:* It quantifies the loss suffered when predicting \hat{y} 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:

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

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

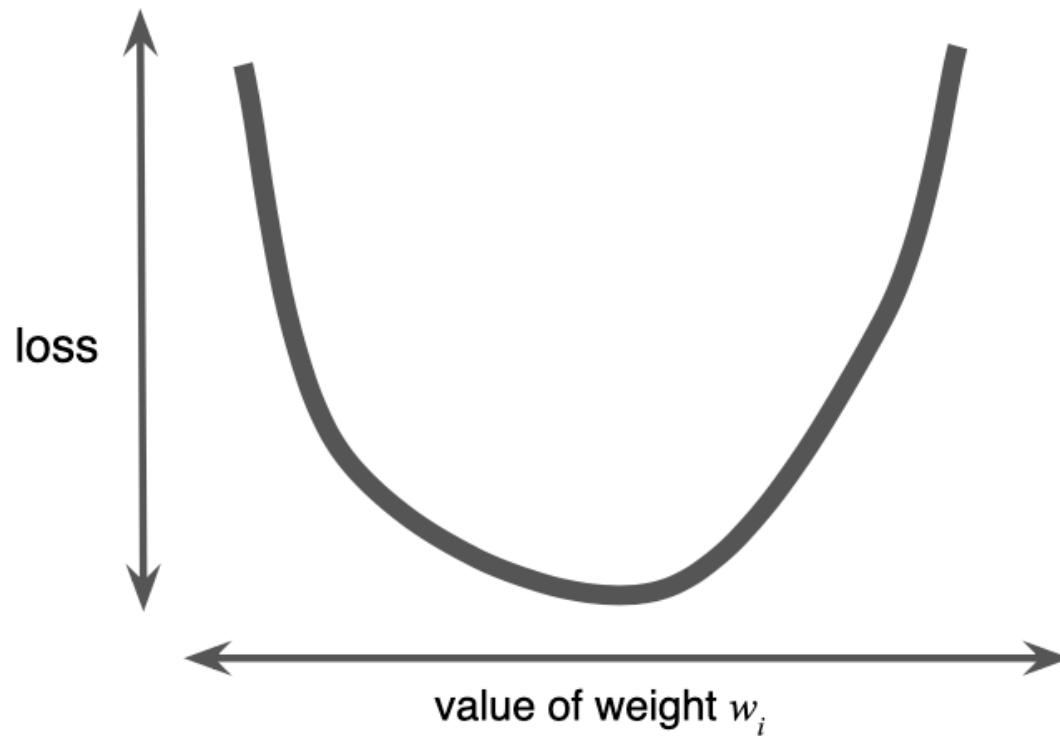
Training as Optimization

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

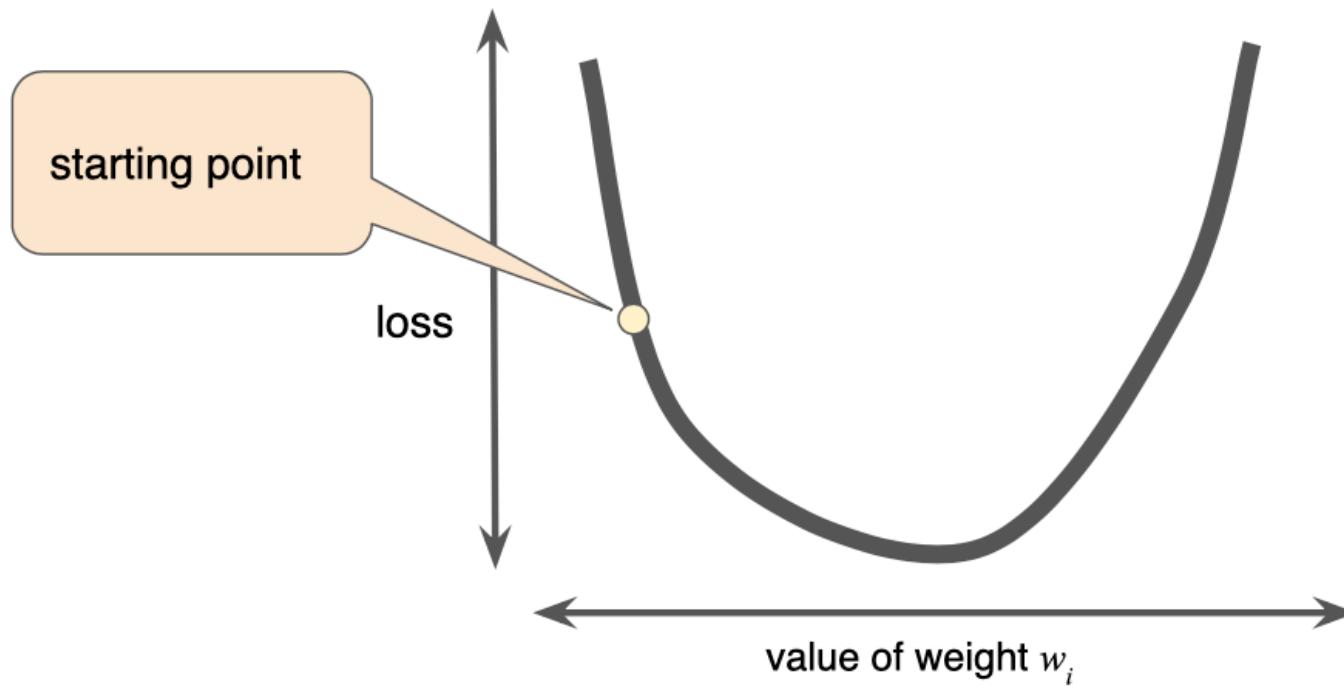
$$\hat{\Theta} = \operatorname{argmin}_{\Theta} \mathcal{L}(\Theta) = \operatorname{argmin}_{\Theta} \frac{1}{n} \sum_{i=1}^n L(\hat{y}, y_i)$$

- ▶ how to find parameter values that minimize loss?

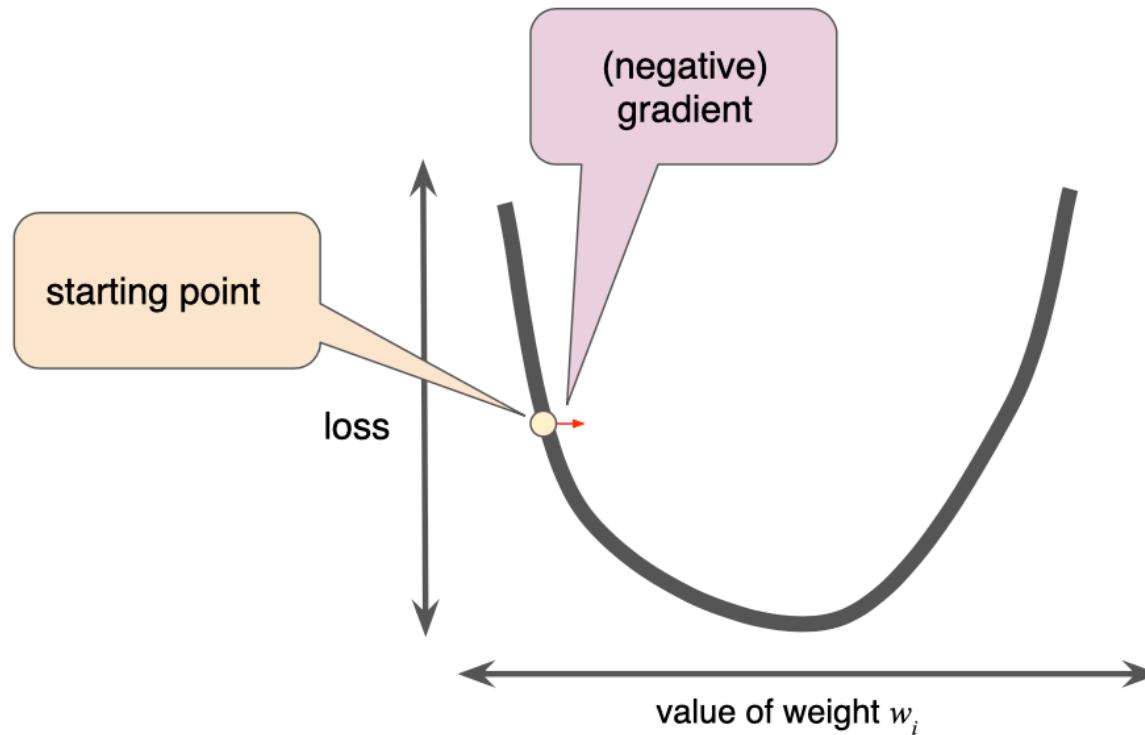
Gradient-based Optimization



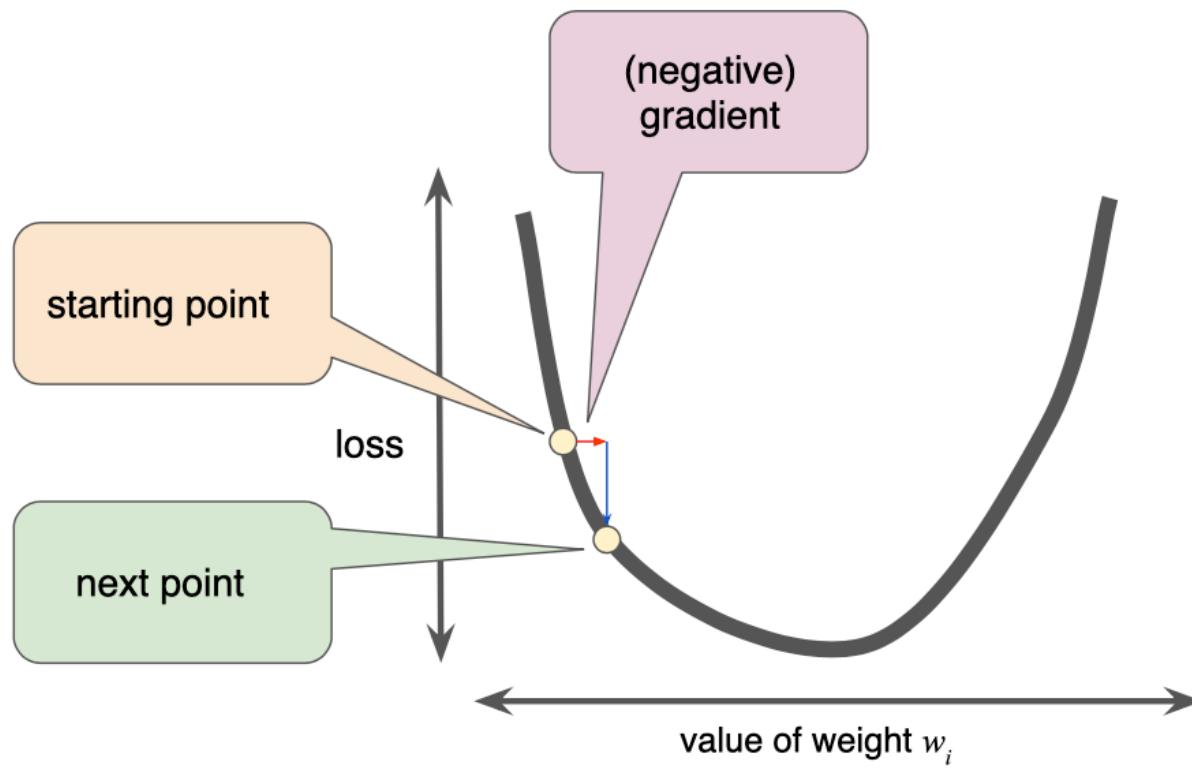
Gradient-based Optimization



Gradient-based Optimization



Gradient-based Optimization



Training as Optimization

- ▶ 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

Algorithm 2.1 Online stochastic gradient descent training.

Input:

- Function $f(\mathbf{x}; \Theta)$ parameterized with parameters Θ .
 - Training set of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n$ and desired outputs y_1, \dots, y_n .
 - Loss function L .
-

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

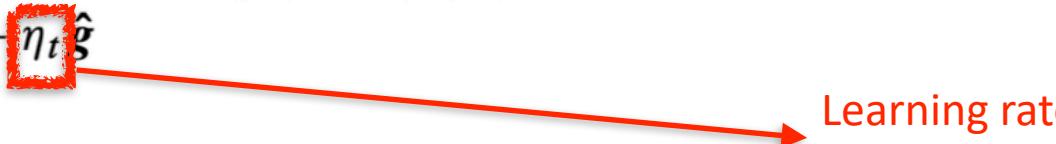
Online SGD Training Algorithm

Algorithm 2.1 Online stochastic gradient descent training.

Input:

- Function $f(\mathbf{x}; \Theta)$ parameterized with parameters Θ .
- Training set of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n$ and desired outputs y_1, \dots, y_n .
- Loss function L .

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



Minibatch SGD Training Algorithm

Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:

- Function $f(\mathbf{x}; \Theta)$ parameterized with parameters Θ .
 - Training set of inputs $\mathbf{x}_1, \dots, \mathbf{x}_n$ and desired outputs y_1, \dots, y_n .
 - Loss function L .
-

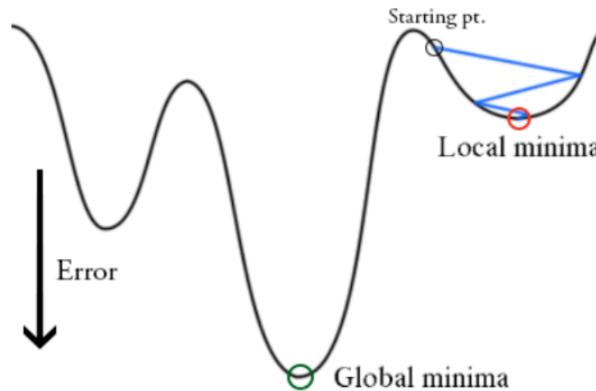
```
1: while stopping criteria not met do
2:   Sample a minibatch of  $m$  examples  $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$ 
3:    $\hat{\mathbf{g}} \leftarrow 0$ 
4:   for  $i = 1$  to  $m$  do
5:     Compute the loss  $L(f(\mathbf{x}_i; \Theta), y_i)$ 
6:      $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} + \text{gradients of } \frac{1}{m}L(f(\mathbf{x}_i; \Theta), y_i) \text{ w.r.t } \Theta$ 
7:    $\Theta \leftarrow \Theta - \eta_t \hat{\mathbf{g}}$ 
8: return  $\Theta$ 
```

While using mini batch SGD?

- ▶ 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)

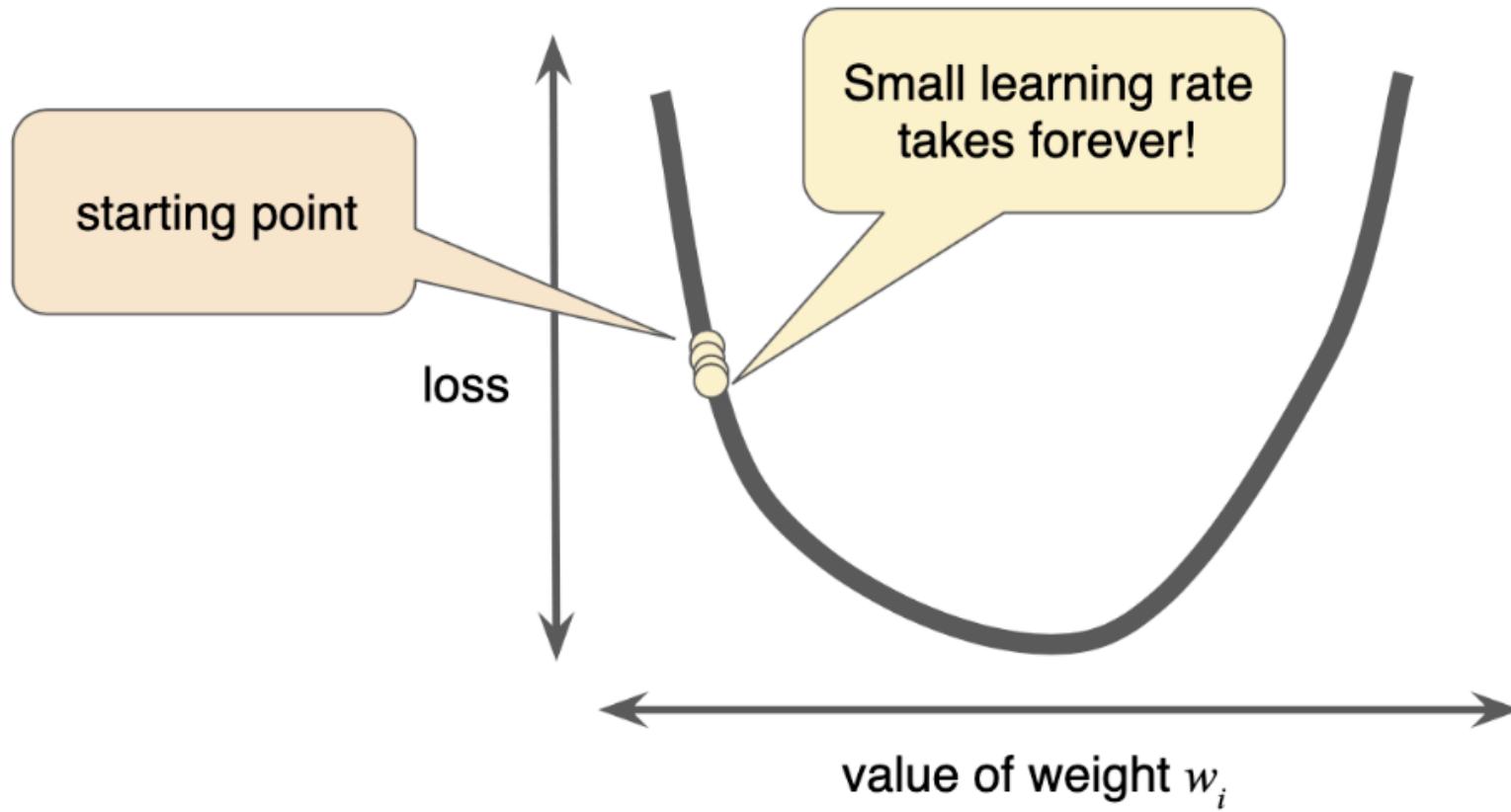
Risks of using SGD

- ▶ gradient descent does not always lead to best solutions

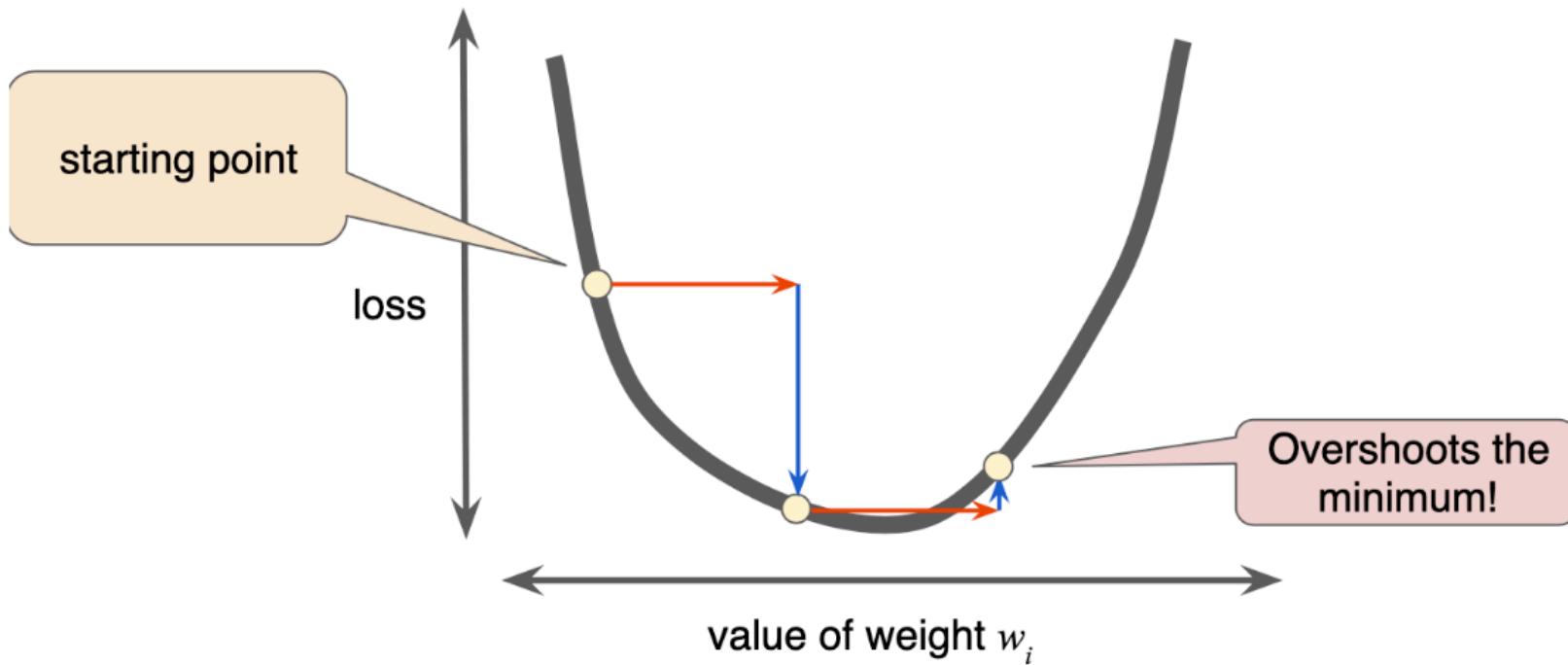


- ▶ why?
- ▶ SGD is sensitive to the learning rate and initial parameter values (starting point)

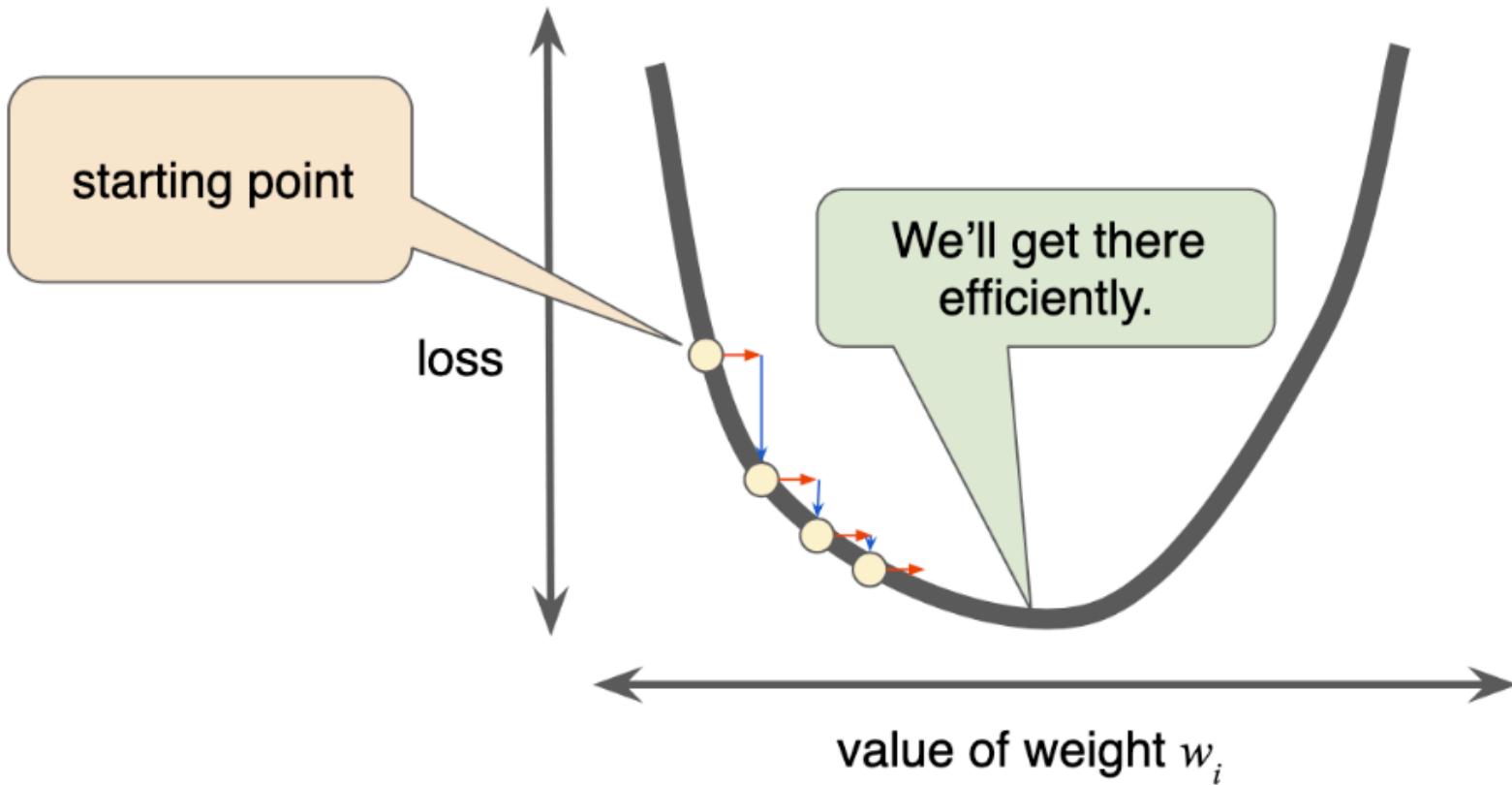
Small Learning Rate



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

- 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 \rightarrow multi-dimensional square loss :

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

Common loss function

Cross-entropy loss

- $\ell(t, y) = - \sum_j t_j \log(y_j)$

Example: $t = (0, 1, 0, 0)$, $y = (0.25, 0.3, 0.4, 0.05)$

Square loss:

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

Cross-entropy loss:

- $-\log(0.3)$

Reading

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