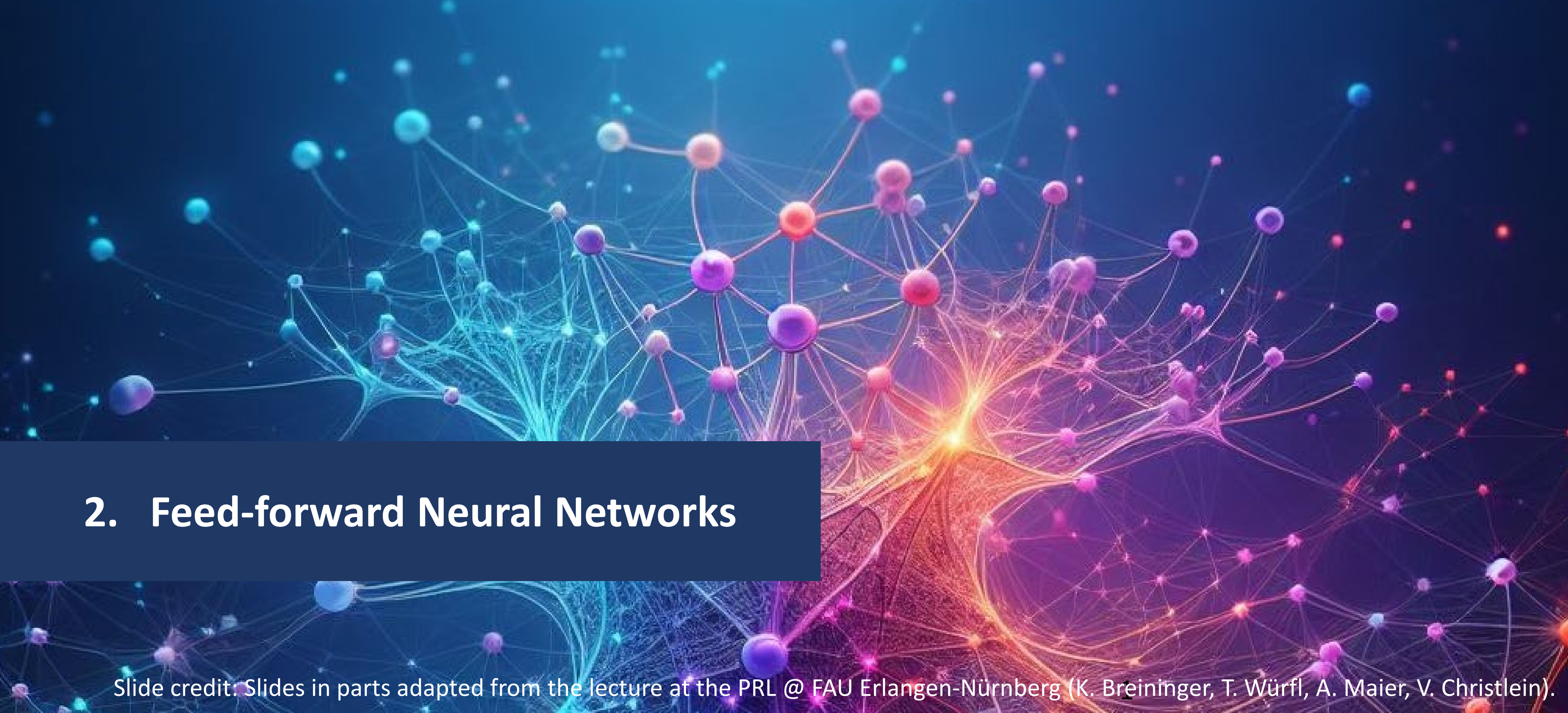
Deep Learning
Summer semester ‘24



2. Feed-forward Neural Networks

Slide credit: Slides in parts adapted from the lecture at the PRL @ FAU Erlangen-Nürnberg (K. Breininger, T. Würfl, A. Maier, V. Christlein).

Who am I and who are we?

Katharina Breininger, PhD
Group Lead

Frauke Wilm
PhD Student, Aug 2020
- ML & data science
- Domain adaptation & transfer learning

Mathias Öttl
PhD Student, January 2021
(with Andreas Maier)
- Representation learning
- Annotation & label collaboration

Zhaoya Pan
PhD Student, March 2021
(with Andreas Maier)
- Interventional imaging
- Artifact detection & robustness

Jonathan Ganz
PhD Student, April 2021
TH Ingolstadt
(with Marc Aubreville)
- Image analysis
- Interpretable ML

Maja Schlereth
PhD Student, July 2021
- Multimodal imaging
- Interpretable ML

Jingna Qiu
PhD Student, July 2021
- Active learning
- ML & data science

Jonas Utz
PhD Student, July 2022
- Morphology analysis
- 3-D data synthesis

Jonas Ammeling
PhD Student, May 2022
TH Ingolstadt
(with Marc Aubreville)
- Human behavior and ML
- Recommendation systems

Anne Tjorven Büßen
PhD Student, May 2023
- Intraoperative imaging
- Workflow analysis

Moritz Schillinger
PhD Student, Dec 2023
- Optoacoustic imaging
- Hyperspectral data

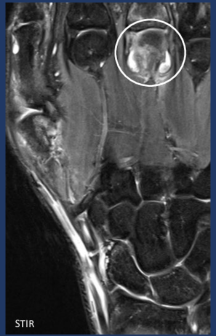
Who am I and who are we?

Friedrich-Alexander University Erlangen-Nürnberg → Julius-Maximilians-University Würzburg

AI in Medical Imaging Lab → Pattern Recognition

Intraoperative & Multimodal Imaging

https://www.healthcare.siemens.com



Prof. Dr. Ostendorf
Heinrich-Heine-Universität Düsseldorf

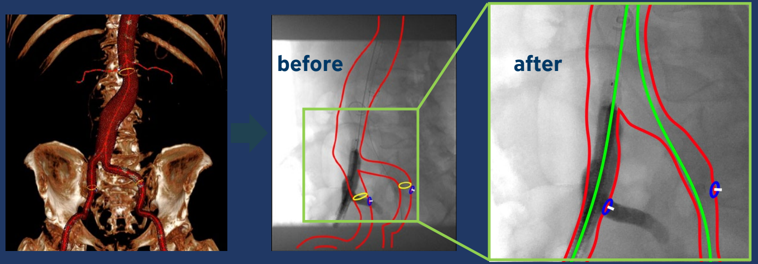


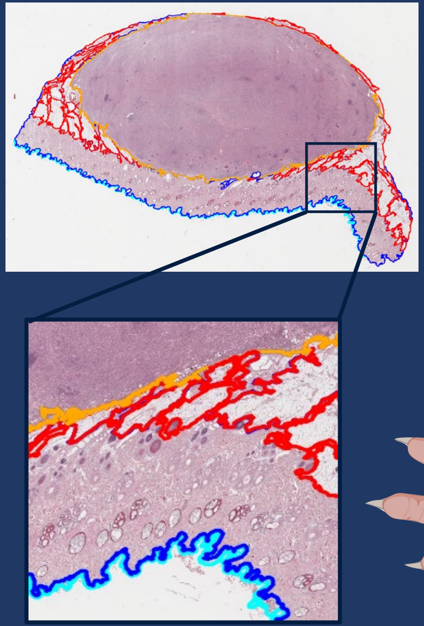
Image courtesy: Prof. Dr. Falkenberg, Sahlgrenska, Sweden

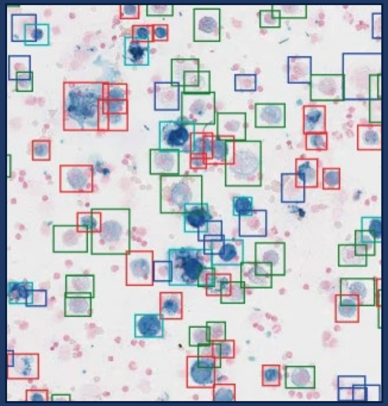
Public Datasets, Annotation & Label-efficient Learning

Machine Learning for Microscopic Imaging

□ Background 🟧 Tumor 🟦 Epidermis 🟦 Dermis 🟥 Subcutis 🟪 Inflammation/Necrosis

Image courtesy: Prof. Dr. Klopfleisch, FU Berlin





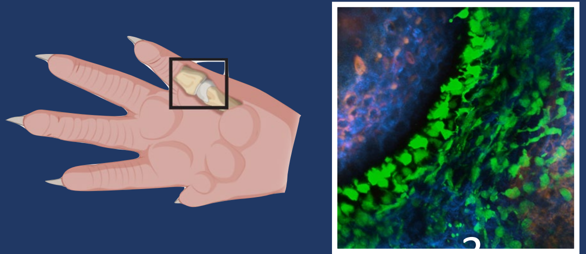


Image courtesy: Prof. Dr. Uderhardt

Goals for today

You should be able to…

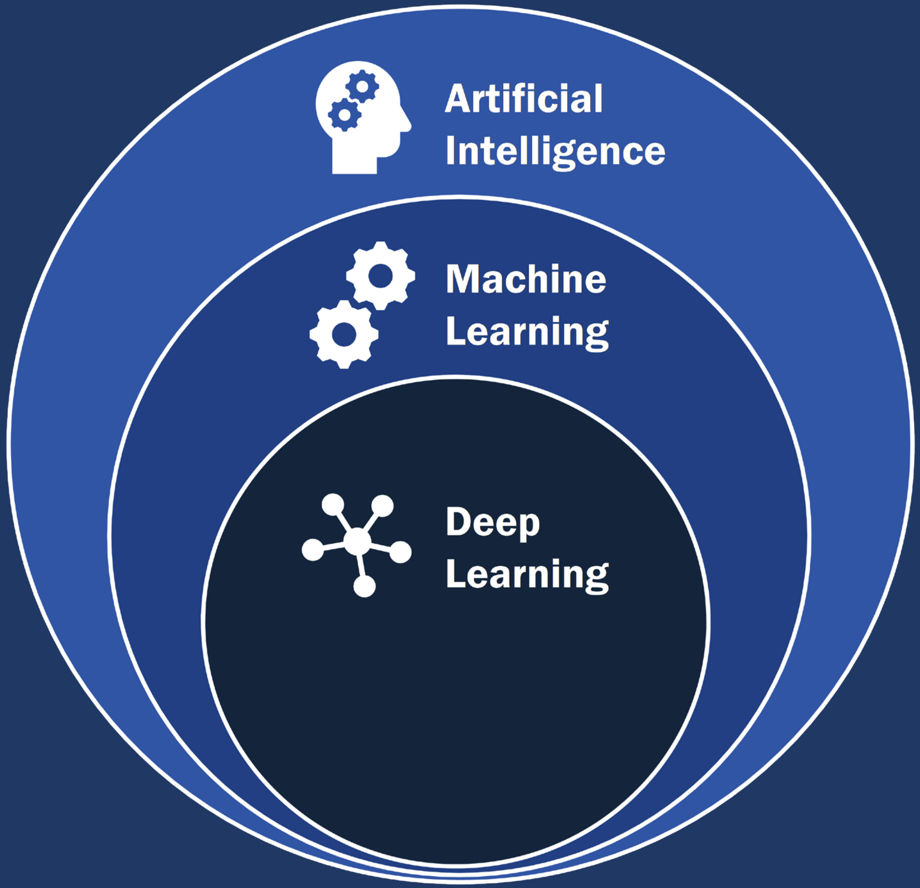
- deepen knowledge of deep learning as a concept
- understand core building blocks of (simple) neural networks
- explain why neural networks are powerful ML approaches
- understand the basics of training neural networks
- discuss benefits and drawbacks of different activation functions

Fahrplan

- Recap: Machine Learning and Deep Learning
- Perceptron
- Fully-Connected Layers and Universal approximation theorem
- From Activations to Classifications
- Credit Assignment Problem
- Activation Functions

AI vs. ML vs. DL

- AI is broader than just ML
- DL is a special type of ML
- 100% of today's **AI hype** is *caused* by DL models

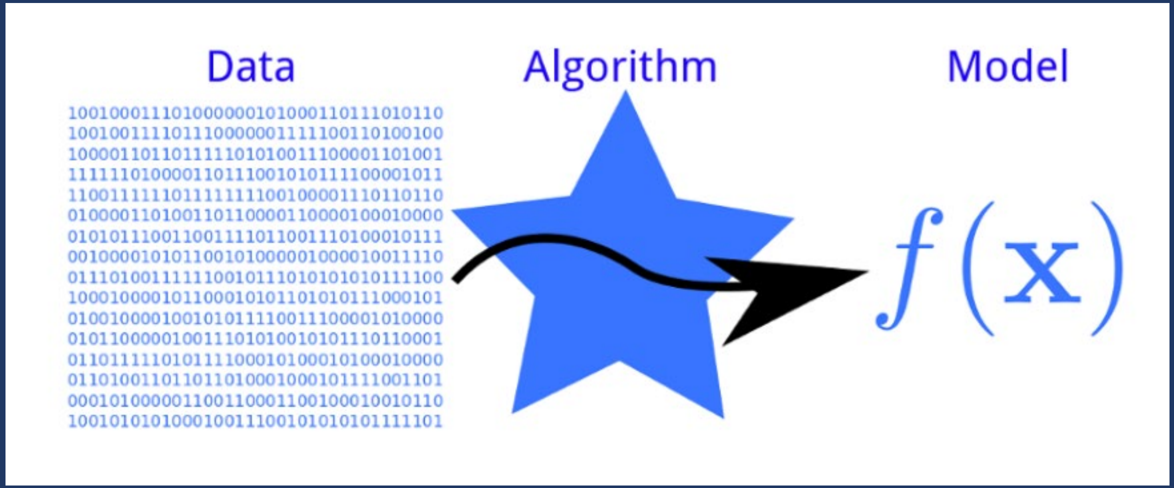


Source: https://tinyurl.com/2yy97tu3

Machine learning

Machine Learning

Machine learning denotes the multitude of algorithms for (semi-)automatic extraction of new and useful knowledge from arbitrary **collections of data** (aka **datasets**). This knowledge is typically captured in the form of rules, patterns, or **models**.



Source: https://tinyurl.com/mpd39647

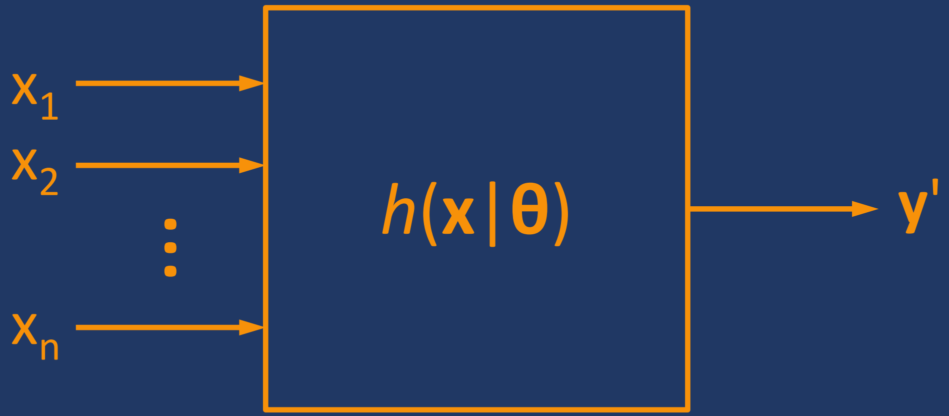
Machine Learning Components

- Any ML algorithm/approach has to have the following three components:
  - **Model**
  - **Objective**
  - **Optimization algorithm**

The Basics of ML…

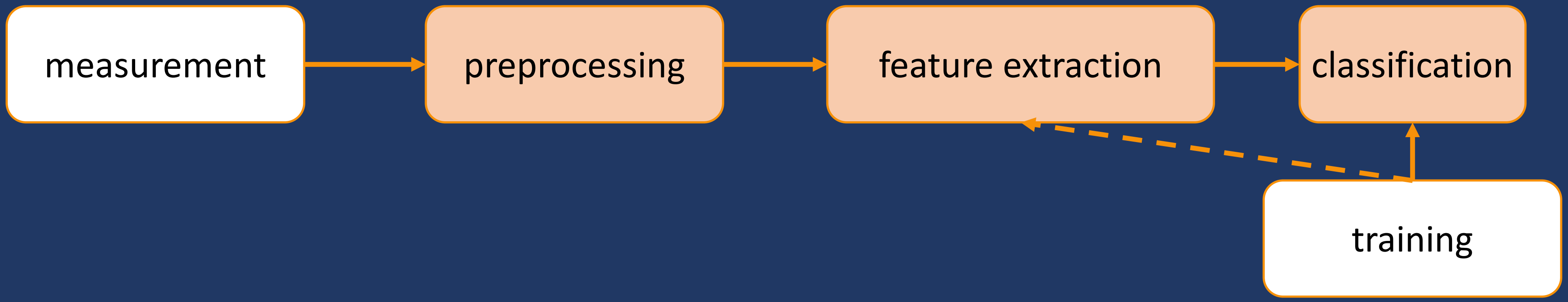
- **Input**: example represented by the **feature vector**: $\mathbf{x} = [x_1, x_2, ..., x_n]$
- **Output** (in supervised learning): the **label** y assigned to the example
  - y is a **discrete class** (in classification problems) or a **score** (in regression problems)

- A machine learning **model** *h* maps an input $[x_1, x_2, ..., x_n]$ to a label y
- The model has a set of *k* **parameters** $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_k]$: $y' = h(\mathbf{x}|\boldsymbol{\theta})$



Notation for ground truth *y* vs. prediction *y'*

“Classical” Machine Learning



- (Multi-layer) perceptron (today‘s lecture) typically works with predefined features
- „Hand-crafted“ feature design replaced by data-driven and end-to-end feature learning in state-of-the-art architectures
- Most concepts are important across architectures

Supervised ML: Toy Example

- You want to **learn** a **classifier** that can differentiate between an apple and a banana

- **Instance/example**: some *concrete* apple or some *concrete* banana.
  - Feature vector **x** = [x_1, x_2, x_3, x_4, ...]

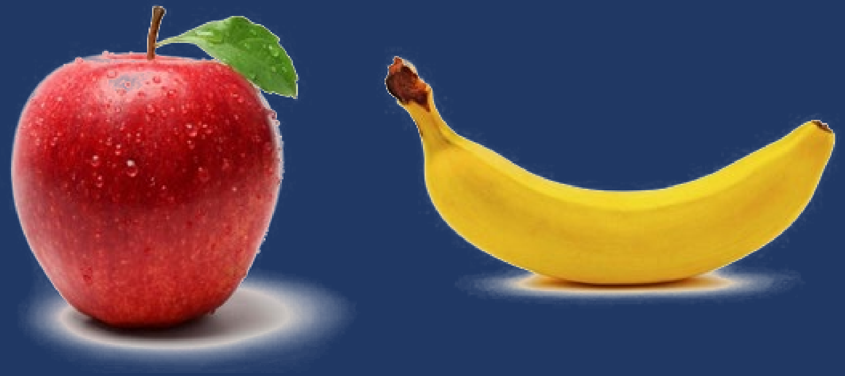
x_1: length of the fruit
x_2: circumference
x_3: weight
x_4: color
...

- **Label**: $y \in \{ c_1 = \text{apple}, c_2 = \text{banana}\}$

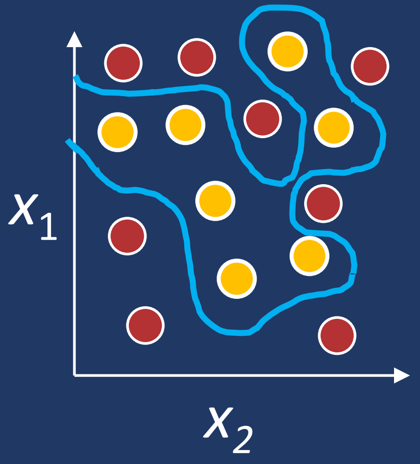
From Machine Learning to Representation Learning

Essential terms in the context of Deep Learning:

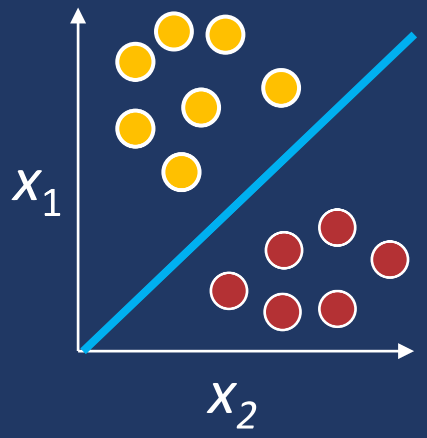
1. Representation of data
2. Transformation
3. Dimensionality reduction



x_1: top left pixel color
x_2: top right pixel color
x_3: bottom left pixel color
x_4: bottom right pixel color
…



x_1: length of the fruit
x_2: circumference
x_3: weight
x_4: average color
…



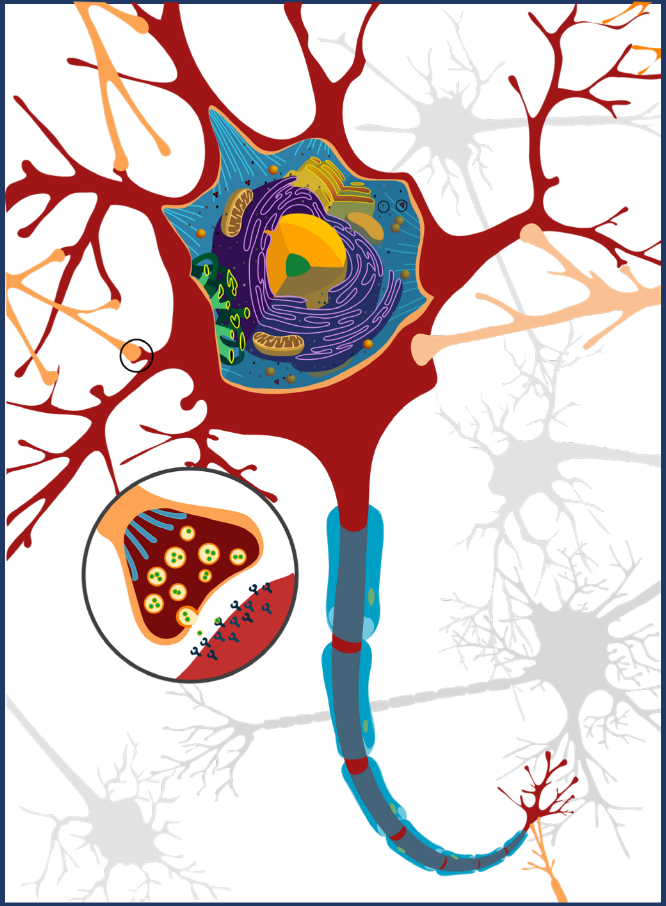
→ Goal: Make final classification (or regression) as easy as possible

Fahrplan

- Recap: Machine Learning and Deep Learning
- Perceptron
- Fully-Connected Layers and Universal approximation theorem
- From Activations to Classifications
- Credit Assignment Problem
- Activation Functions

Toward Neural Networks

- Core contribution:
  Rosenblatt's perceptron (1957) [1]
  aka: McCulloch–Pitts neuron
- **Goal:** Model a single (artificial) neuron with incoming connections
- Motivated by **biological neurons**
  - Connected by synapses
  - If the sum of incoming activations is large enough, an action potential is created
  - "All-or-nothing" response based on a threshold
  - Exhibits non-linear behavior



Adapted from Wikimedia Commons, Link

[1] Frank Rosenblatt. The Perceptron–a perceiving and recognizing automaton. 85-460-1. Cornell Aeronautical Laboratory, 1957.

The Perceptron

Learned via a suitable learning rule

- *Incoming signals*: weighted sum of inputs $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$ with weights $\mathbf{w} = [w_1, w_2, \ldots, w_n]$ and w_0

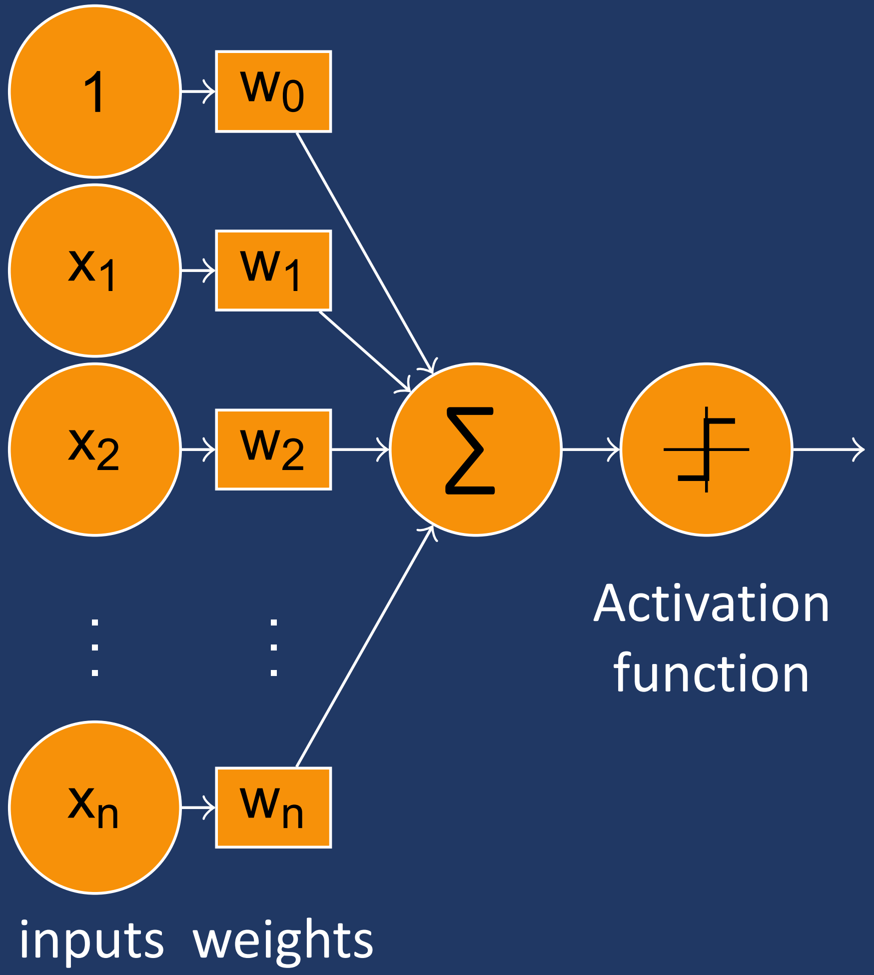
$$z = \mathbf{w}^T\mathbf{x} + w_0$$

→Linear transformation of input

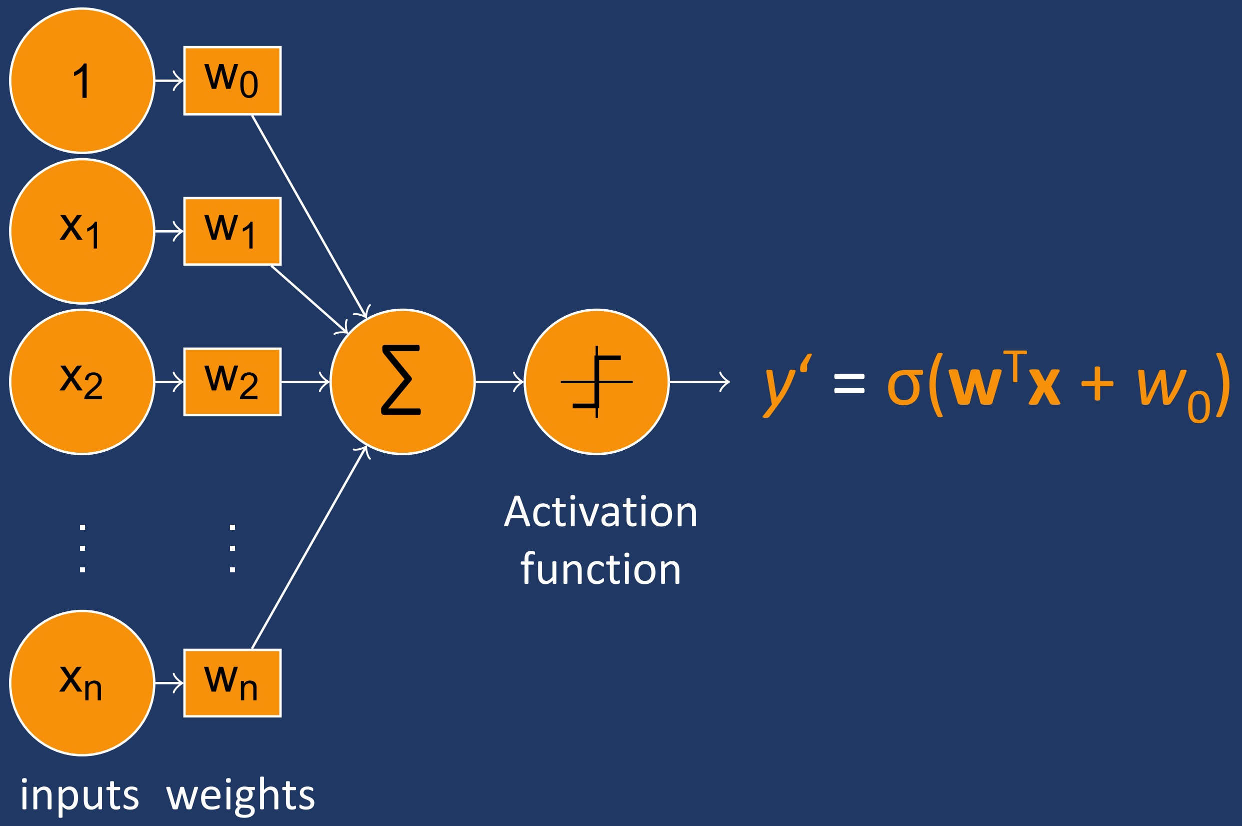
- *"All-or-nothing" response* (Heaviside):

$$y' = \sigma(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{otherwise} \end{cases}$$

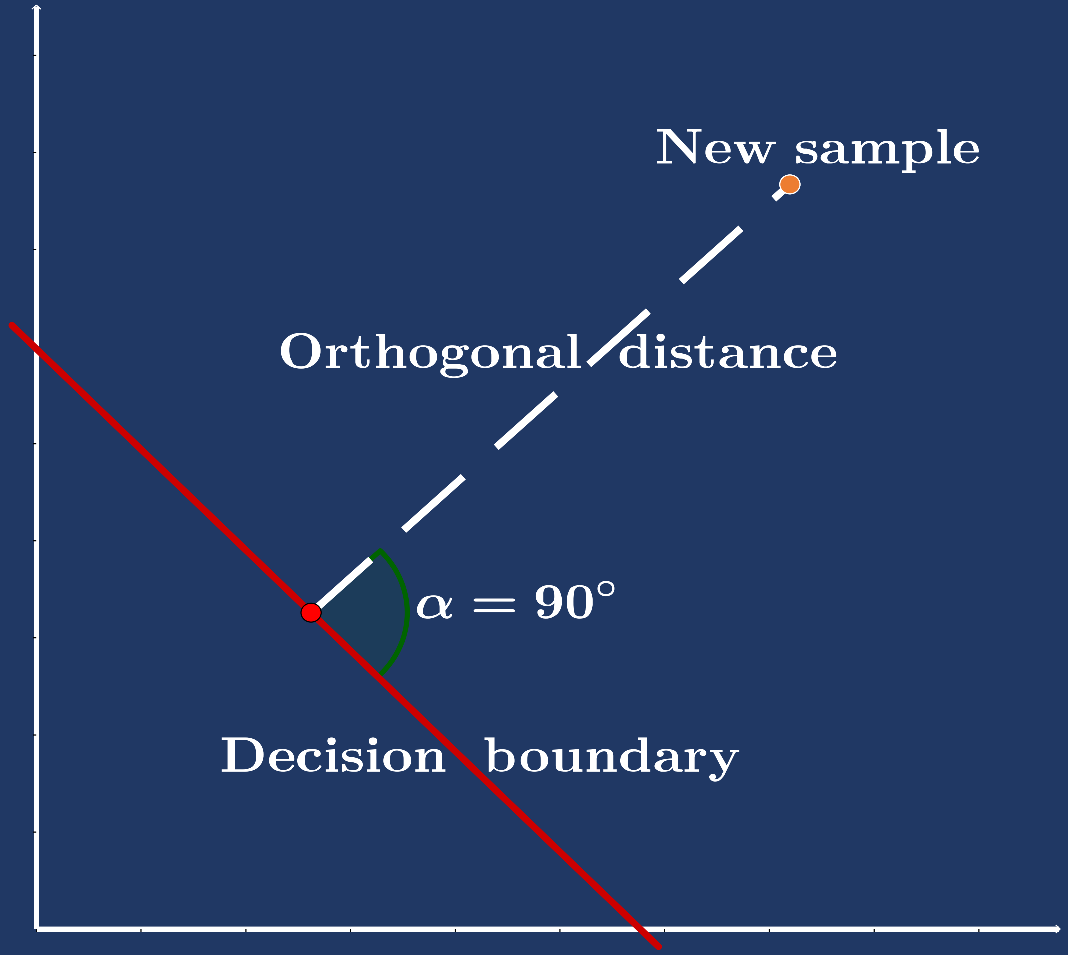
→ Binary classification $y \in \{0, 1\}$



Decision Boundary of a Perceptron

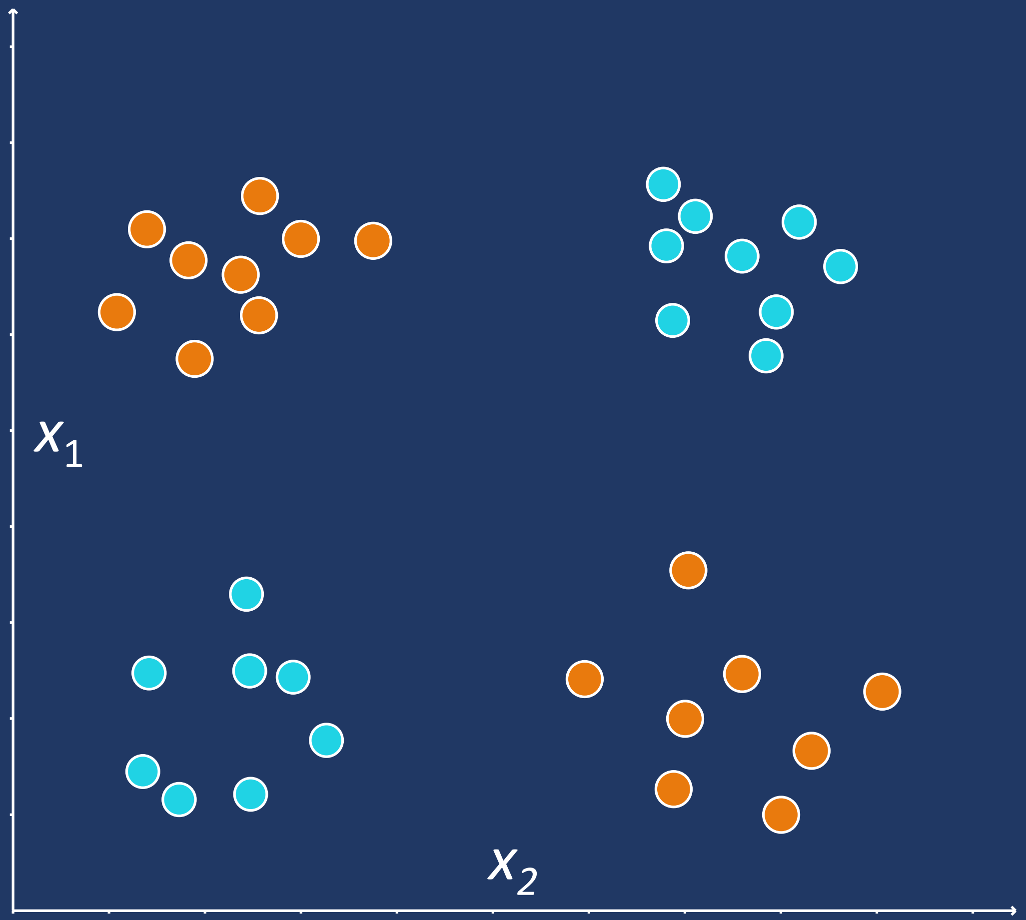


$y' = \sigma(\mathbf{w}^T\mathbf{x} + w_0)$



XOR-Problem

- Q: Why is this problem (c_1:●, c_2:●) not solvable with a perceptron?
- No linear projection exists that separates the two classes
- 1969: “Perceptrons” [2] described limitations of neural networks
  → First “AI winter”



---

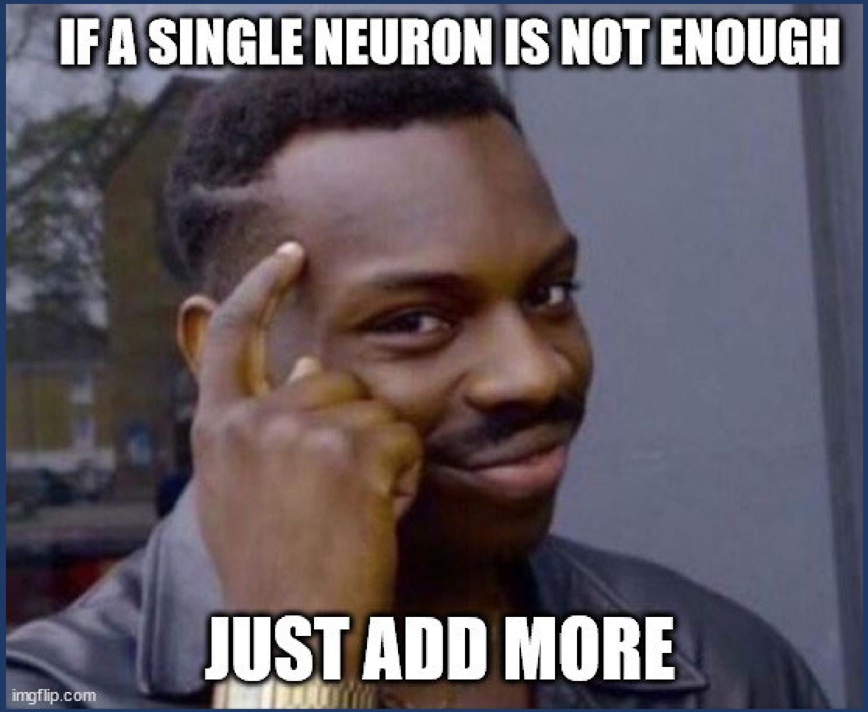
[2] Marvin Minsky, Seymour A. Papert. Perceptrons: An Introduction to Computational Geometry. The MIT Press, 2017 (Original 1969/1987). https://doi.org/10.7551/mitpress/11301.001.0001

Fahrplan

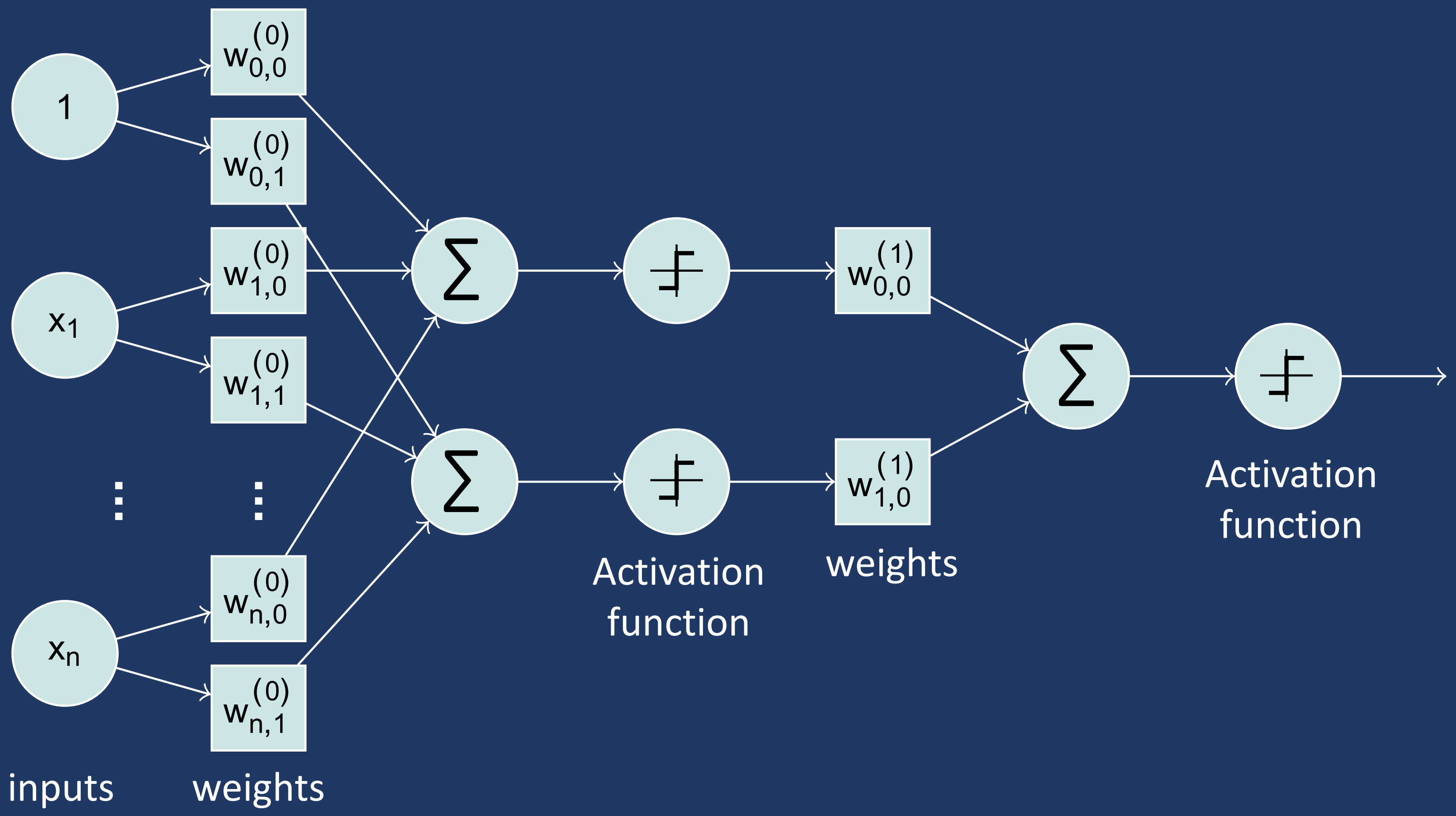
- Recap: Machine Learning and Deep Learning
- Perceptron
- Fully-Connected Layers and Universal Approximation Theorem
- From Activations to Classifications
- Credit Assignment Problem
- Activation Functions

From Single to Multilayer Perceptrons

- A single perceptron ≈ a single neuron
  → complex decisions need many neurons
- Use multiple neurons as a layer
- Important synonym: fully-connected layer
- Chain layers of neurons



Multilayer Perceptron



Universal Approximation Theorem (UTA)

- Let $\sigma(\cdot)$ be a non-constant, bounded and monotonically increasing function.
- For any $\varepsilon > 0$ and any continuous function f defined on a compact subset of there exist an integer M, real constants v_i, $b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$ where $i = 1, \dots, M$, such that

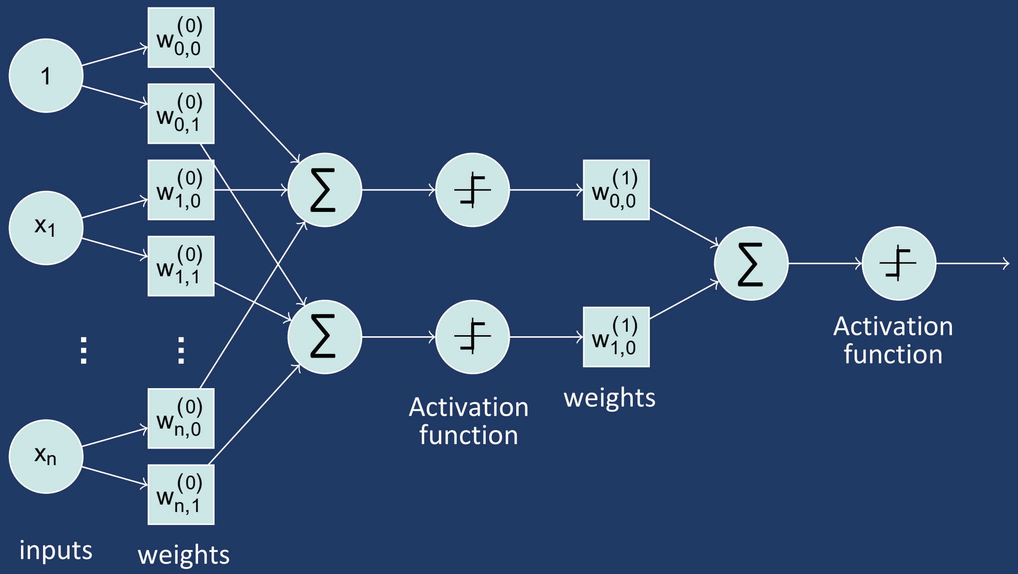
$$F(\mathbf{x}) = \sum_{i=1}^{M} v_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i) \quad \text{with}$$

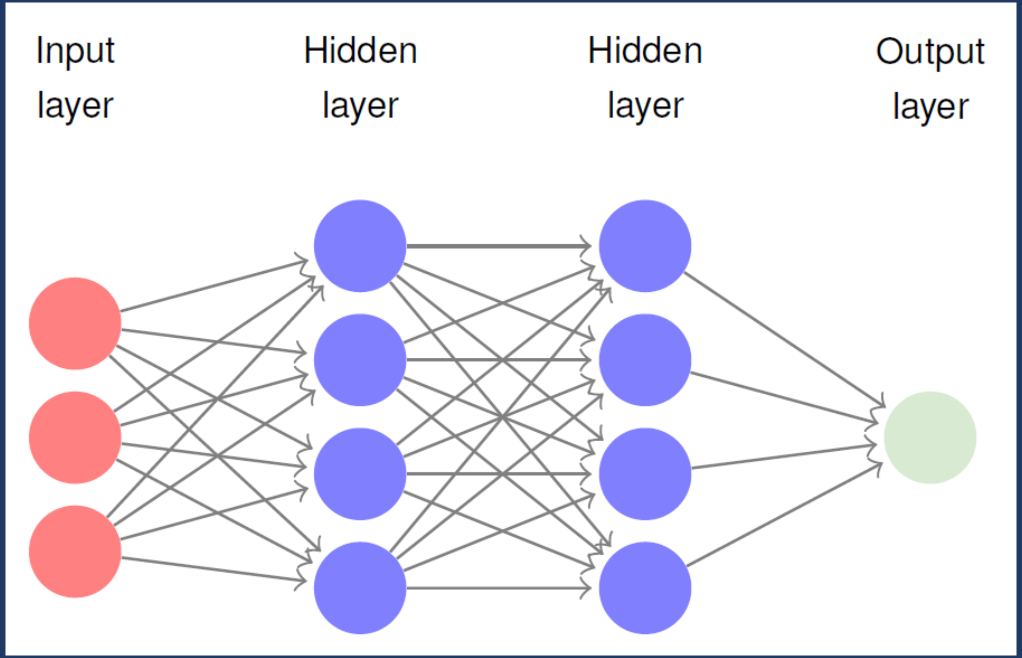
$$|F(\mathbf{x}) - f(\mathbf{x})| < \varepsilon$$

→ We can approximate any function with just one hidden layer with a sensible activation function

→ But: we have no algorithm how to: how many nodes, how to train, ...

Terminology





- Typically: Input layer, hidden layers, output layer
- A single hidden layer (of arbitrary width) can already be shown to be a *universal function approximator*
- Non-linear functions:
  - are called activation functions in hidden layers
  - provide the final output and are used for the loss function

Notation and Abstraction to Layers

Dimensionalities:

$$\mathbf{x} \in \mathbb{R}^n \rightarrow \mathbf{x}' \in \mathbb{R}^{n+1}$$

$$\mathbf{w} \in \mathbb{R}^n \rightarrow \mathbf{w}' \in \mathbb{R}^{n+1}$$

$$z/z_m \in \mathbb{R}^1$$

$$\mathbf{W} \in \mathbb{R}^{M\times(n+1)}$$

$$\mathbf{z} \in \mathbb{R}^M$$

- Single neuron:

$$z = \mathbf{w}^T\mathbf{x} + w_0 = [w_1, w_2, \ldots, w_n] \cdot \mathbf{x} + w_0$$

→Elegant vector computation:

$$z = [w_0, w_1, w_2, \ldots, w_n] \cdot [1, x_1, x_2, \ldots, x_n]^T = \mathbf{w}'^T\mathbf{x}'$$

dropping ’ for convenience

- For M neurons in a layer with ($\mathbf{w}_0, \ldots, \mathbf{w}_{m-1}$)

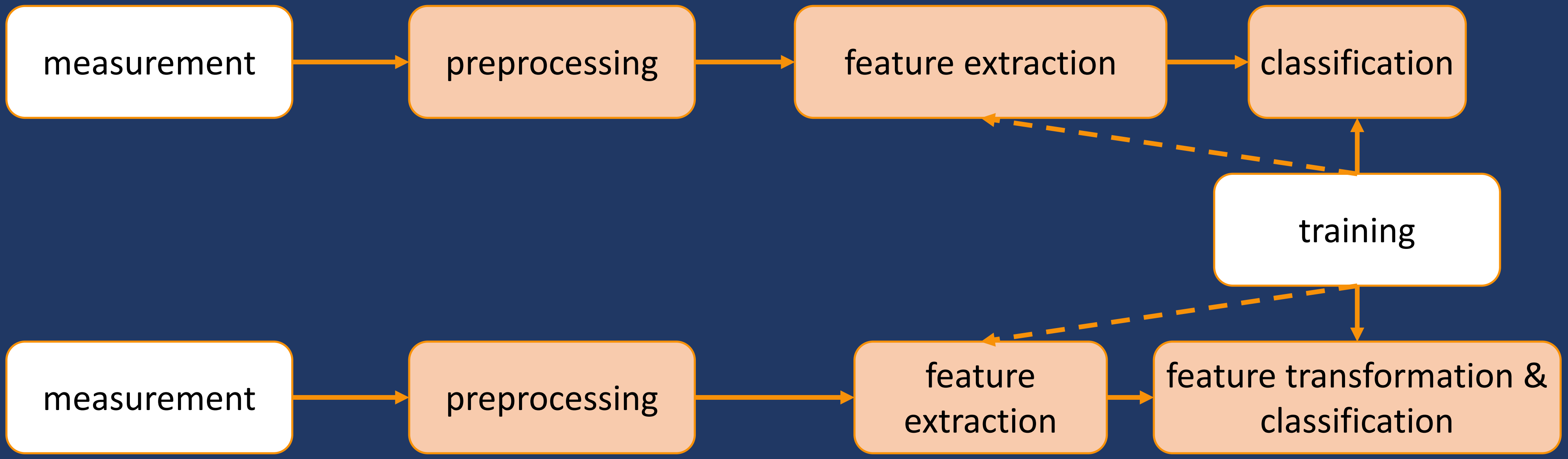
$$z_m = \mathbf{w}_m{}^T\mathbf{x}$$

- This means we can formulate a matrix multiplication → layer view

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

For layer 0: $h_0(\mathbf{x}, \mathbf{W}_0) = \sigma(\mathbf{W}_0\mathbf{x})$

“Classical” Machine Learning vs. Representation Learning

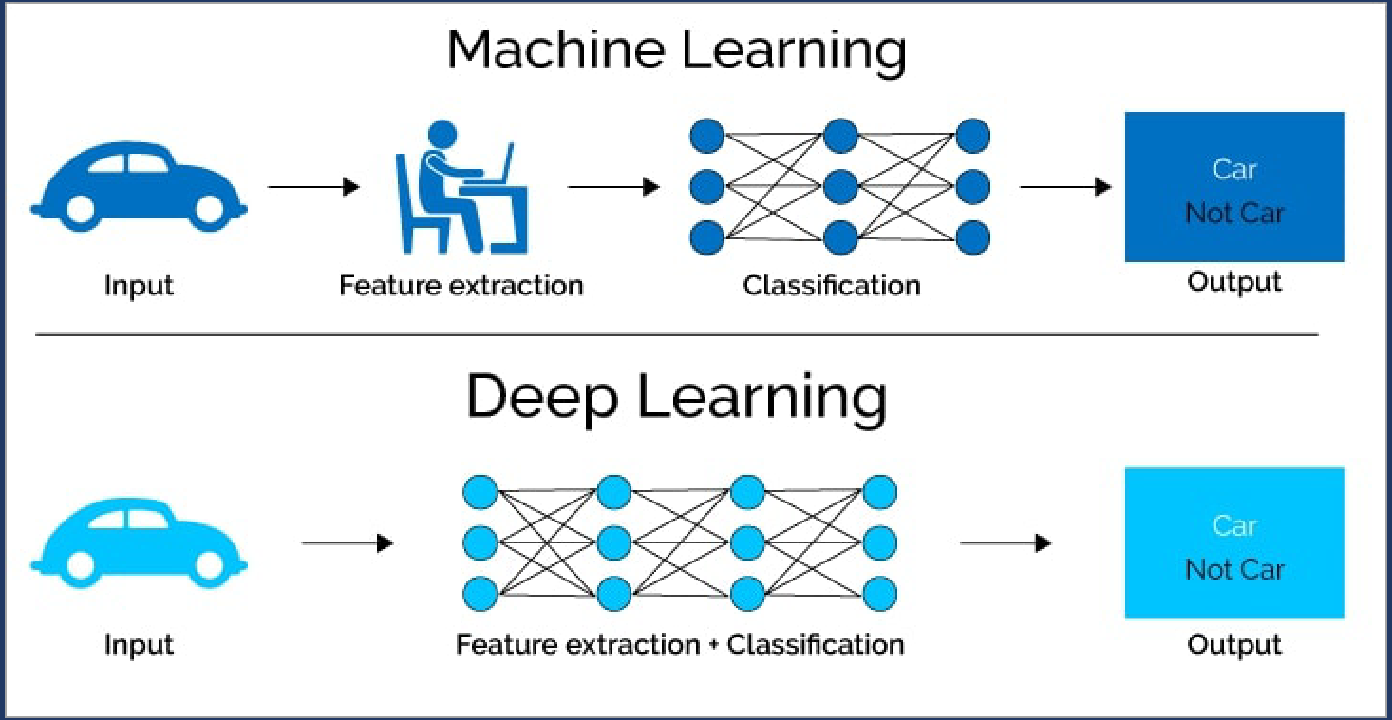


- (Multi-layer) perceptron iteratively transform features
- Neural networks are a concatenation of functions:

$$h(\mathbf{x}, \mathbf{W}) = h_{n-1}(\dots h_1(h_0(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1), \dots \mathbf{W}_{n-1}))$$

DL vs. ML: Representation Learning

The key principle of deep learning is representation learning:
Instead of precomputing features according to human intuition,
let's learn features from the raw data



Source: https://levity.ai/blog/difference-machine-learning-deep-learning

Fahrplan

- Recap: Machine Learning and Deep Learning
- Perceptron
- Fully-Connected Layers and Universal Approximation Theorem
- From Activations to Classifications
- Credit Assignment Problem
- Activation Functions

From Activations to Classification: Softmax Function

- So far: ground truth/estimated label described by $y/y' \in \{0, 1\}$
- Instead, we can use a vector $\mathbf{y} = (y_1, \dots, y_K)^T$ where K = #classes
- For exclusive classes, $\mathbf{y}$ is then:

$$y_k = \begin{cases} 1 & \text{if } k \text{ is the index of the true class,} \\ 0 & \text{otherwise} \end{cases}$$

- Called one-hot encoding: Only one element is ≠ 0
- Follows properties of a probability distribution:

1. $\sum_{k=1}^{K} y_k = 1$
2. $y_k \geq 0 \quad \forall y_k \in \mathbf{y}$

Softmax activation function

- One-hot ground truth needs matching prediction
- Softmax-function rescales a vector **z**:

$$y'_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

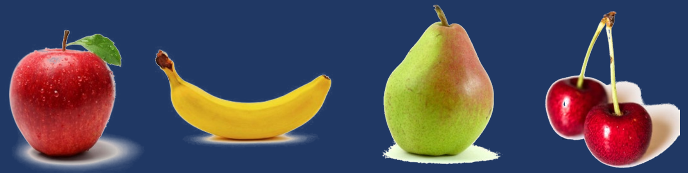
- Allows to treat the output as normalized probabilities
- Softmax function is also known as the normalized exponential function

Example: Ground truth & Softmax

- Softmax-function rescales a vector **z**:

$$y'_k = \frac{\exp(z_k)}{\sum_{j=1}^{K} \exp(z_j)}$$

- Four-class problem: **y** = $[y_1, \dots, y_4]^T$



- New sample: **y** = $[0, 1, 0, 0]^T$

Source: https://www.chefsculinar.de/rote-obstbanane-8515.htm
https://www.chefsculinar.de/chefsculinar/ds_img/assets_800/wk-01-rote_obstbanane.jpg

Label	z_k	$\exp(z_k)$	y_k'
Apple	-3.44	0.03	0.0006
Banana	1.16	3.19	0.0596
Pear	-0.81	0.44	0.0083
Cherry	3.91	49.90	0.9315

Prediction: **y'** = $[0.00, 0.06, 0.01, 0.93]^T$

Loss function

- We now have two probability distributions (ground truth/prediction) → they should be as similar as possible
- The cross entropy H of probability distributions **p** and **q**

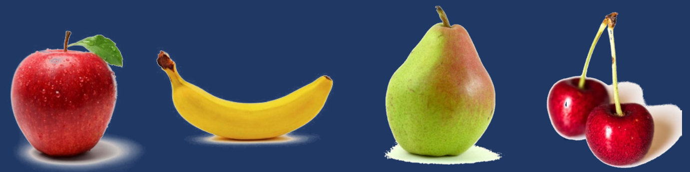
$$\mathrm{H}(\mathbf{p}, \mathbf{q}) = -\sum_{k=1}^{K} p_k \log(q_k)$$

- Based on H, we formulate a loss function L:

$$L(\mathbf{y}, \mathbf{y}') = -\log(y'_k)|_{y_k=1}$$

→ More about this in the next lecture

Example: Ground truth & Softmax



- Four-class problem: $\mathbf{y} = [y_1, \ldots, y_4]^T$

$$L(\mathbf{y}, \mathbf{y}') = -\log(y'_k)|_{y_k=1}$$

- Ground truth: $\mathbf{y} = [0, 1, 0, 0]^T$
- Prediction: $\mathbf{y}' = [0.00, 0.06, 0.01, 0.93]^T$

→ Loss / Error for this specific sample: - log(0.06) = 1.22

"Softmax loss"

- Cross-entropy and the Softmax function typically appear together

$$L(\mathbf{y}, \mathbf{z}) = -\log\left(\frac{\exp(z_k)}{\sum_{j=1}^{K}\exp(z_j)}\right)\bigg|_{y_k=1}$$

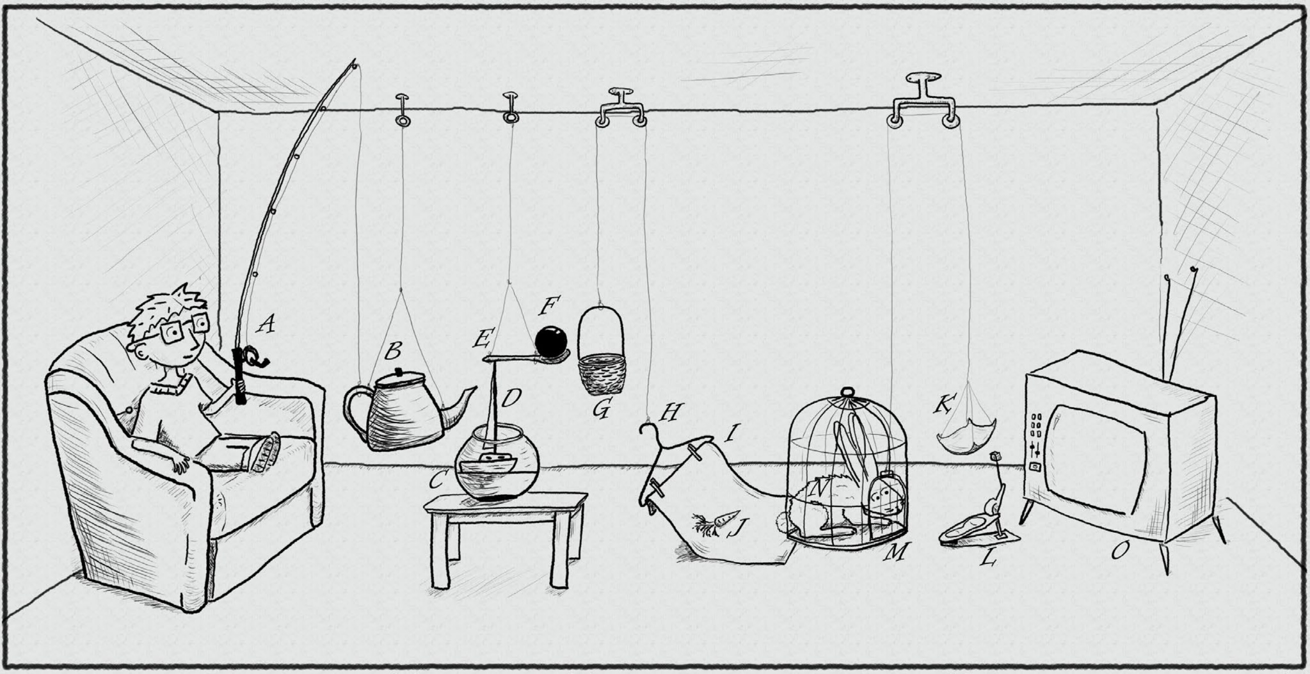
- Naturally handles multiple class problems
- Teaser: One-hot encoding, softmax, & cross-entropy allow generalization to multi-label & label smoothing (non-unique class assignments)

Fahrplan

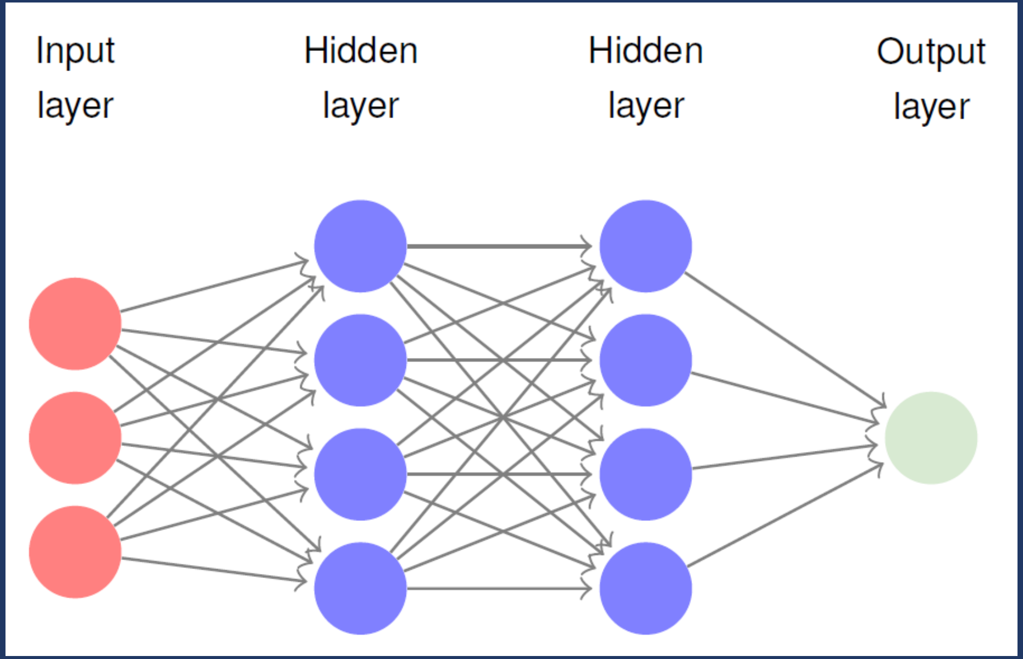
- Recap: Machine Learning and Deep Learning
- Perceptron
- Fully-Connected Layers and Universal Approximation Theorem
- From Activations to Classifications
- Credit Assignment Problem
- Activation Functions

Optimization: Credit Assignment Problem

What do these two images have in common?



https://krypt3ia.files.wordpress.com/2011/11/rube.jpg



→ Difficult to identify which parts to adjust to change the output in a specific direction

Formalization as Optimization Problem

$$h(\mathbf{x}, \mathbf{W}) = h_2(h_1(h_0(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1), \mathbf{W}_2))$$

Goal: Find *best* weights **W** for all layers

- Abstract the whole network as a function:

$$L(\mathbf{W}, \mathbf{x}, \mathbf{y})$$

- Consider all N training samples:

$$\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}(\mathbf{x},\mathbf{y})}\left[L(\mathbf{W}, \mathbf{x}, \mathbf{y})\right] = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{W}, \mathbf{x}, \mathbf{y})$$

- We want to minimize the loss criterion:

$$\underset{\mathbf{W}}{\text{minimize}} \quad \left\{L(\mathbf{W}, \mathbf{x}, \mathbf{y})\right\}$$

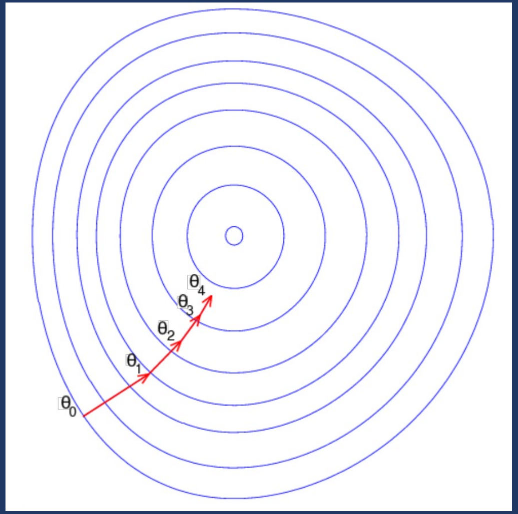
Gradient Descent

$$\underset{\mathbf{W}}{\operatorname{argmin}} \left\{ \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{W}, \mathbf{x}, \mathbf{y}) \right\}$$

Method of choice: Gradient Descent

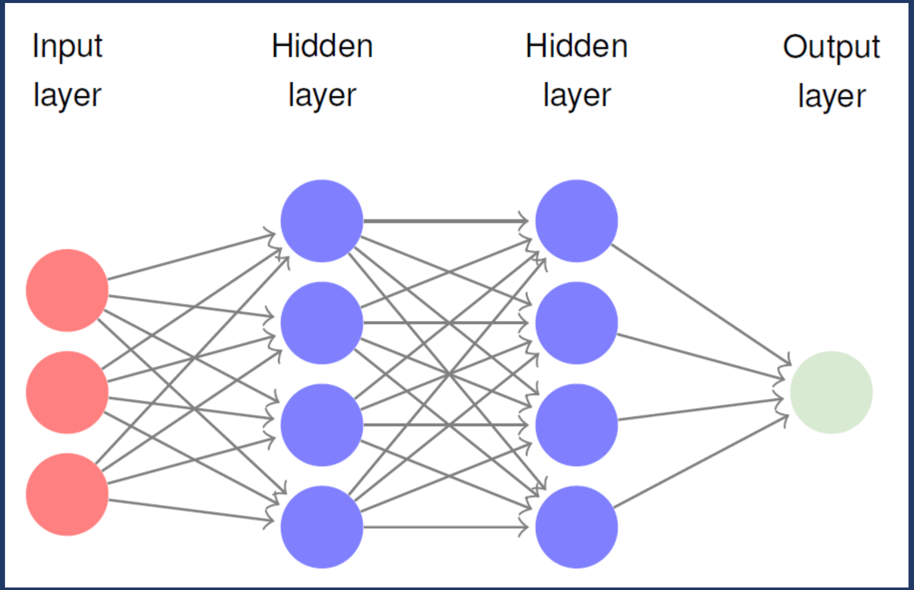
1. Initialize **W**
2. Iterate until convergence

$$\mathbf{w}^{k+1} = \mathbf{w}^{k} - \eta \nabla_{\mathbf{w}} \frac{1}{M} \sum_{m=1}^{M} L(\mathbf{w}, \mathbf{x}, \mathbf{y})$$



where η is commonly referred to as the learning rate

What is this L we are trying to optimize?



Complex network can be seen as a composed functions:

$$h(\mathbf{x}, \mathbf{W}) = h_2(h_1(h_0(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1), \mathbf{W}_2))$$

→ Gradient for each weight matrix needs to be determined

Backpropagation – Excessively Applying the Chain Rule

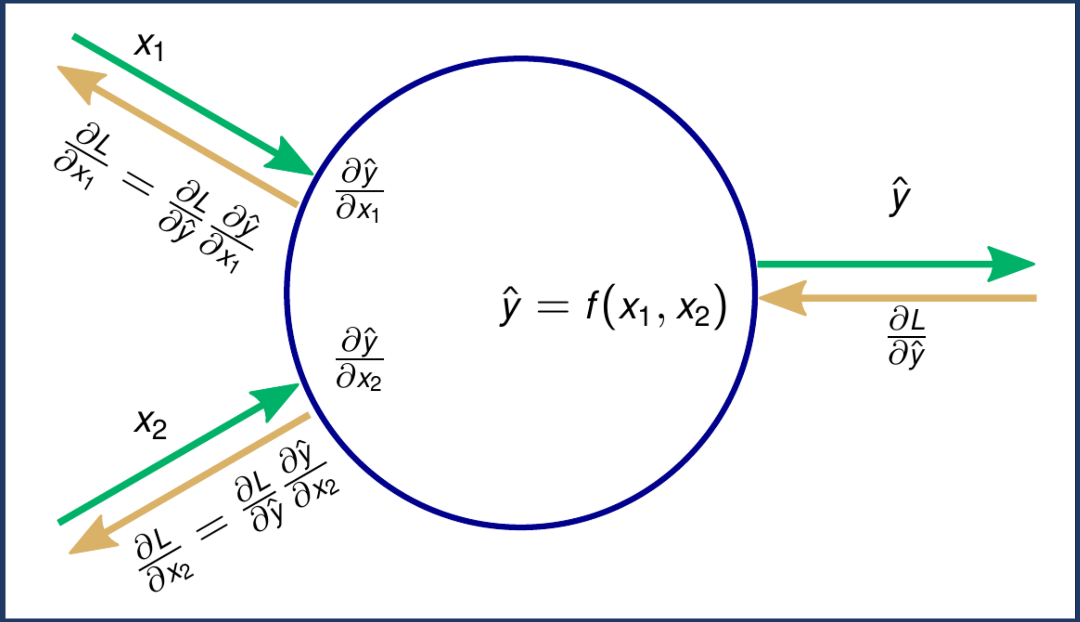
- Network is a set of composed (linear and non-linear) functions

$$h(\mathbf{x}, \mathbf{W}) = h_2(h_1(h_0(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1), \mathbf{W}_2))$$

- Chain rule:

$$\frac{d}{dx} f(g(x) = \frac{d}{dg} f(g(x)) \cdot \frac{d}{dx} g(x)$$

- Important: Need to compute weights both for **W** and (intermediate) **z**



Additional Information on Backpropagation
Excessively Applying the Chain Rule

We define $\mathbf{y}_i = \begin{cases} h_i(\dots) & \text{for } i > 0 \\ \mathbf{x} & \text{otherwise} \end{cases}$,

$\mathbf{z}_i = \mathbf{W}_i^T \mathbf{y}_i$

and

let $h_i(\mathbf{x}, W) = \sigma(\mathbf{W}x)$ i.e., a fully connected layer with activation function σ.

Then:

$$
\begin{aligned}
\frac{d}{d\mathbf{W}_2} L(\mathbf{W}, \mathbf{x}, \mathbf{y}) &= \frac{d}{d\mathbf{W}_2} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{W}_2} h(\mathbf{x}, \mathbf{W}) \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{W}_2} h_2(\overbrace{h_1(h_0(\mathbf{x}, \mathbf{W}_0), \mathbf{W}_1)}^{\substack{\text{To ease notation, replace with } \mathrm{y}_1 \\ \text{Does not depend on } \mathbf{W}_2}}, \mathbf{W}_2) \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{W}_2} h_2(\mathbf{y}_1, \mathbf{W}_2) \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{W}_2} \sigma(\mathbf{W}_2 \mathbf{y}_1) \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{z}_1} \sigma(\mathbf{z}_1) \cdot \frac{d}{d\mathbf{W}_2} \mathbf{W}_2 \mathbf{y}_1 \\
&= \frac{d}{dh} L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) \cdot \frac{d}{d\mathbf{z}_1} \sigma(\mathbf{z}_1) \cdot \mathbf{y}_1^T \\
&= 2(h(\mathbf{x}, \mathbf{W}) - \mathbf{y}) \cdot \mathbb{1} \cdot \mathbf{y}_1^T
\end{aligned}
$$

Matrix cookbook:
$\frac{d\mathbf{X}\mathbf{a}}{d\mathbf{X}} = \mathbf{a}^T$

For the case of $L(h(\mathbf{x}, \mathbf{W}), \mathbf{y}) = ||h(\mathbf{x}, \mathbf{W}) - \mathbf{y}||_2^2$,
and (for simplicity) $\sigma(x) = x$ (and therefore $\frac{d}{dx}\sigma(x) = 1$)

To improve your understanding:

- Think about what dimensions $\frac{d}{d\mathbf{W}_2} L(\mathbf{W}, \mathbf{x}, \mathbf{y})$ should have and how we arrive at this dimension.
- Try to derive the gradient for $\frac{d}{d\mathbf{W}_1} L(\mathbf{W}, \mathbf{x}, \mathbf{y})$ yourself. What intermediate gradients do you have to compute along the way?

Fahrplan

- **Recap: Machine Learning and Deep Learning**
- **Perceptron**
- **Fully-Connected Layers and Universal Approximation Theorem**
- **From Activations to Classifications**
- **Credit Assignment Problem**
- **Activation Functions**

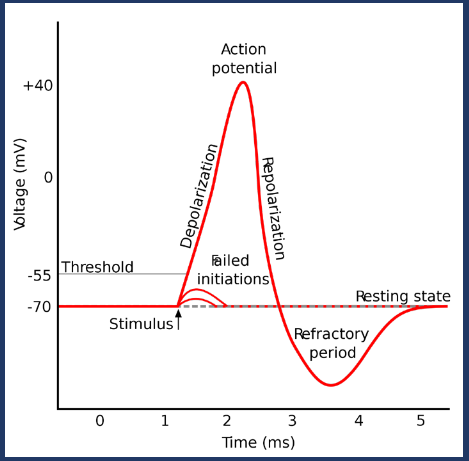
Activation Functions (Recap)

- Recap 1: Biological neurons generate “all-or-nothing” response
- Recap 2: UTA requires non-linear[1] function σ
- Recap 3: Composition of two linear transforms

$\mathbf{W}_1 \cdot \mathbf{W}_0$ is again a linear transform

→ Non-linearity “prevents” collapse

- Recap 4: In perceptron: Heaviside function

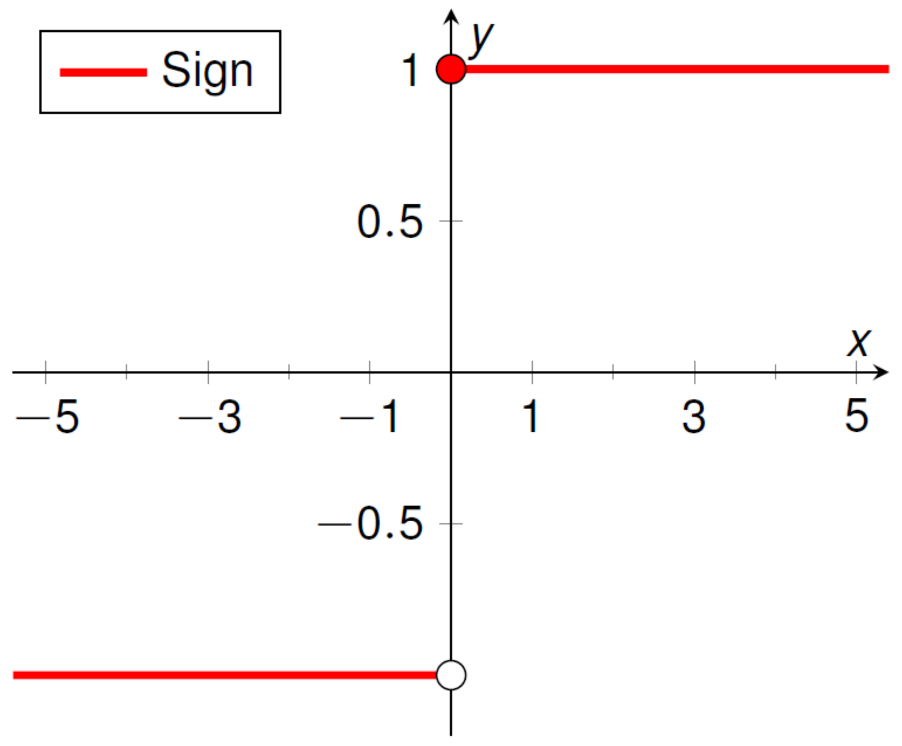


$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i \sigma(\mathbf{w}_i^T \mathbf{x} + b_i)$$

---

[1] 1: plus additional properties

Sign activation function



Sign function:

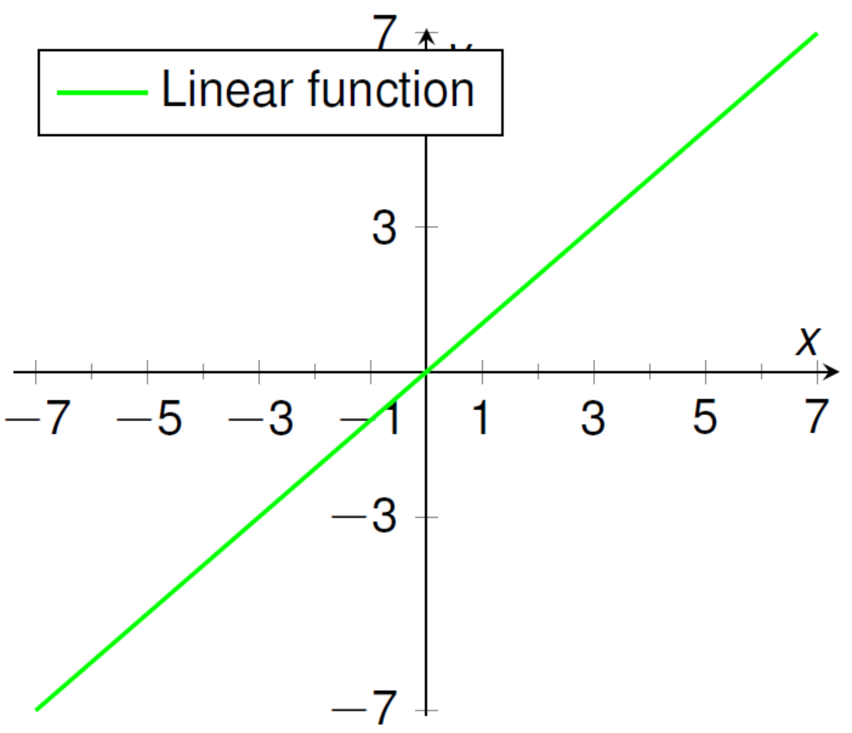
$$f(x) = \begin{cases} +1 & \text{for } x \geq 0 \\ -1 & \text{for } x < 0 \end{cases}$$

$$f'(x) = 2\delta(x)$$

$$= \begin{cases} \infty & \text{for } x = 0 \\ 0 & \text{for } x \neq 0 \end{cases}$$

- \+ Normalized output
- \- Gradient still vanishes almost everywhere
- ↯ Backpropagation

Linear activation function



Linear function with parameter α

$$f(x) = \alpha x$$

$$f'(x) = \alpha$$

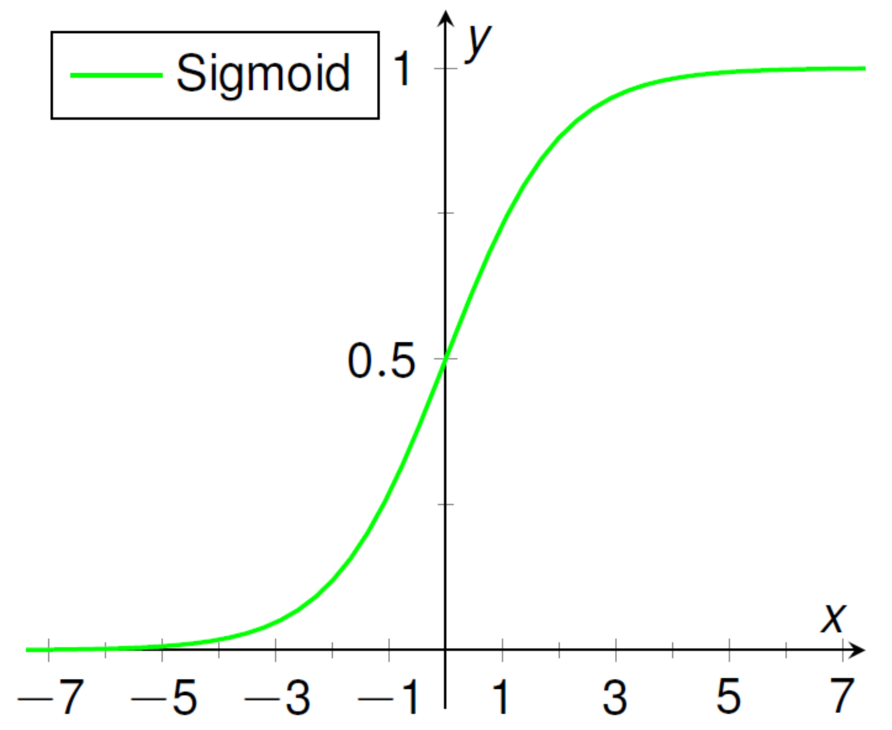
Source: https://tenor.com/de/view/captain-obvious-super-hero-superhero-gif-18644946

- Provides scaling / identity

\+ Simple, good for certain proofs

\- Does not introduce non-linearity

Sigmoid activation function



Sigmoid (logistic) function:

$$f(x) = \frac{1}{1 + exp(-x)}$$

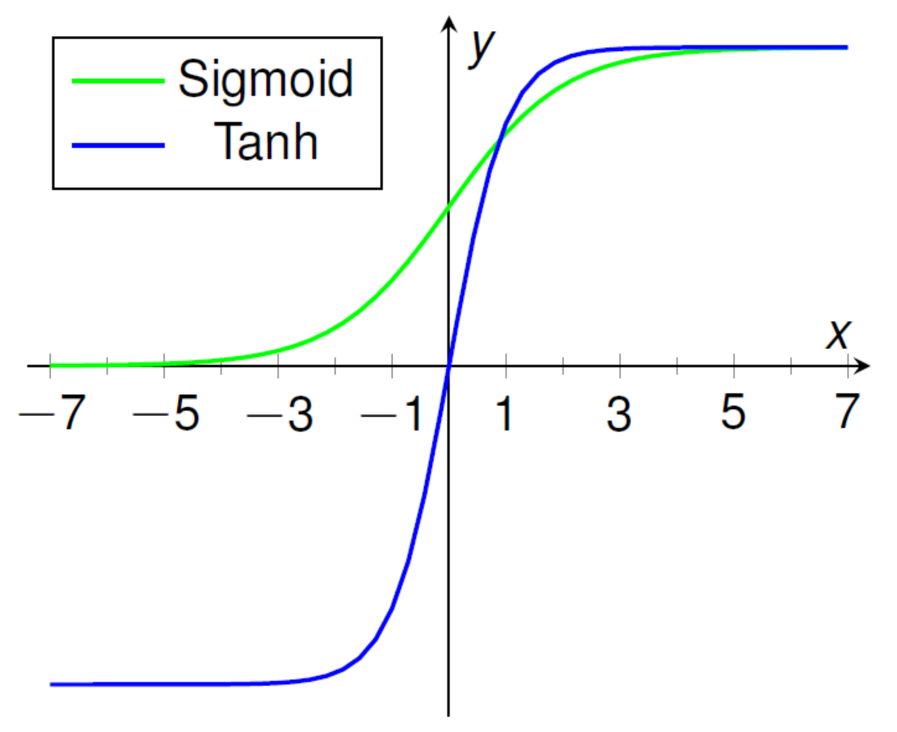
$$f'(x) = f(x)(1 - f(x))$$

- Close to biological model, but differentiable
- \+ Probabilistic output
- – Saturates for x $\ll$ 0 and x $\gg$ 0
- – Not zero-centered

Why zero-centering?

- Sigmoid: $f : \mathbb{R} \mapsto]0, 1[$
- Output of activation always $+$
  - ➔ $\nabla_{\mathbf{w}}$ will either be all $+$ or all $-$
- A mean $\mu = 0$ of the input distribution will always be shifted to $\mu > 0$
  - ➔ **co-variate shift** of successive layers
  - ➔ layers **constantly** have to **adapt** to the shifting distribution
- Batch learning reduces the variance σ of the updates

Tanh Activation Function



Tanh (hyperbolic tangent) function

$$f(x) = tanh(x)$$
$$f'(x) = 1 - f(x)^2$$

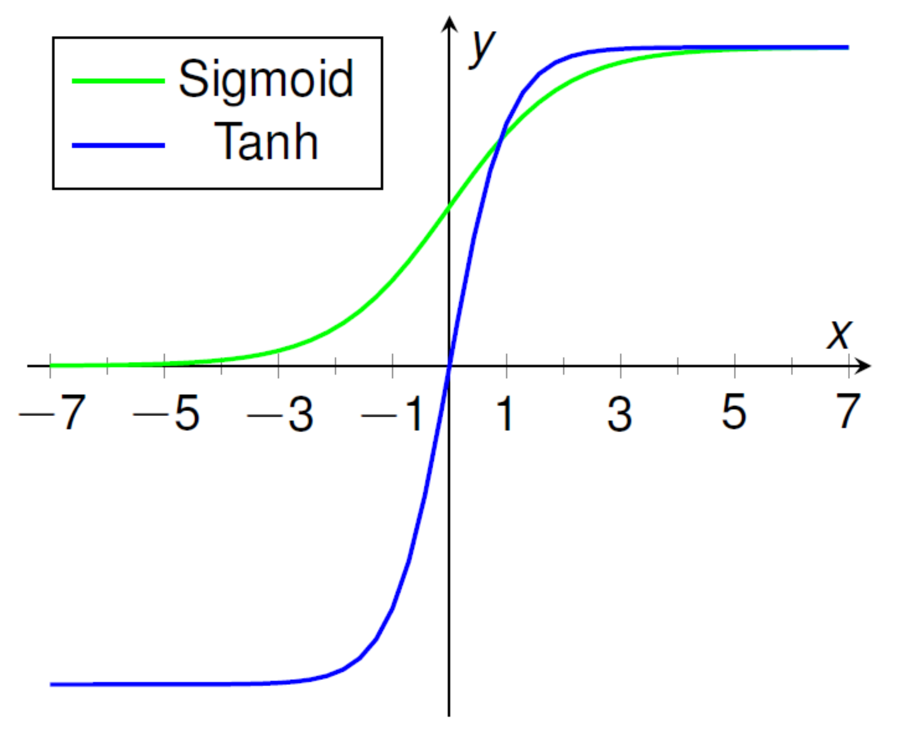
- Shifted version of the sigmoid function tanh(x) = 2σ(2x) - 1

\+ Zero-centered (LeCun ’91)

− Still saturates for x ≪ 0 and x ≫ 0

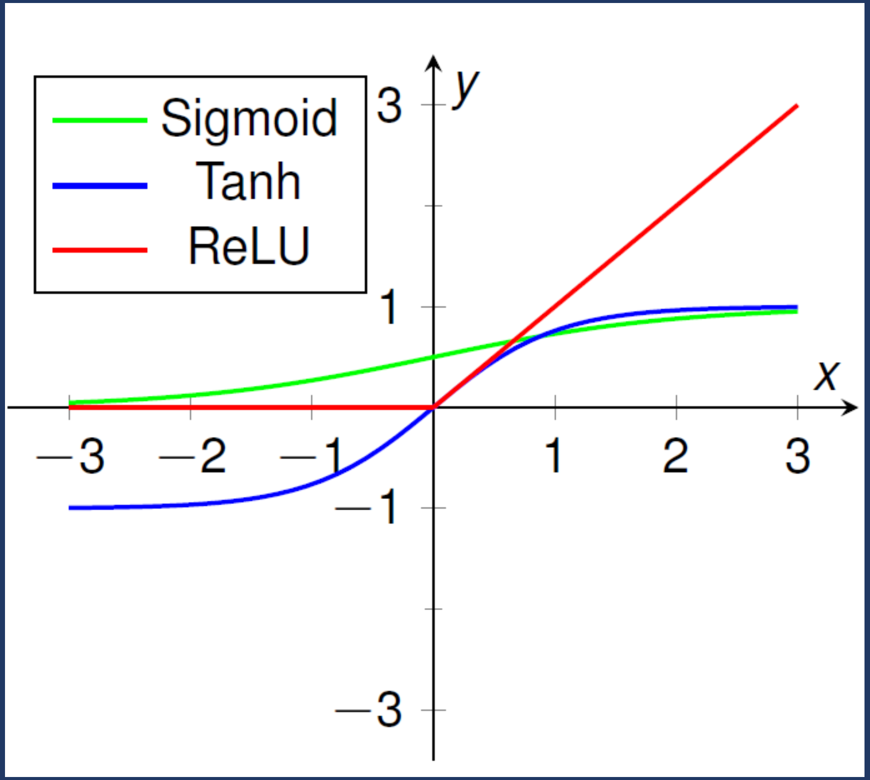
Why are vanishing gradients a problem?

- Essence of learning: How does **x** affect **y**?
- Sigmoid/tanh map large regions of **X** to a small range in **Y**
- A large change in **x** ↦ minimal change in y
- Problem is amplified by backpropagation: Multiplication of small gradients
- Related problem: Exploding gradients



Rectified Linear Unit

So vanishing gradients are a problem → linear function + non-linearity



Rectified Linear Unit (ReLU):

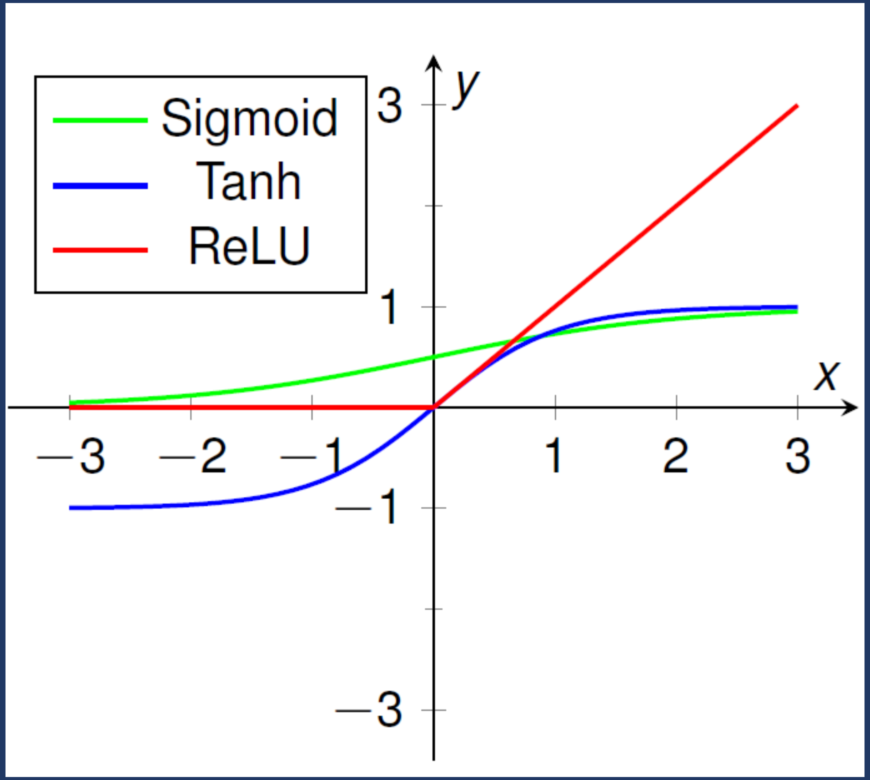
$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

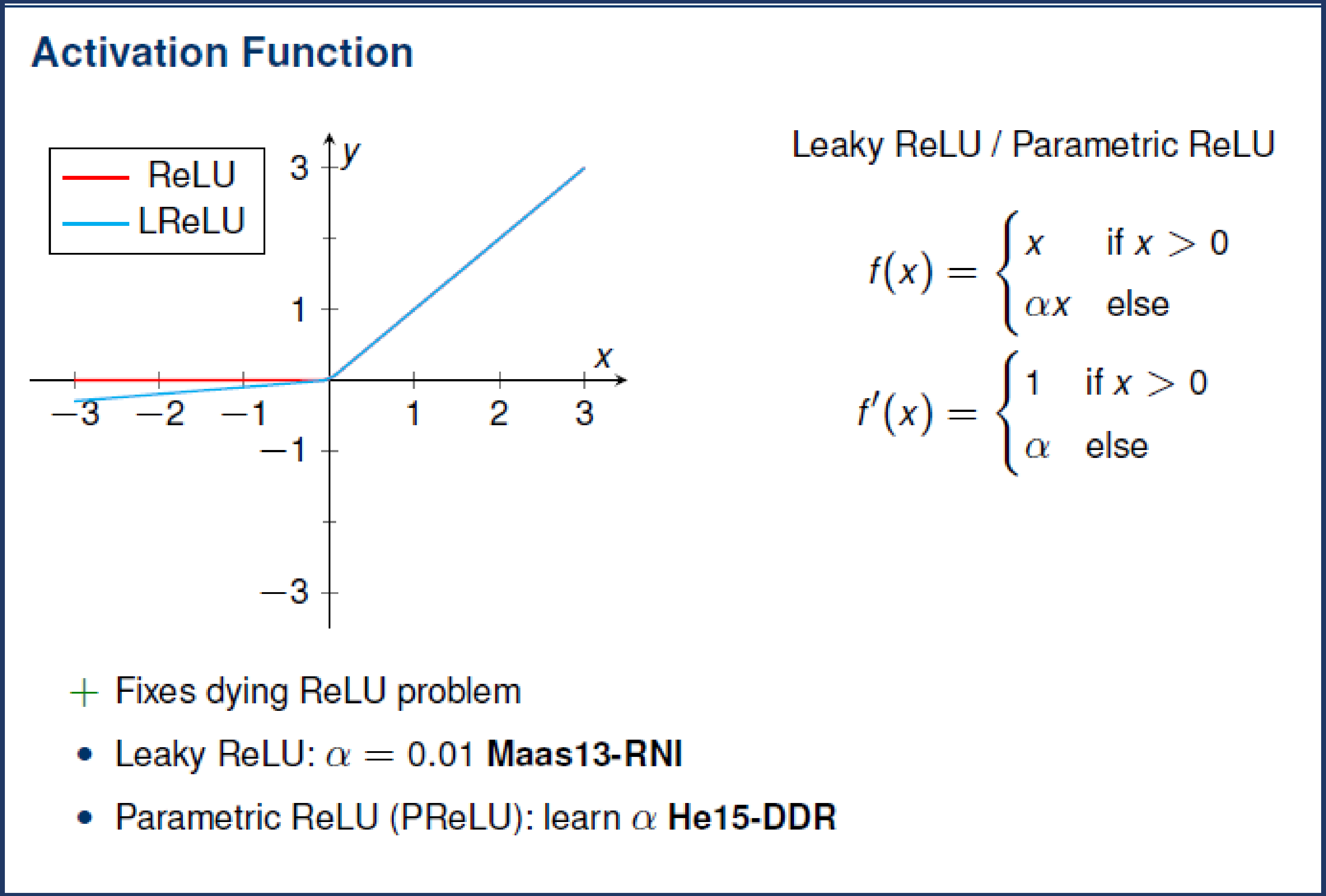
- \+ Good generalization due to piece-wise linearity
- \+ Speed up during learning (6x (Krizhevsky '12))
- \+ No vanishing gradient problem
- \- No signal <= 0
- \- Not zero-centered

Piecewise-linear Activation Function

- ReLUs were a big step forward!
- ReLUs enable deep supervised neural networks without unsupervised pretraining
- First derivative is 1 if the unit is active, second derivative is 0 almost everywhere
  → no second-order effects



Variants



Activation Function

Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

\+ Fixes dying ReLU problem

- Leaky ReLU: $\alpha = 0.01$ **Maas13-RNI**
- Parametric ReLU (PReLU): learn α **He15-DDR**

Swish/Sigmoid Linear Unit (SiLU) function

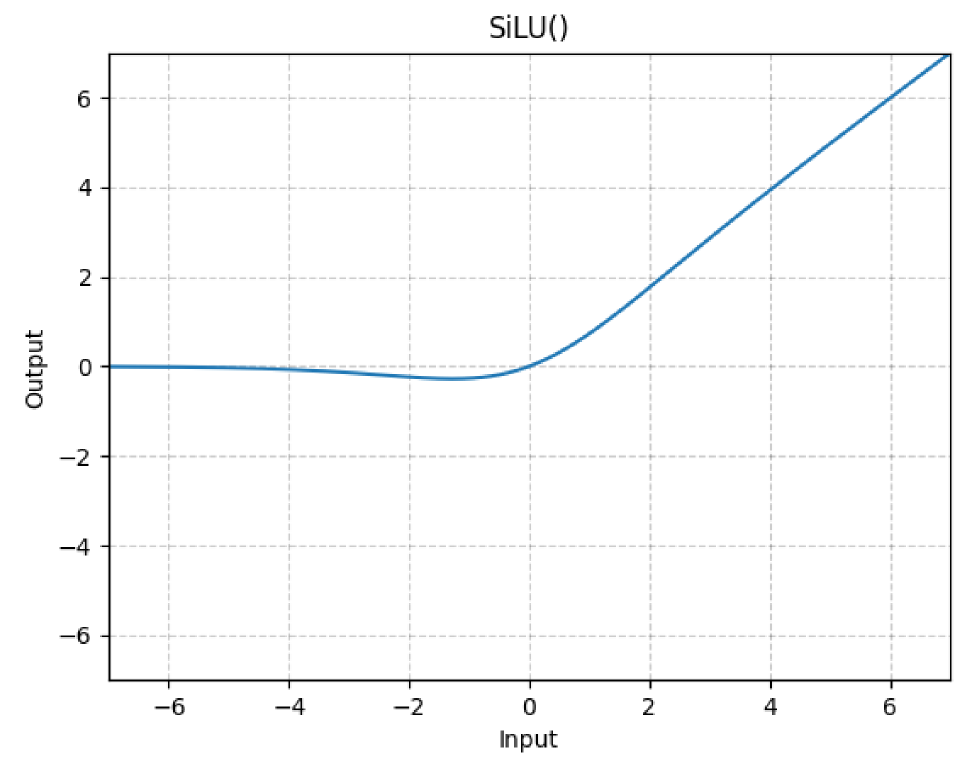
Combination of Sigmoid and ReLU:

$$f(x) = x \cdot \sigma(x)$$

$$f'(x) = \sigma(x) + x \cdot \sigma'(x)$$

- Trainable version:

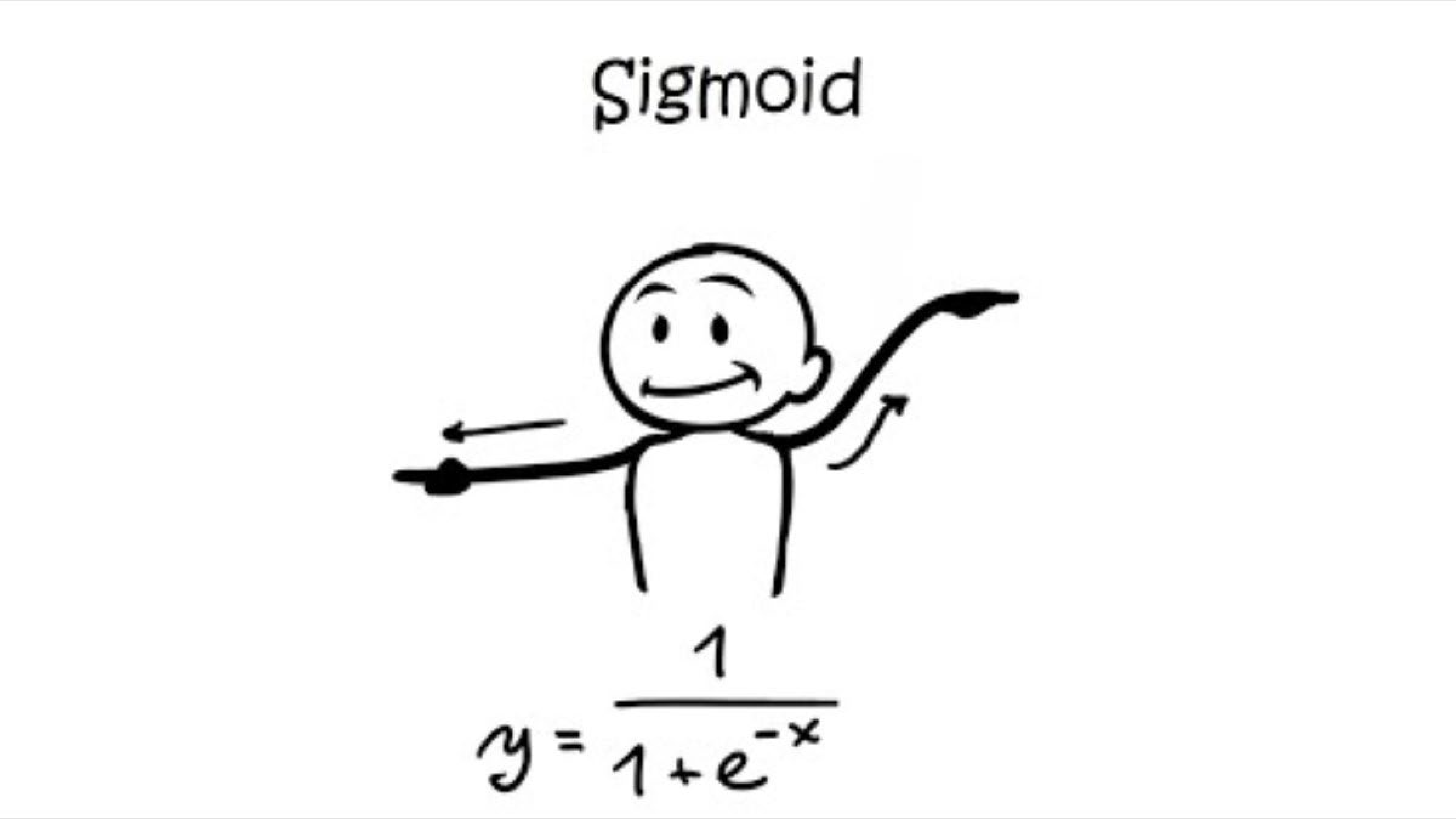
$$f(x) = x \cdot \sigma(\beta x)$$



- Preserves flow of gradients for x < 0
- Smoother gradient flow that leaky ReLU
- superior or comparable performance to ReLU on deeper models and complex datasets

→ Exercise ☺

Dancing activation functions



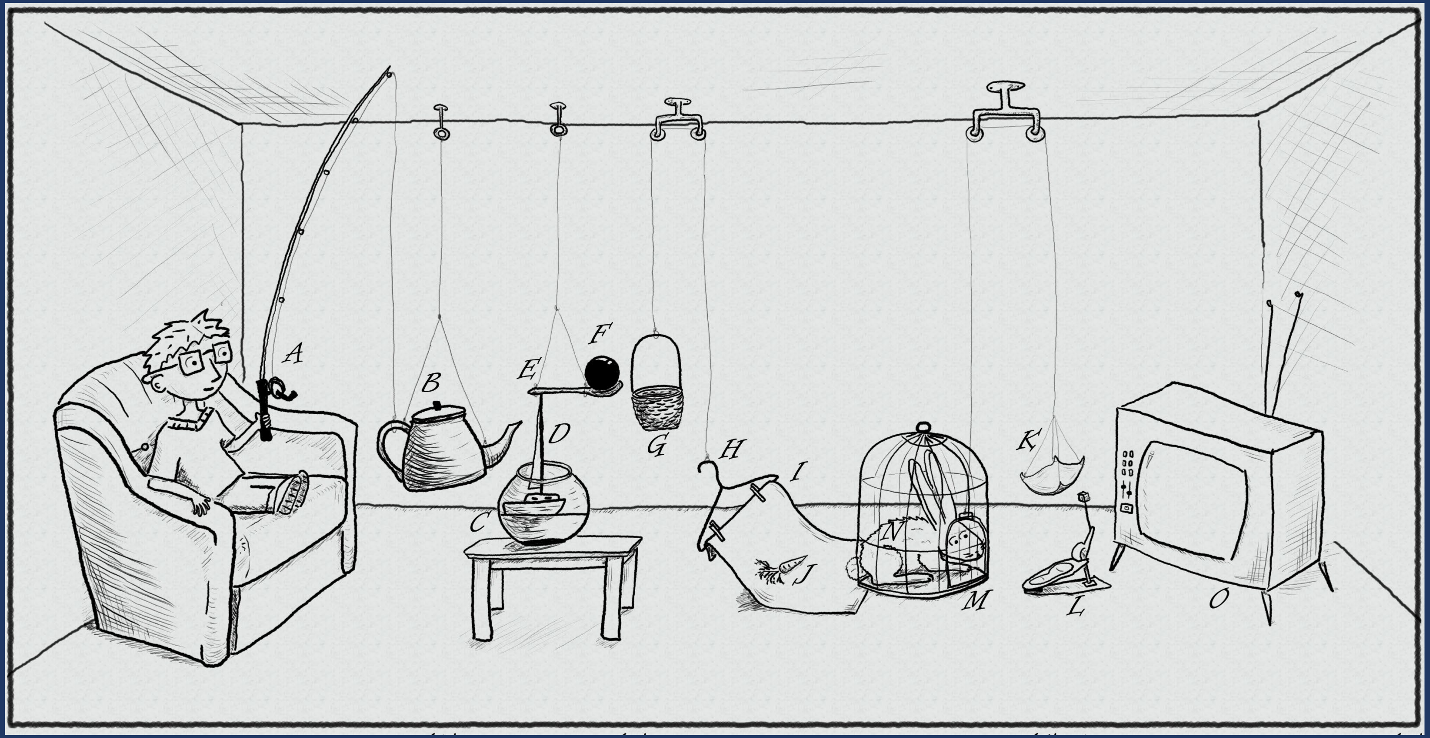
Summary

- Core building blocks:
  - Linear Transformation
  - Activation Function
  - Loss Function
- Perceptron as an artificial neuron, inspired by biology
  → linear transformation + non-linearity
- Multilayer fully-connected networks with suitable activation functions are universal function approximators (but how to get there…)
- Comparison of probability distributions: Softmax & cross-entropy
- Credit Assignment Problem: How to update what & Backpropagation
- Activation Functions: Non-linearity, no vanishing gradients, ReLU and SiLU as good standard options

NEXT TIME

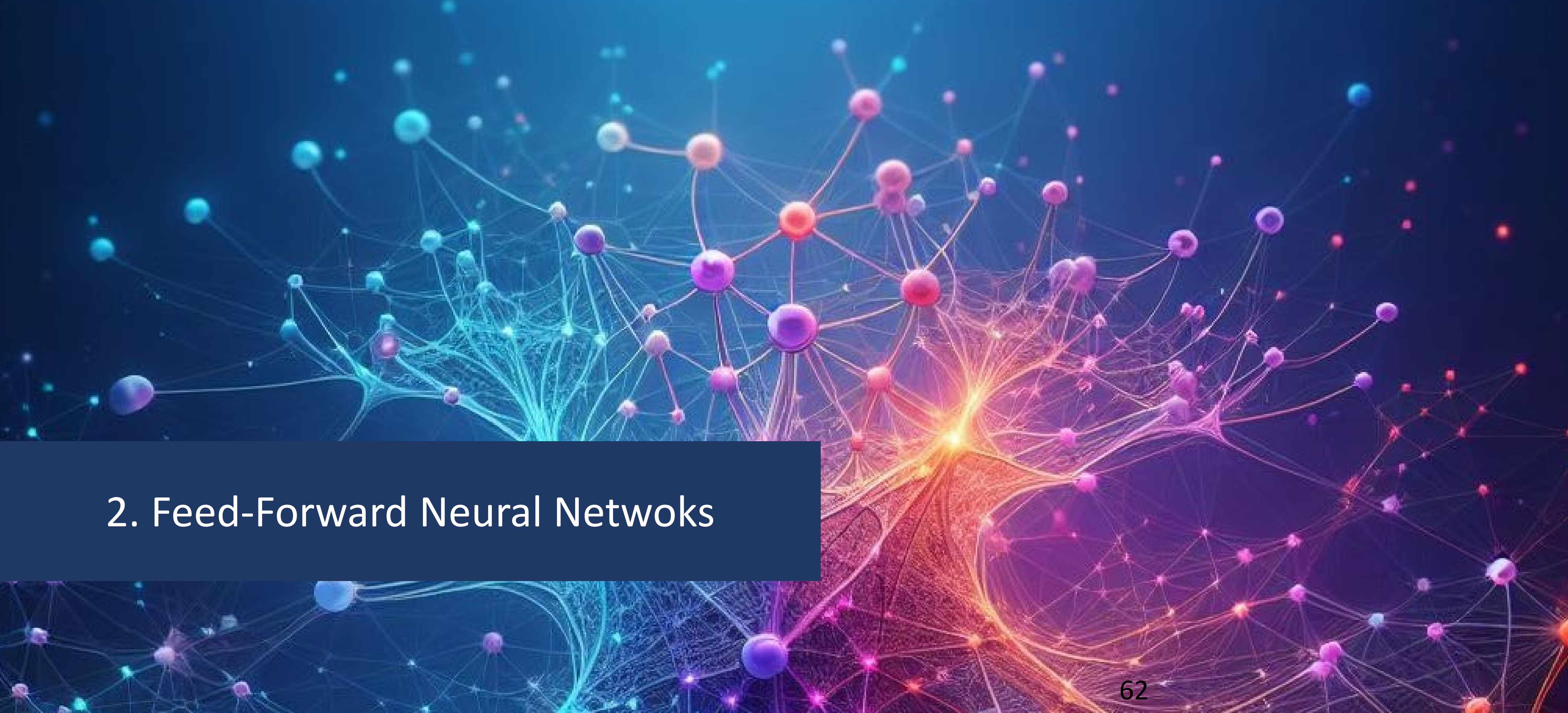
ON DEEP LEARNING

Optimization and Training (April 29)



https://krypt3ia.files.wordpress.com/2011/11/rube.jpg

Photograph by Twentieth Century Fox Film Corp., Link



Deep Learning
Summer semester ‘24

2. Feed-Forward Neural Netwoks

Learning algorithm / Update rule of the perceptron

Task: find weights that minimize the distance of misclassified samples to the decision boundary.

Training set: $(\mathbf{X}, \mathbf{Y}) = [(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_m, y_m)]$

Let M be the set of misclassified feature vectors $y_i \neq y_i' = \sigma(\mathbf{w}^T\mathbf{x}_i + w_0)$

according to a given set of weights $\mathbf{w}$

Optimization problem:

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ D(\mathbf{w}) = -\sum_{\mathbf{x}_i \in \mathcal{M}} y_i \cdot (\mathbf{w}^T \mathbf{x}_i) \right\}$$

Update rule of the perceptron

- Objective function depends on misclassified feature vectors *M*: iterative optimization
- In each iteration, the cardinality and composition ofMmay change
- The gradient of the objective function is:

$$\nabla D(\mathbf{w}) = -\sum_{x_i \in \mathcal{M}} y_i \cdot \mathbf{x}_i$$

Update rule of the perceptron

- Strategy 1: Process all samples, then perform weight update
- Strategy 2: Take an update step right after each misclassified sample
- Update rule in iteration (k + 1) for the misclassified sample $\mathbf{x}_i$ simplifies to:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha\,(y_i - y_i') \cdot \mathbf{x}_i$$

where α is the step size

- Optimization until convergence or for a predefined number of iterations

Machine Learning Components

- Any ML algorithm/approach has three components:

1. Model

  - A set of functions among which we’re looking for the „best” one

$$H = \{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$$

  - **Hypothesis** *h* = a concrete function obtained for some concrete values of $\boldsymbol{\theta}$
  - Model = set of hypotheses

Machine Learning Components

- Any ML algorithm/approach has three components:

2. Objective

- We’re looking from the best hypothesis *h* in the model H = $\{h(\mathbf{x}|\boldsymbol{\theta})\}_{\boldsymbol{\theta}}$
  - Q: But „best” according to what?
- **Objective** *J* is a function that quantifies how good/bad a hypothesis *h* is
  - Usually *J* is a „loss function” that we’re minimizing
- We’re looking for *h* (that is, values of parameters $\boldsymbol{\theta}$) that maximize or minimize the objective *J*

$$h^* = \mathrm{argmin}_{h \in H}\, J(h(\mathbf{x}|\boldsymbol{\theta}))$$
$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}}\, J(h(\mathbf{x}|\boldsymbol{\theta}))$$

- ML thus amounts to solving optimization problems

Machine Learning Components

- Any ML algorithm/approach has three components:

1. Optimization algorithm

  - An exact algorithm that we use to solve the optimization problem

$$\boldsymbol{\theta}^* = \mathrm{argmin}_{\boldsymbol{\theta}}\, J(h(\mathbf{x}|\boldsymbol{\theta}))$$

  - Selection/type of the optimization algorithm depends on the two functions – the model H and the objective J