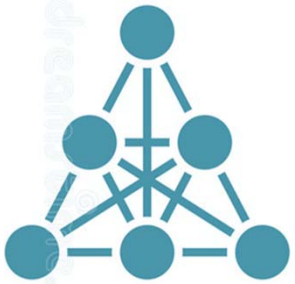


Introduction to Neural Networks

Matthias Zeppelzauer

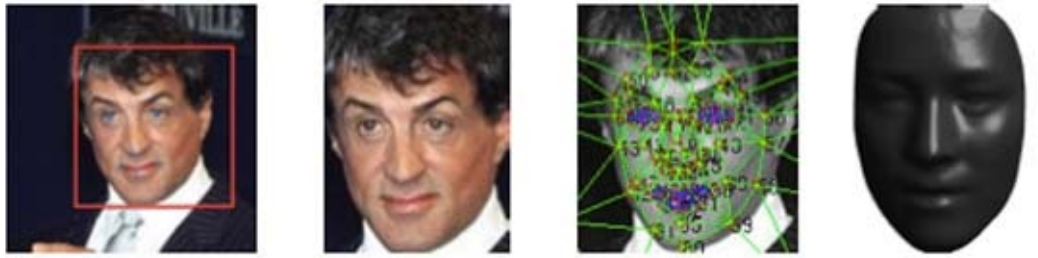
Applied Computer Vision

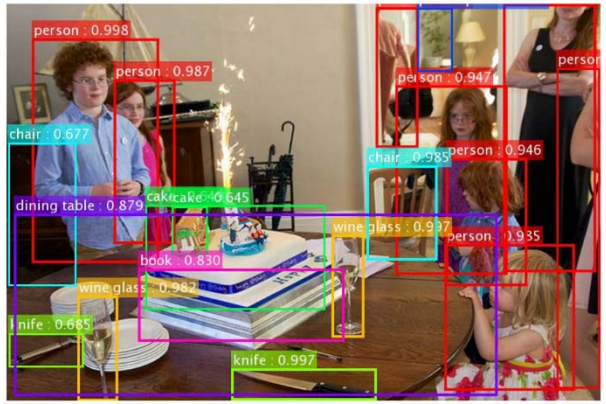
MOTIVATION

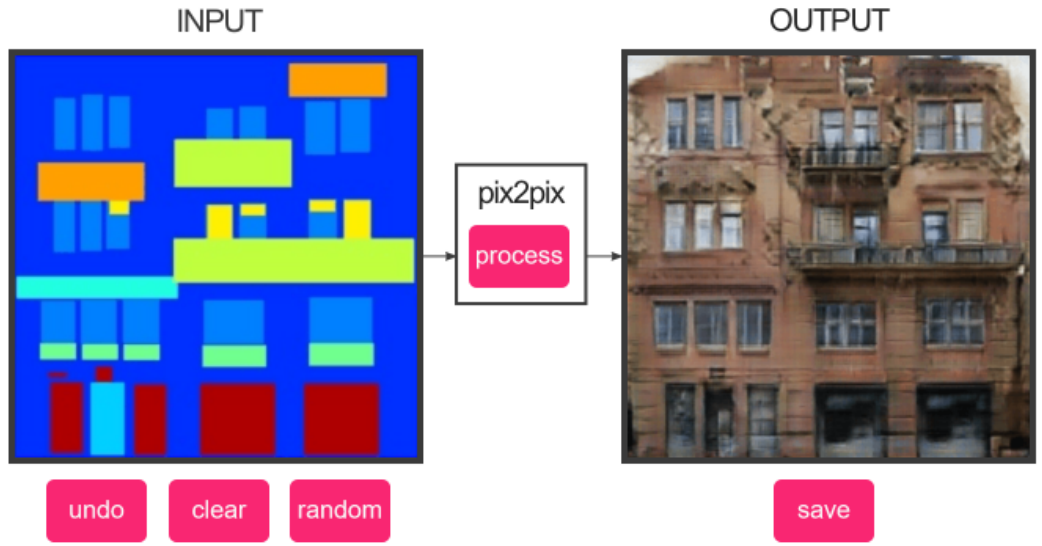
DeepL

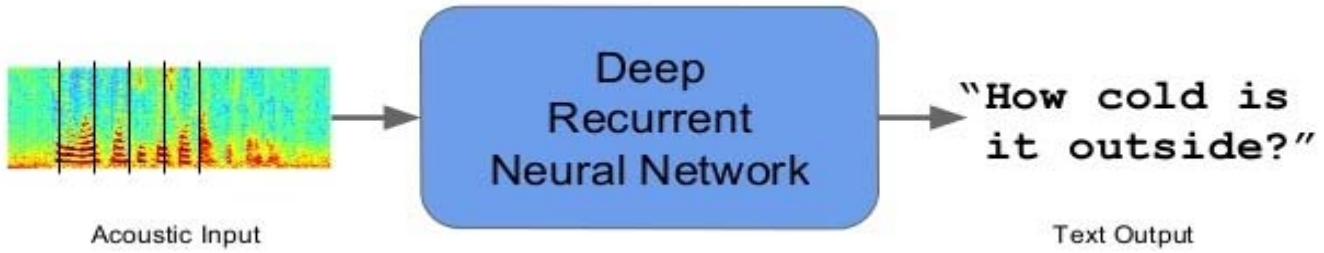
Hey Siri

IBM Watson Analytics









Strengths of Neural Networks

- Neural networks can be used for many different purposes
  - Different data types (image, time series, text etc.)
  - Different problems and tasks (segmentation, classification, content generation)
  - Networks can be combined, can have multiple inputs and multiple outputs

- Neural networks are:
  - Extremely flexible
  - Powerful learning methodology
  - State-of-the-art in many domains today

AGENDA

Introduction to neural networks

Neurons: the perceptron, the sigmoid neuron

Layers, multi-layer perceptron (MLP)

Basic intuition of networks training

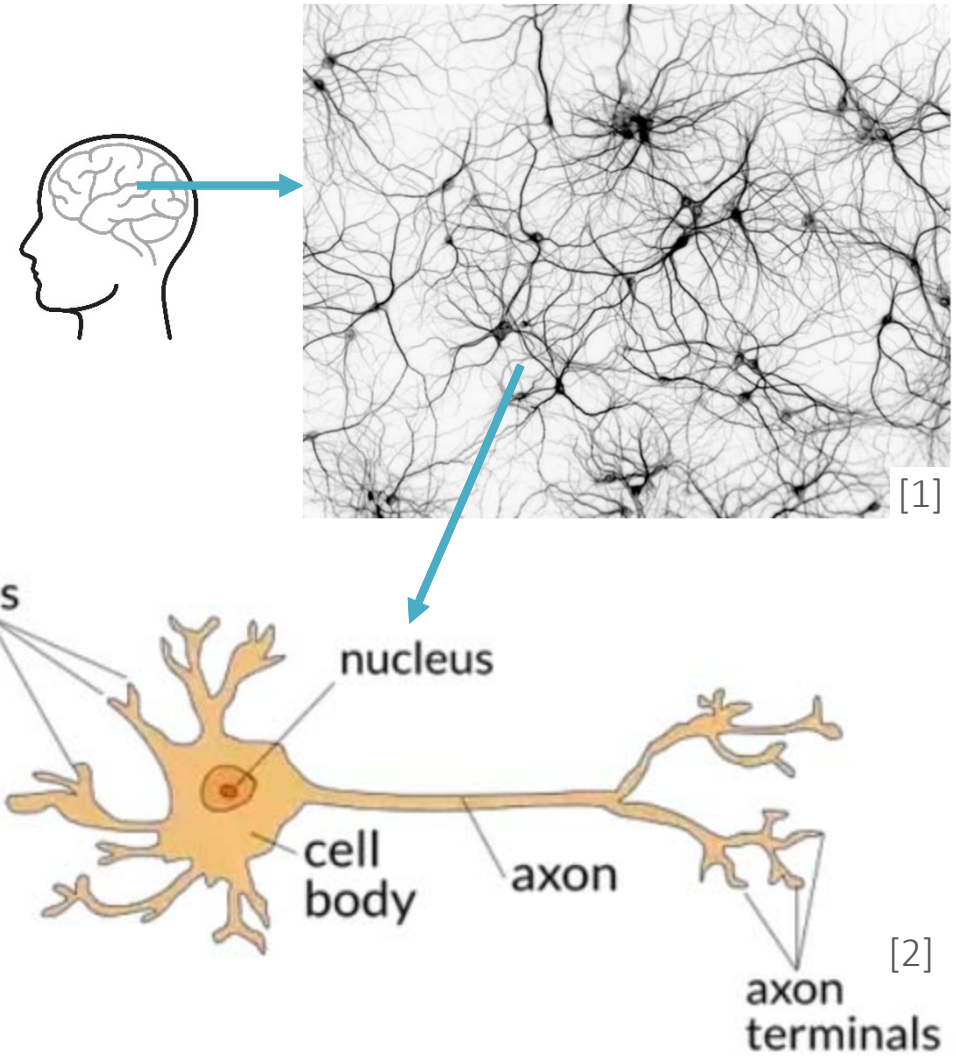
Example: Designing a network

Outlook

INTRODUCTION TO NEURAL NETWORKS

Neural Networks and the Human Brain

- Neural network = machine learning method (among many others)
- *Biologically-inspired* programming paradigm for learning from *data*
- Model = human brain
  - Extremely powerful and flexible
  - Approx. 10^{11} neurons
  - Processing power: 20Mbit/s (optical nerve)
- Neural network = *coarse* approximation

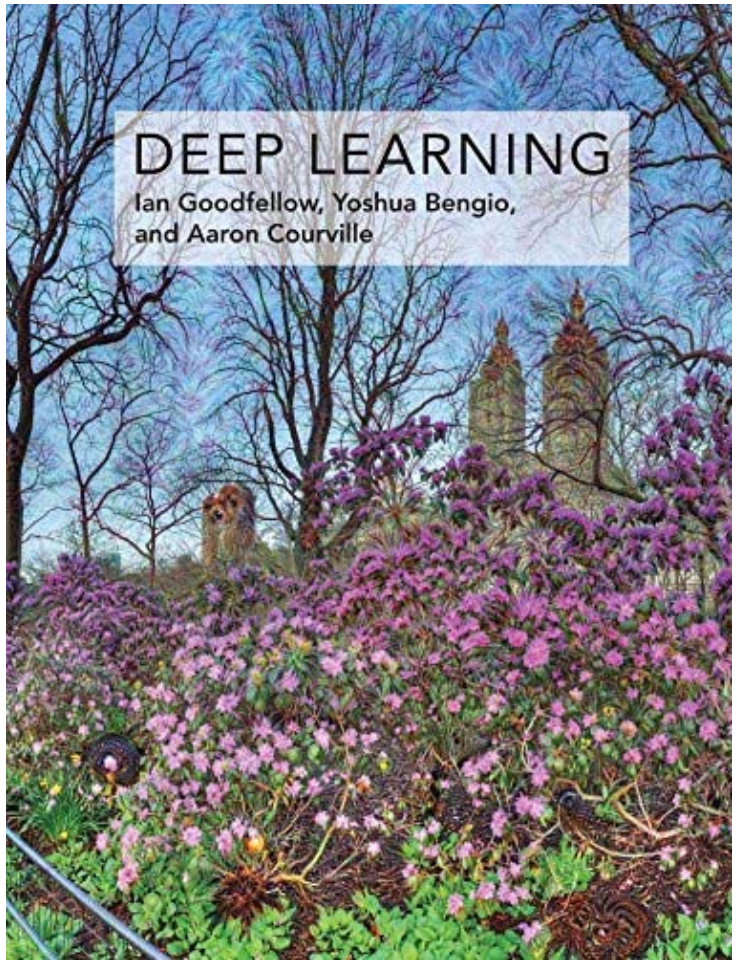


[1]

[2]

“Deep Learning”

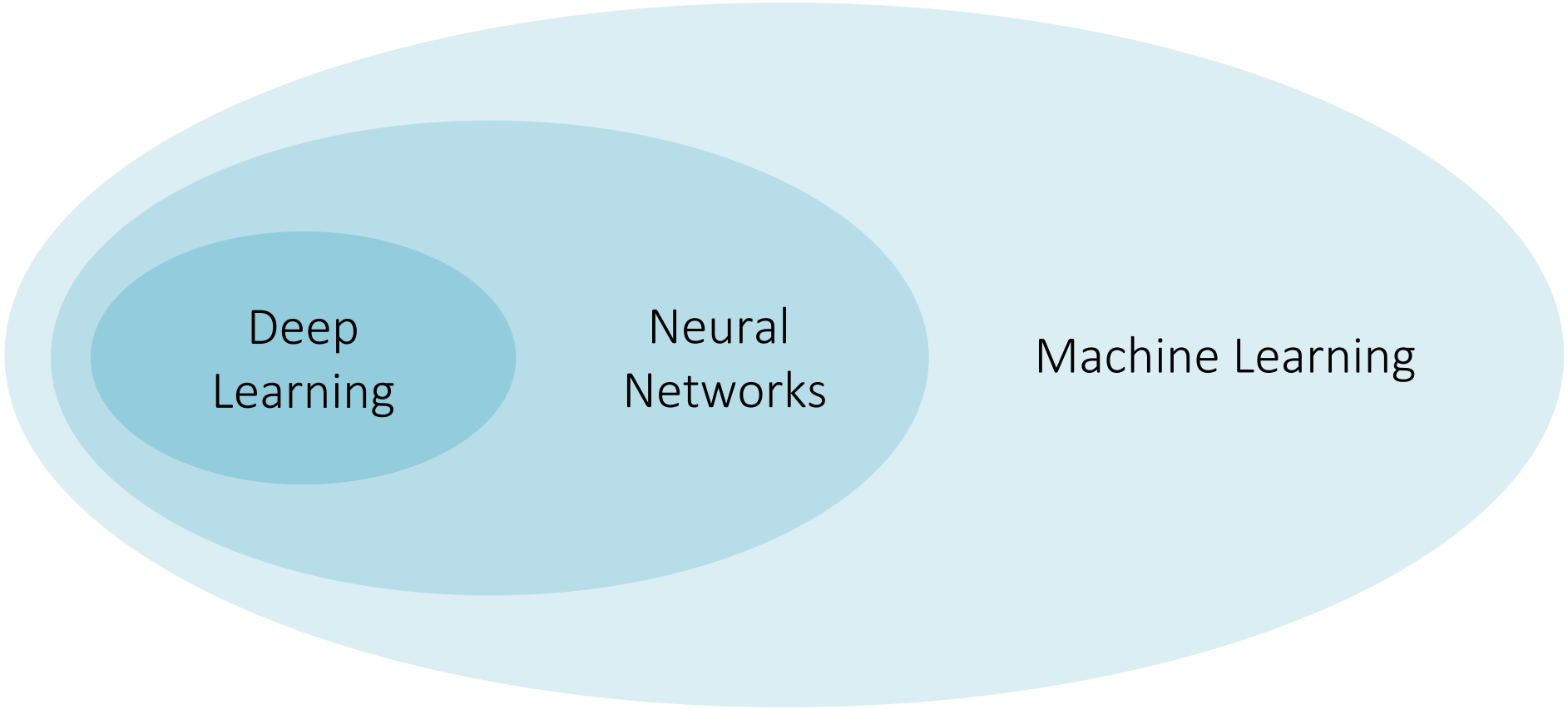
- “Deep learning”: a powerful set of techniques for learning complex neural networks, i.e. with multiple layers
- Deep learning = current state-of-the-art in
  - Image recognition,
  - Speech recognition,
  - Natural language processing,
  - ...
- A network is considered deep when it has more than one *hidden layer*



Goodfellow et al., “Deep Learning”, MIT Press 2016
http://www.deeplearningbook.org

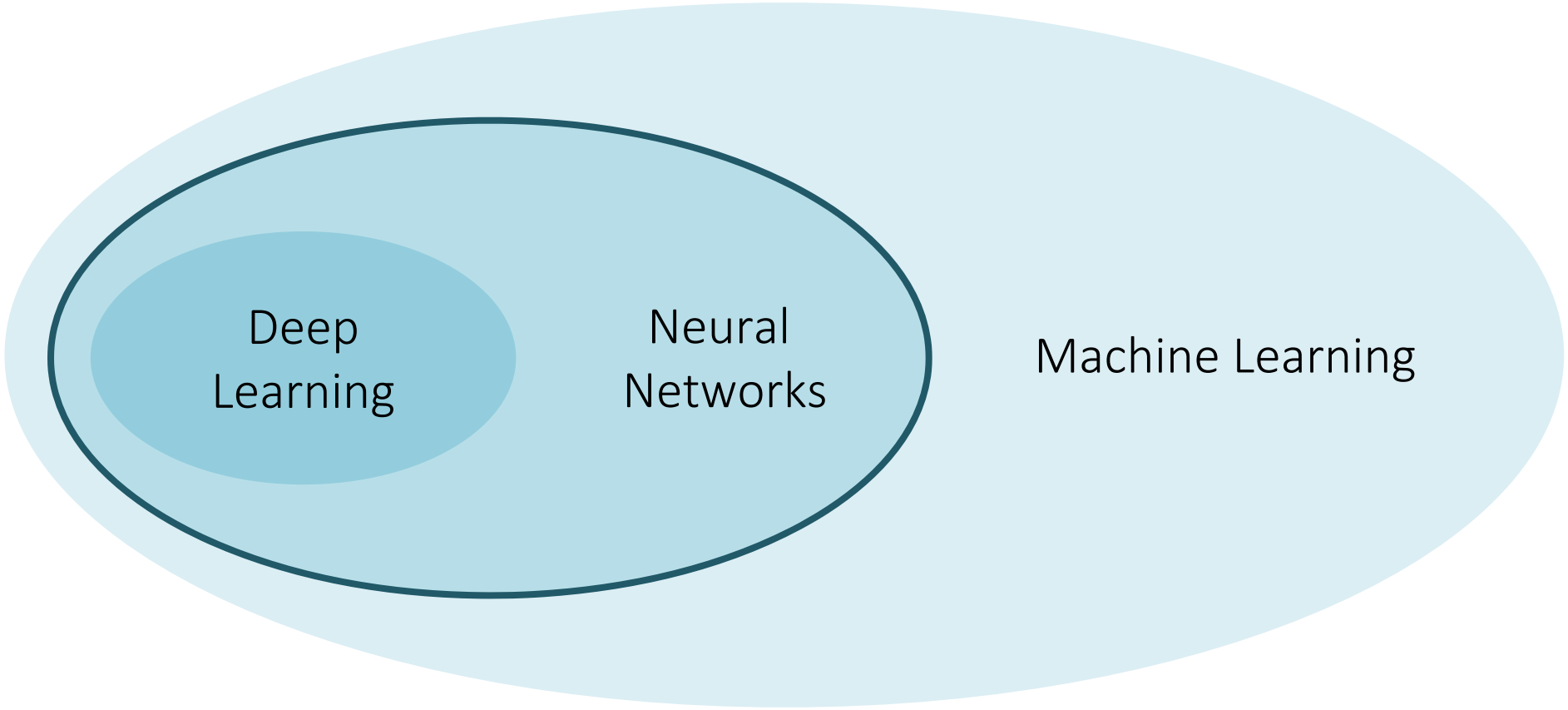
Methodological Categorization

Machine learning vs. neural networks vs. deep learning



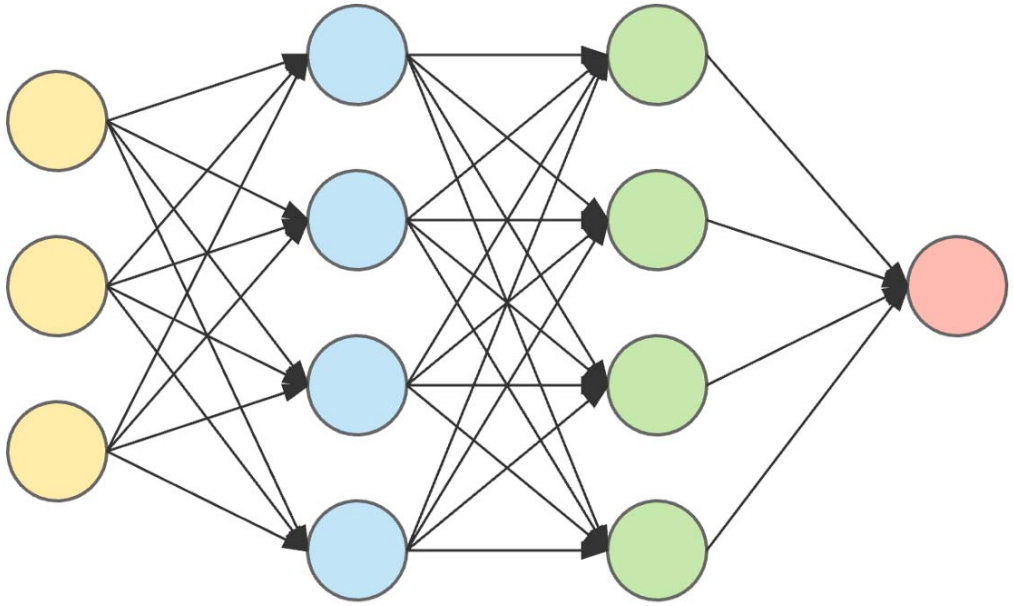
Methodological Categorization

Machine learning vs. neural networks vs. deep learning



Neural Networks

- Basic building blocks:
  - Individual neurons
  - Connections between neurons
  - Layers (groups of neurons)
- Formally: map input x to output y by a function f that represents our neural network

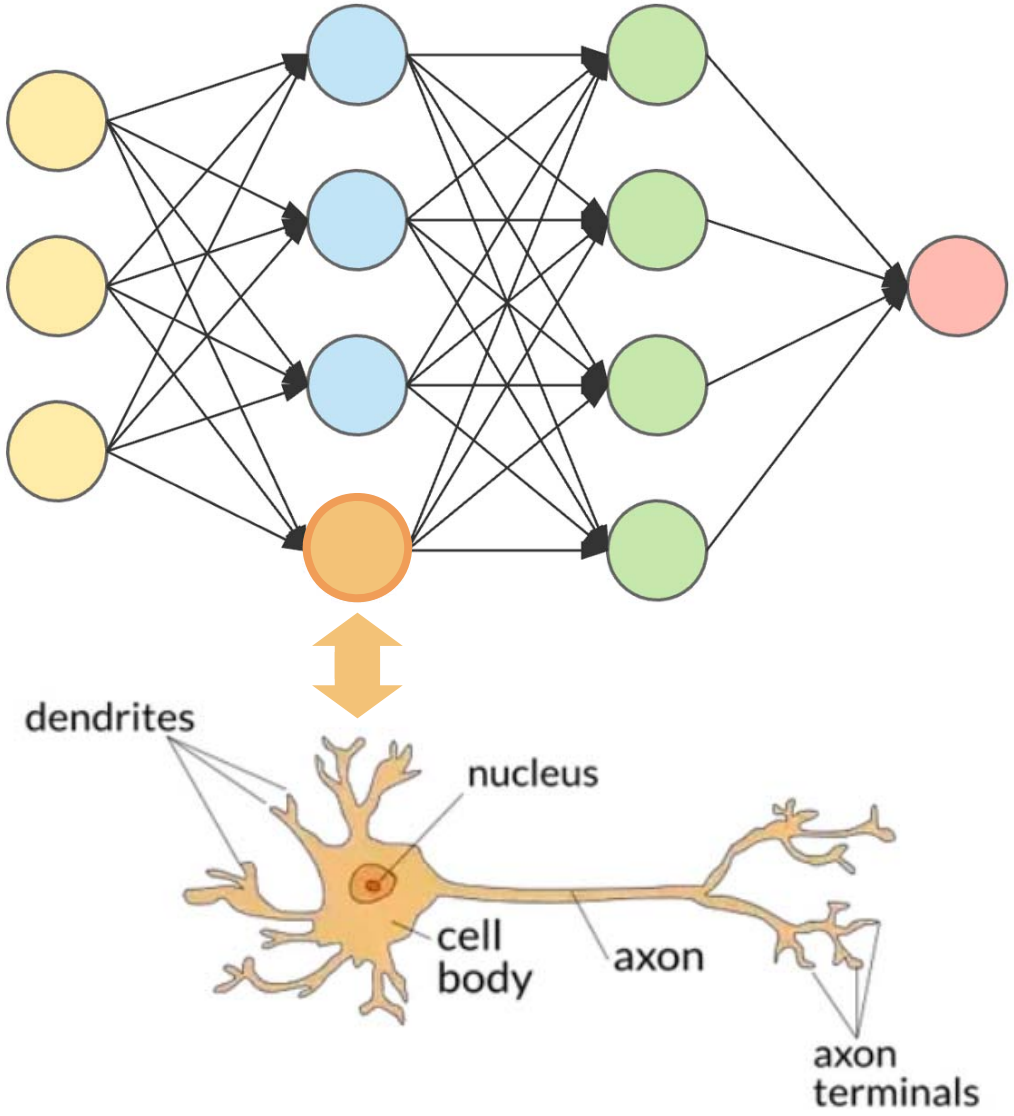


x f y

$$y = f(x)$$

Building Blocks of Networks: the Neuron

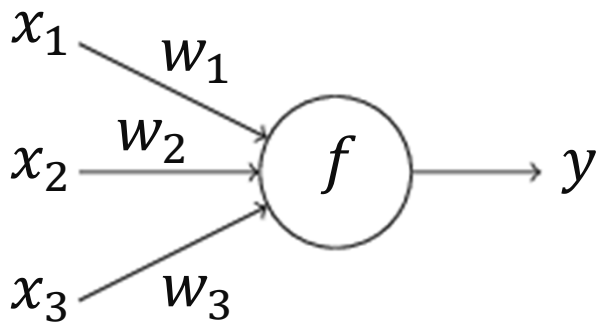
- A network is built of *artificial neurons*
- Different models of neurons exist (e.g. the *perceptron*)
- Much simpler than the biological equivalent in our brain
- Simplest networks: all neurons structurally the same



The Perceptron

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

- Developed by Frank Rosenblatt in 1950s*
- Early artificial neuron model



- A perceptron takes several inputs, x_1,x_2,…, and produces a single binary output y (0|1)
- Weights, w_1, w_2,…, (real numbers) express the importance of the respective inputs
- Neuron function f:

$$y = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

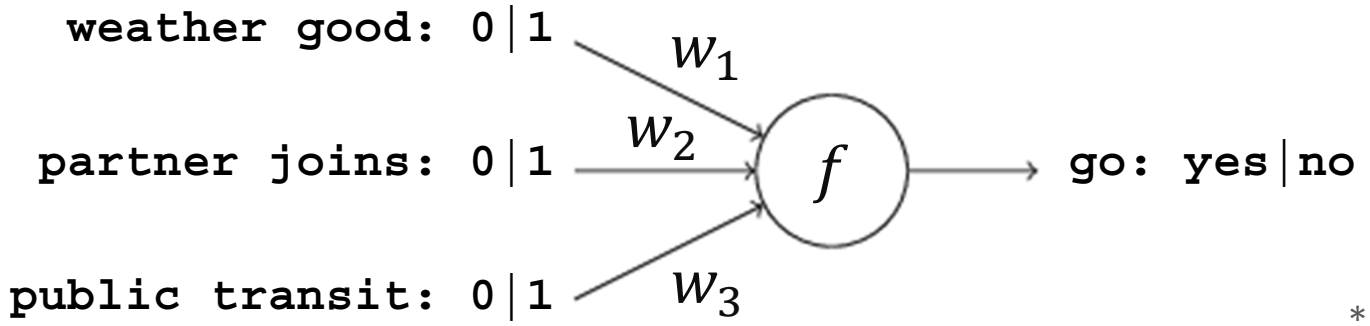
* The perceptron - a probabilistic model for information storage and organization in the brain. F. Rosenblatt. *Psychological Review* 65, 1958.

The Perceptron…

… can be considered a decision maker

Shall I go to the Cheese Festival?*

- The weekend is coming up, and you've heard that there's going to be a cheese festival in your city. You like cheese, and are trying to decide whether or not to go to the festival.
- You might make your decision by weighing up three factors:
  1. Is the weather good? Yes | No
  2. Does your partner want to accompany you? Yes | No
  3. Is the festival near public transit (you don't own a car)? Yes | No
- Build your own perceptron that helps you with the decision making:



How to set the weights?

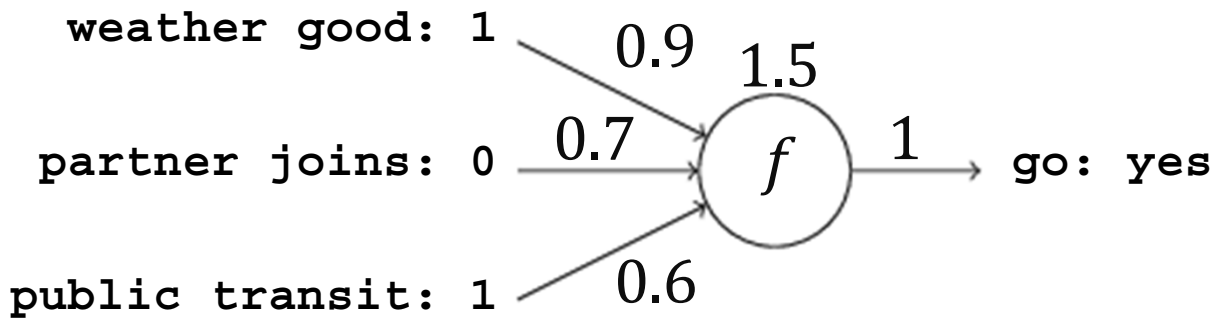
Choose your preference..

* http://neuralnetworksanddeeplearning.com/chap1.html

Shall I go to the Cheese Festival?*

- Example
  - I hate rain: $w_1 = 0.9$
  - I prefer accompany: $w_2 = 0.7$
  - I am lazy: $w_3 = 0.6$
  - $\text{threshold} = 1$
- Given:

$w_j = 0$ means I don't care



$$y = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$y = 1 * 0.9 + 0 * 0.7 + 1 * 0.6 = 1.5 \Rightarrow f(x) = 1$$

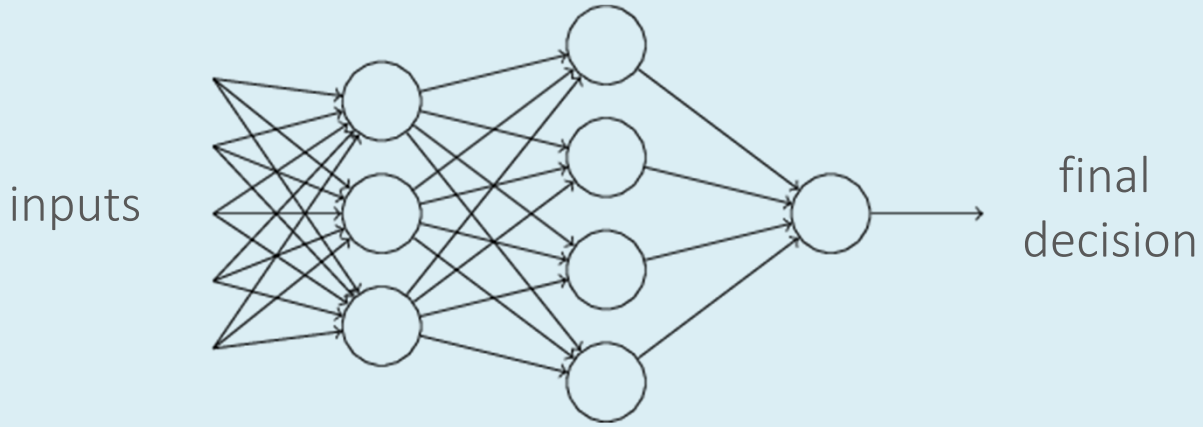
* http://neuralnetworksanddeeplearning.com/chap1.html

This was a Simple Decision..

Example: whole group of people wants to jointly decide to go to the cheese festival

… what about more complex decisions?

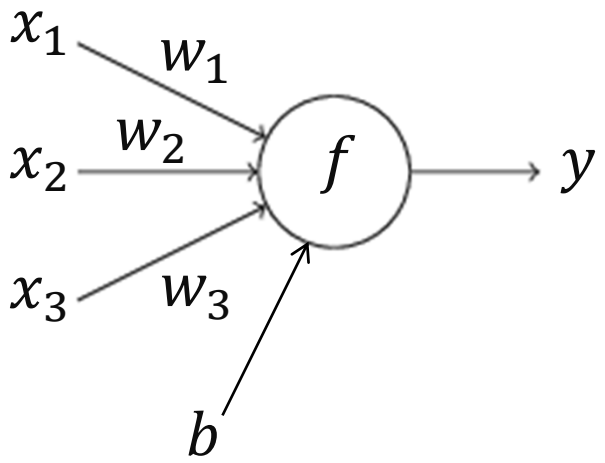
Connect multiple perceptrons to each other



Role of the Threshold

- Question: how to set the threshold?
- Recap:

$$y = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$



- Answer: learn it from data (same for weights)
- Replace sum by dot product and threshold by variable “b”:

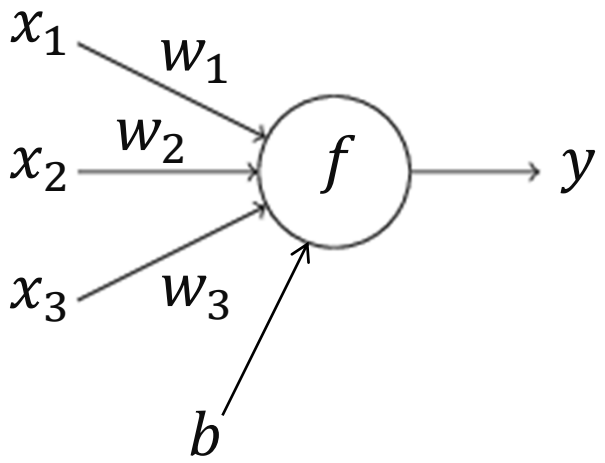
$$y = \begin{cases} 0 \text{ if } \mathrm{w} \cdot \mathrm{x} + b \leq 0 \\ 1 \text{ if } \mathrm{w} \cdot \mathrm{x} + b > 0 \end{cases}$$

common notation in the literature

- b is called “bias”, note: $b = -\text{threshold}$
- A large bias means: it’s likely that the neuron outputs a 1 (i.e. it “fires”)

The Perceptron as a Classifier

- Output: binary
- Decision function: linear



Perceptron is a binary linear classifier

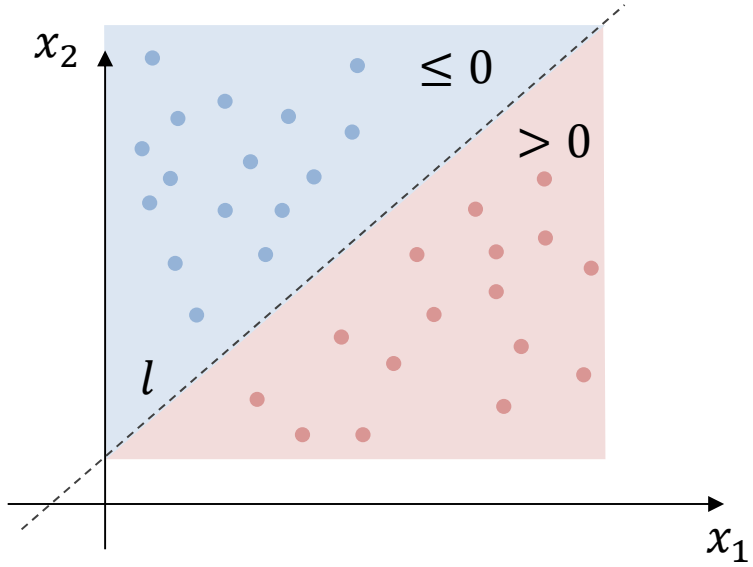
The Perceptron: Example in 2D

- Perceptron is a binary linear classifier
- Interpretation in 2D:
  - Input: 2 variables x_1 and x_2
  - 2 weights w_1 and w_2, bias b
  - Decision function f:

$$y = \begin{cases} 0 \text{ if } w_1 \cdot x_1 + w_2 \cdot x_2 + b \leq 0 \\ 1 \text{ if } w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0 \end{cases}$$

  - Equations describe 2 regions separated by line:

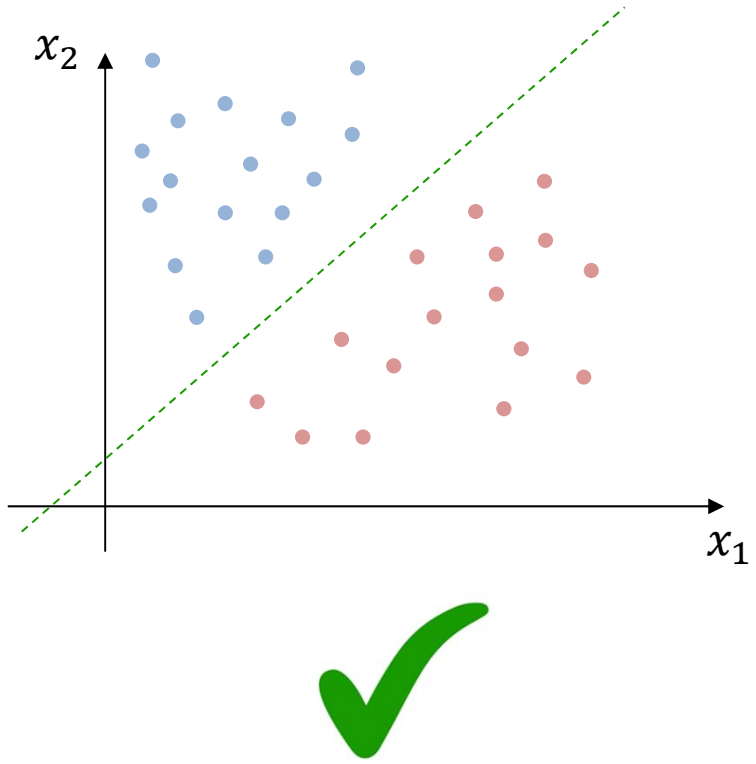
$$l\colon\ w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$$

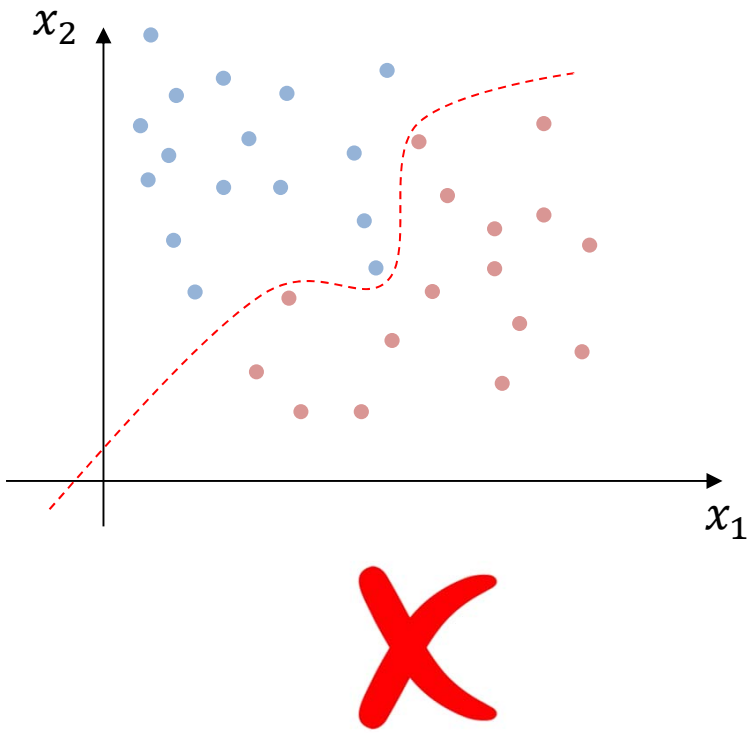


$w = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ is the normal vector of the line

Limitations of the Perceptron

› Perceptron can solve only linearly separable problems





The XOR Problem

- Perceptron can solve only linearly separable problems
- Simple logical operations:

1	0	1
0	0	0
AND	0	1

✔

1	1	1
0	0	1
OR	0	1

✔

1	1	0
0	1	0
NOT	0	1

✔

- The XOR Problem is not linearly separable:

1	1	0
0	0	1
XOR	0	1

✘

→ led to the first AI winter (1970s) →

[4]

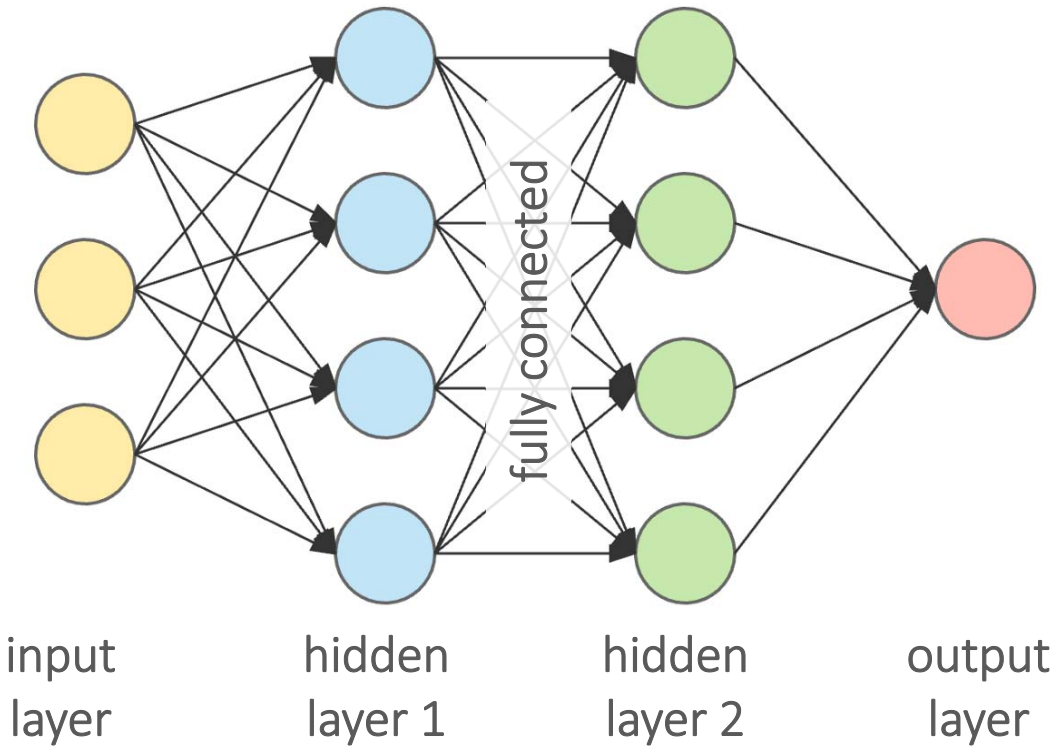
Minsky, M., & Papert, S. A. (1969). Perceptrons: An introduction to computational geometry. MIT press.

One Perceptron cannot do…

… but several can!

The Multi-Layer Perceptron

- Combine multiple perceptrons to a network
- Arrange perceptrons in layers
  - Input layer
  - Hidden layers
  - Output layer
- Connections between neurons
  - full connectedness → fully connected layers
  - Feed-forward architecture



The Multi-Layer Perceptron

- Good news:
  - Can solve the XOR Problem (by learning multiple linear decision boundaries)
  - MLP with more than one layer of perceptrons (with binary activation) can learn any Boolean function

- Bad news:
  - Perceptrons cannot learn non-linear relationships
  - Inefficient for training

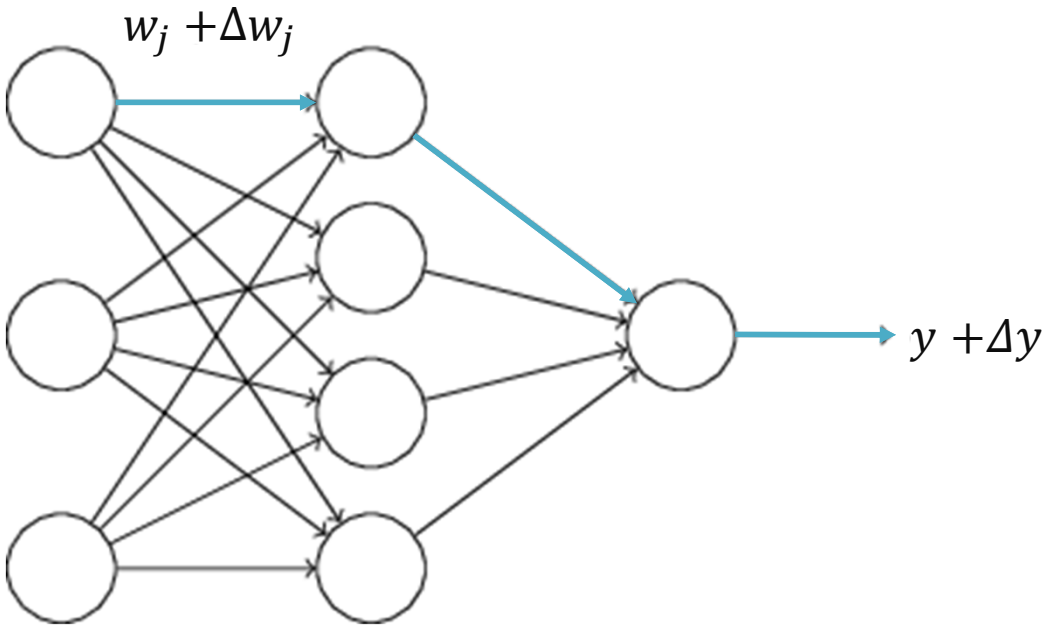
Strong limitation in practice!

What's wrong with the Perceptron?

- Consider we want to train an MLP
- Manually setting weights and biases = tedious!
  - Even simple MLPs easily have several thousands of weights and biases
- Better: estimate weights and biases from the data (and its labels)
- Basic idea:
  - Start with random values for weights and biases
  - Change weights and biases so that desired output is achieved
- Basic requirement: **continuity**

Pre-requisite for Learning

- Learning requires continuity (no “jumps”)
- Small change in learning parameter → small change in output



Not the case for the perceptron (output may flip from 0 to 1 abruptly), remember:

$$y = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

- Difficult to assess if change by Δw_j is beneficial
- Danger to “forget” already learned patterns

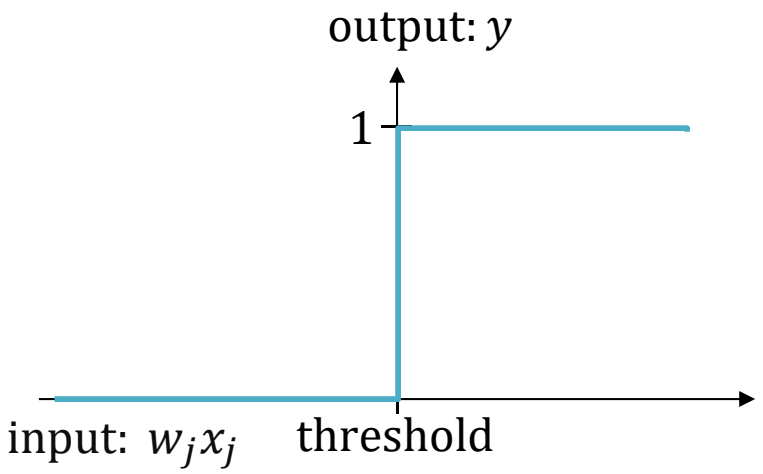
What’s wrong with the Perceptron?

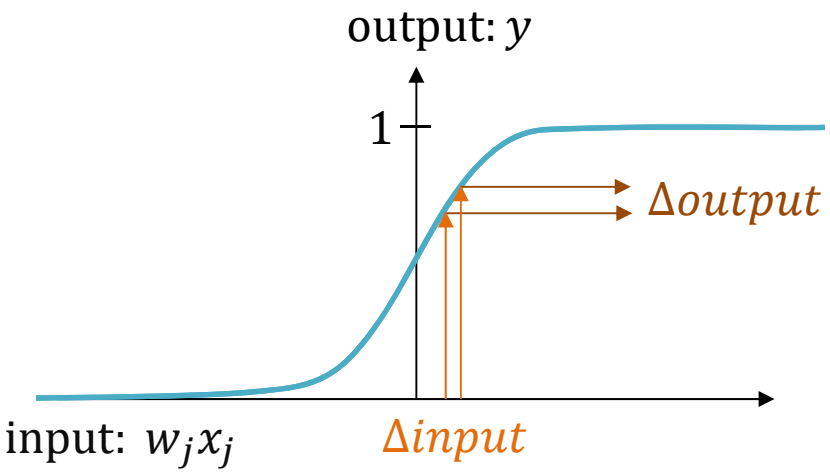
The binary output!

We need a neuron with continuous output!

Smooth the Activation Function

› Replace step-function of perceptron with a continuous “step-like” function





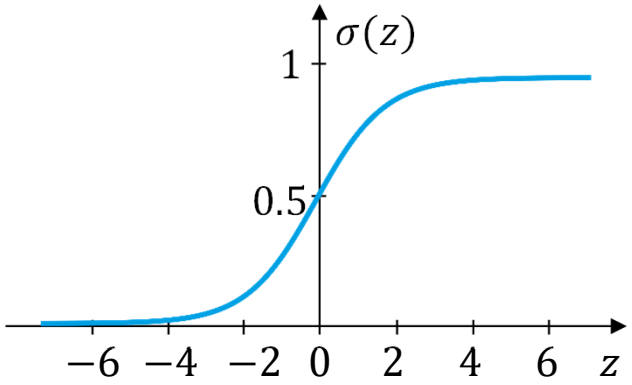
$$y = \begin{cases} 0 \text{ if } \sum_j w_j x_j \leq \text{threshold} \\ 1 \text{ if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Small change in input leads to small change in output!

The Sigmoid Neuron

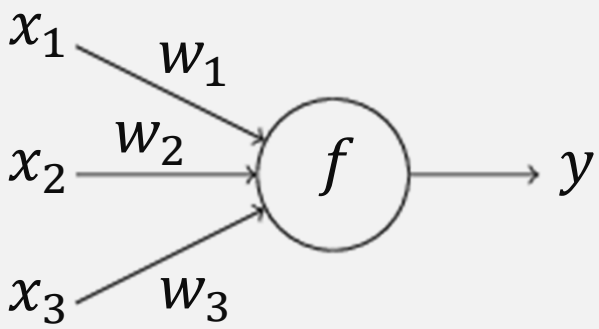
- The sigmoid function:

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



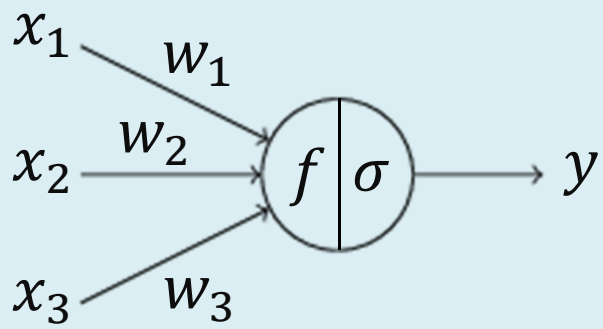
- Scales inputs to *0..1*
- For large inputs: same as step function of perceptron
- For moderate inputs: non-linear scaling

Perceptron



$$y = \begin{cases} 0 \text{ if } \mathrm{w} \cdot \mathrm{x} + b \leq 0 \\ 1 \text{ if } \mathrm{w} \cdot \mathrm{x} + b > 0 \end{cases}$$

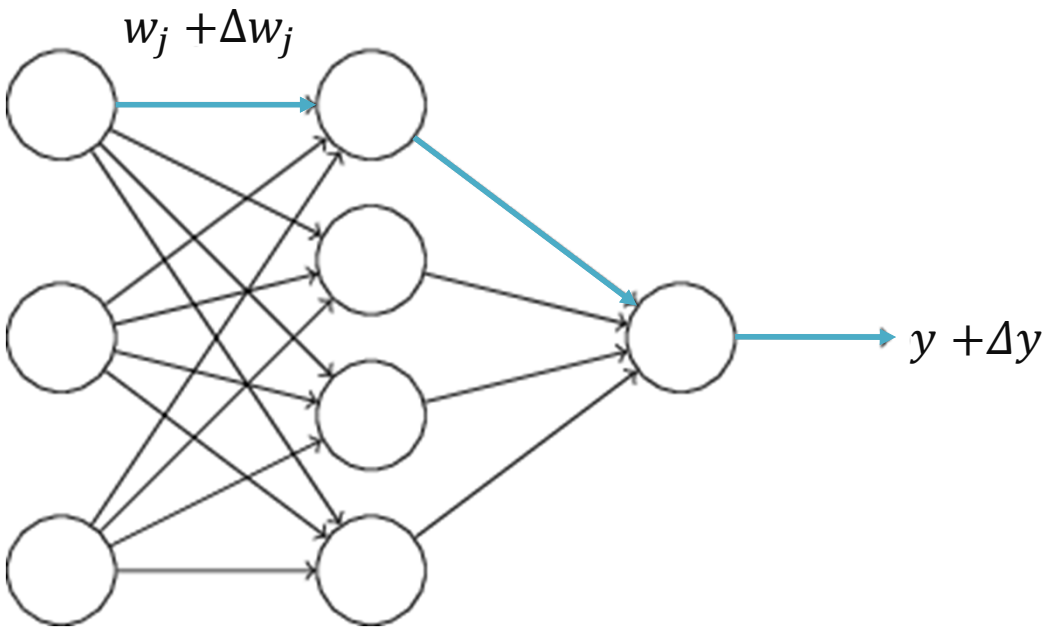
Sigmoid Neuron



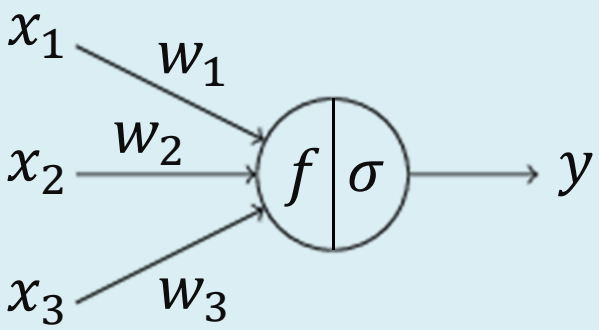
$$y = \sigma(\mathrm{w} \cdot \mathrm{x} + b)$$

Learning with Sigmoid Neurons

› Sigmoid is smooth: small change in parameter → small change in output



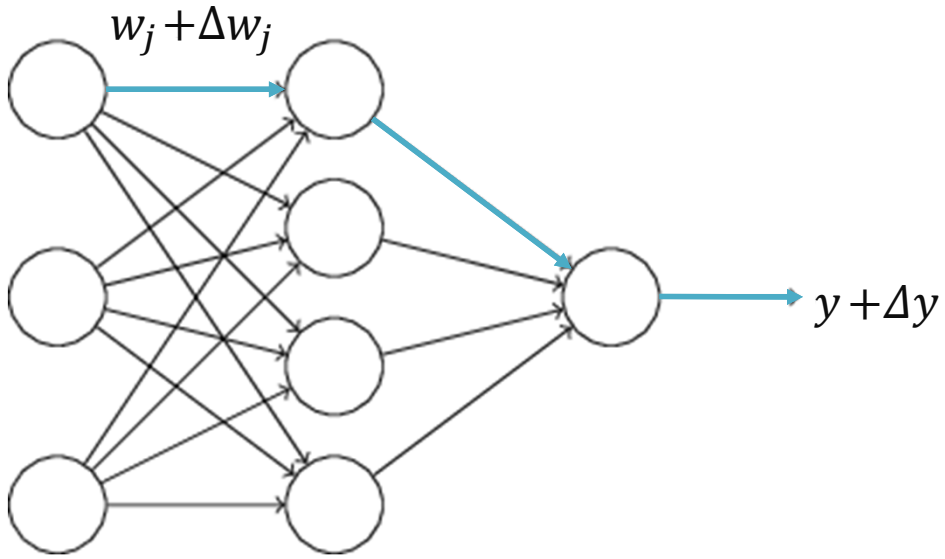
Sigmoid Neuron



$$y = \sigma(\mathrm{w} \cdot \mathrm{x} + b)$$

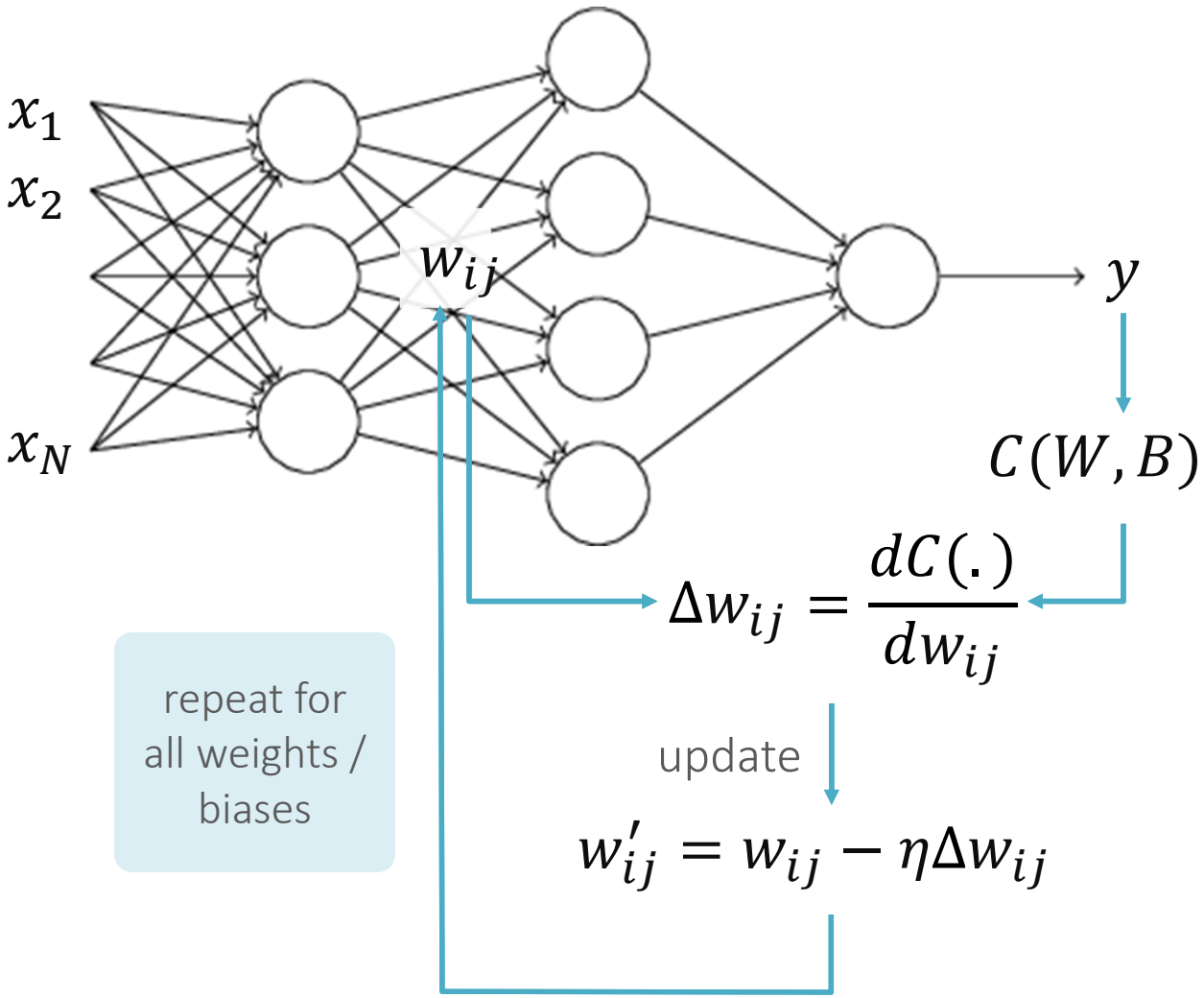
Learning with Sigmoid Neurons

- Learn from data $X = \{x^{(1)}, x^{(2)}, \dots, x^{(N)}\}$ to predict target Y given by labels $\{\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(N)}\}$
- Intuition behind learning
  - Start with random weights & biases
  - Adjust weights by small Δw so that outputs $y^{(i)}$ of the network gets closer to desired outputs $\hat{y}^{(i)}$
- How to adjust the weights to get closer to the desired output?



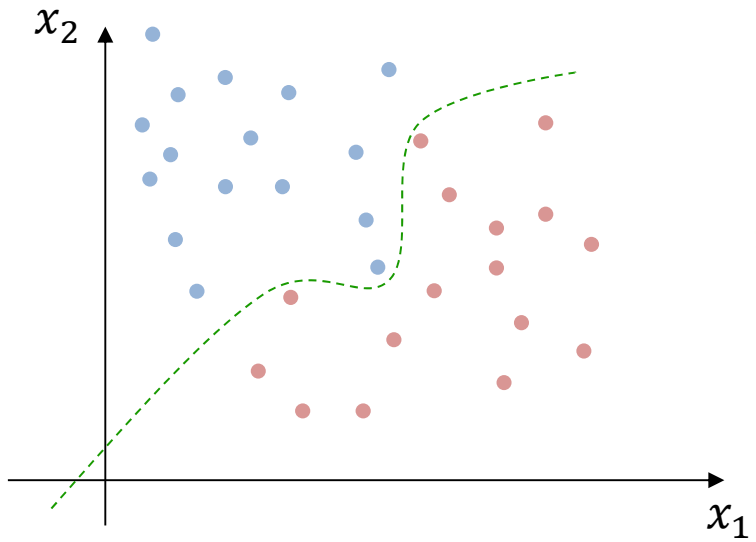
The Backpropagation Algorithm – An Intuition

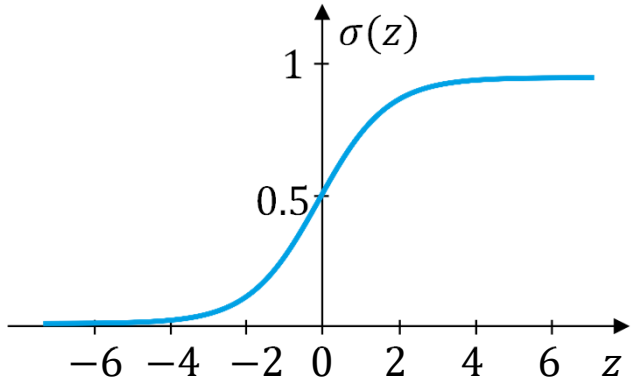
- Estimate output quality by a cost function:
  $C(W,B) = \frac{1}{2N}\sum_i \|y - \hat{y}\|$
- Goal: find Δw that decreases cost function
- Backpropagation algorithm provides an efficient solution for this.
  - Idea: estimate influence of parameter w_{ij} on output (=partial derivative w.r.t. cost function)
  - This tells us, how to change parameter w_{ij} to get closer to y' (and thereby to minimize C)
  - η is the learning rate



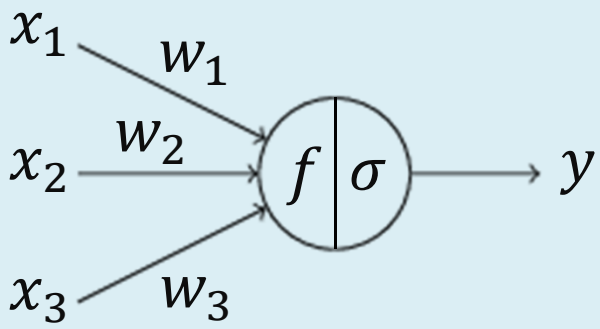
Making Non-Linear Decision

- The sigmoid neuron is not only smooth, it is also non-linear → allows to model non-linear relationships
- MLP based on sigmoid neuron is a non-linear classifier which can approximate nonlinear decision boundaries!





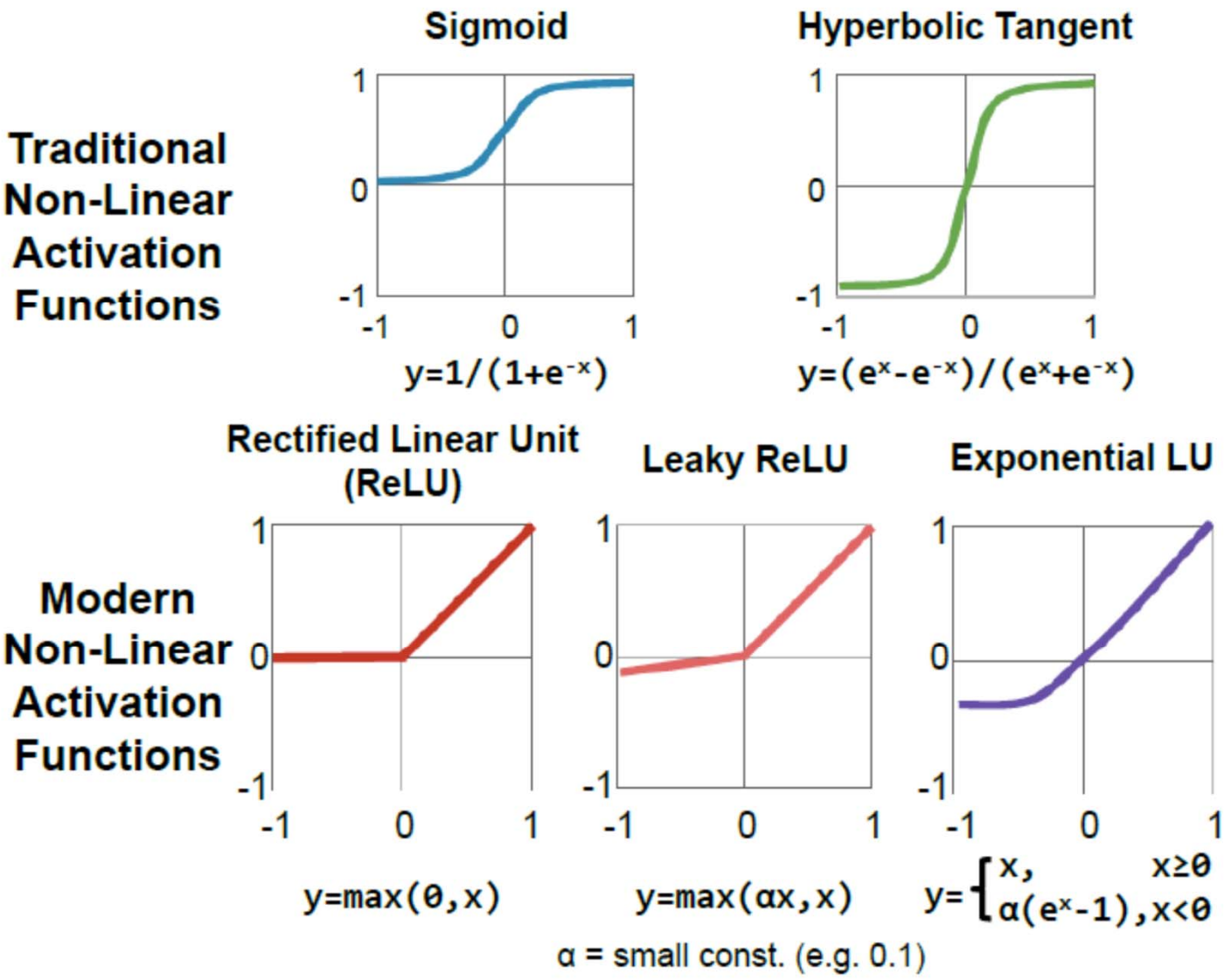
Sigmoid Neuron



$$y = \sigma(\mathrm{w} \cdot \mathrm{x} + b)$$

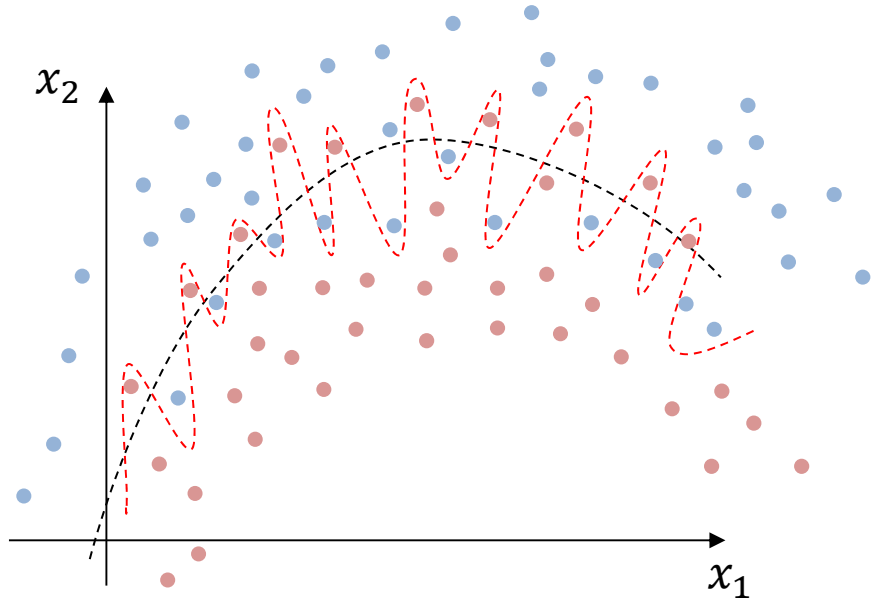
Activation Functions

- The sigmoid function in case of the sigmoid neuron is the “activation function” of the neuron
- Many other activation functions have been designed
- Important: they should be continuous and differentiable to facilitate learning!



Practical Considerations in Network Training

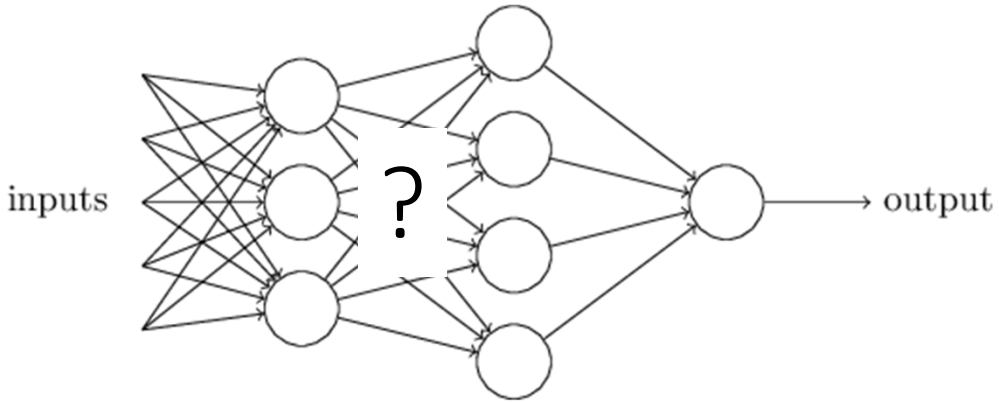
- MLP based on sigmoid neurons can model highly-complex data and decision boundaries
- Tendency to *overfit* on the data
- No guarantee that an optimal solution is found because the search space for finding a *good* function is huge!
- *Good* means a smooth function that fits the statistical *distribution* of the input data
- Data always limited: a neural network in only as good as its training data allows



Model individual points
→ poorly generalizes to new data

Designing a Neural Network

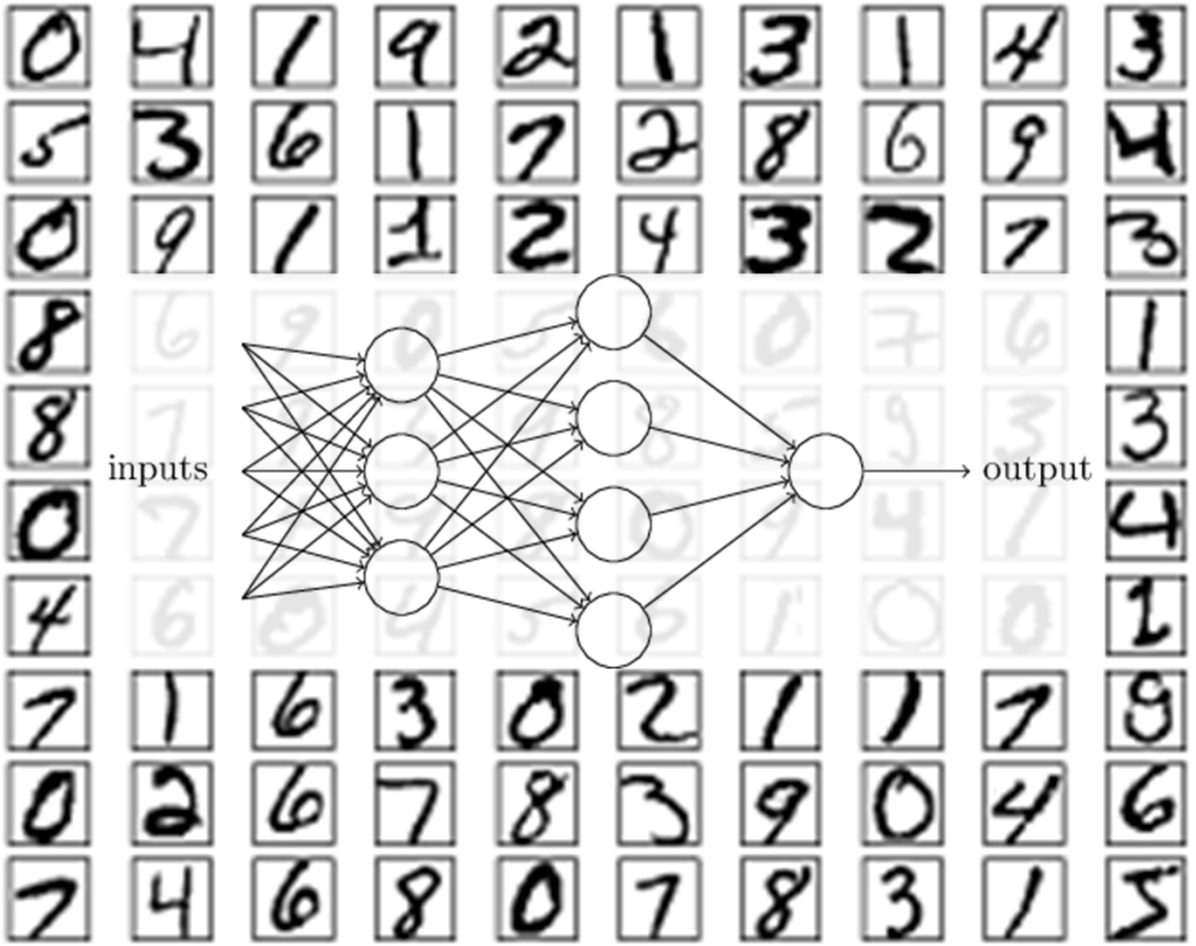
- Task: predict hand written digits
  - Input: image (size w x h), here w=28, h=28
  - Output: class (0,1,...,9)
- How could an MLP solution look like?



MNIST Dataset: http://yann.lecun.com/exdb/mnist/

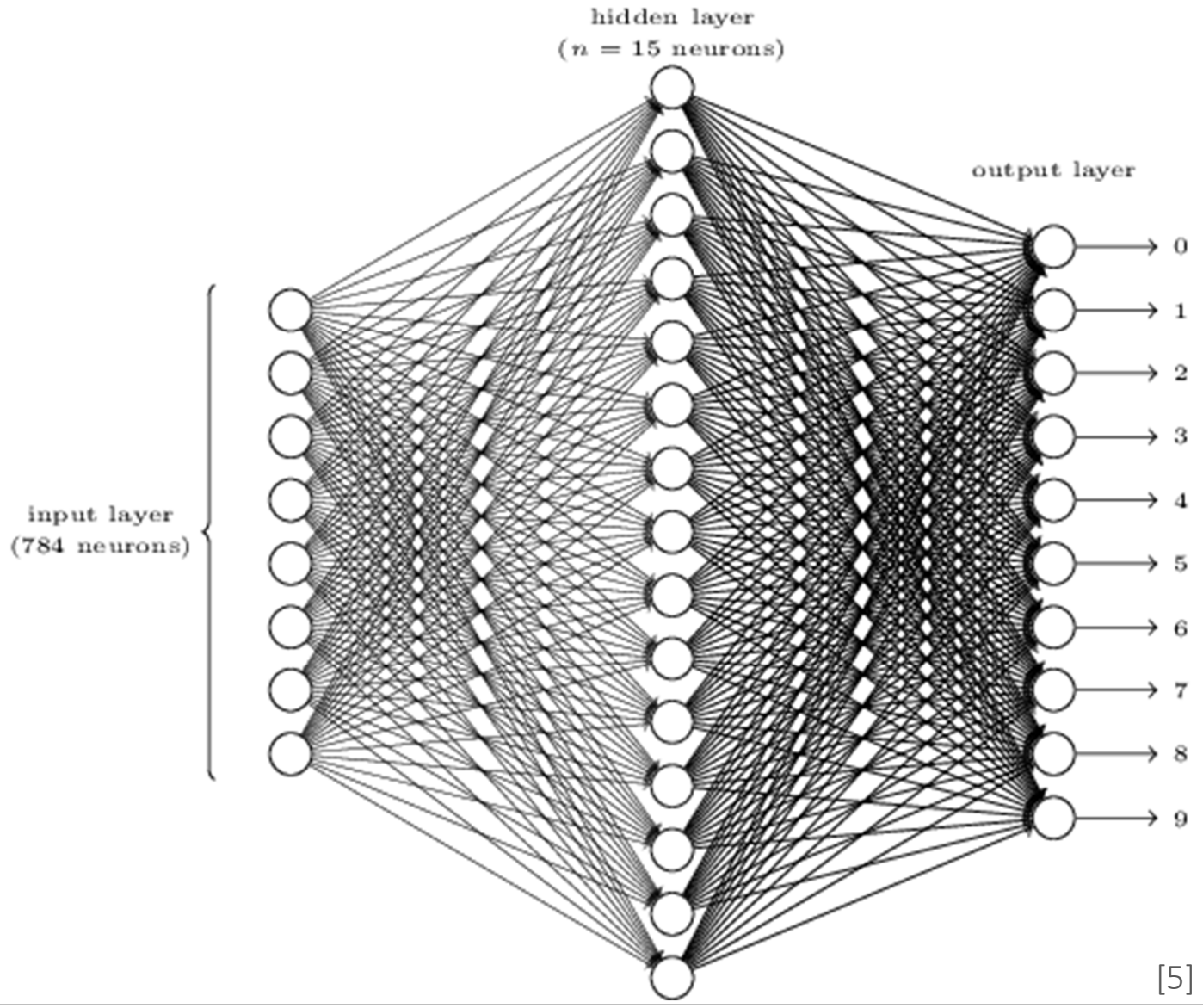
Designing a Neural Network

- Input layer:
  - w*h inputs (28x28=784)
  - Grayscale images: 1.0=white, 0.0=black
  - Input layer simply provides the raw data – no neurons
  - Here: inputs = raw image pixel values
- Output layer:
  - 10 Outputs: 0, 1, …, 9 (one per class)
- Hidden layers
  - How many layers? / neurons per layer?
  - Which connection pattern?



Designing a Neural Network

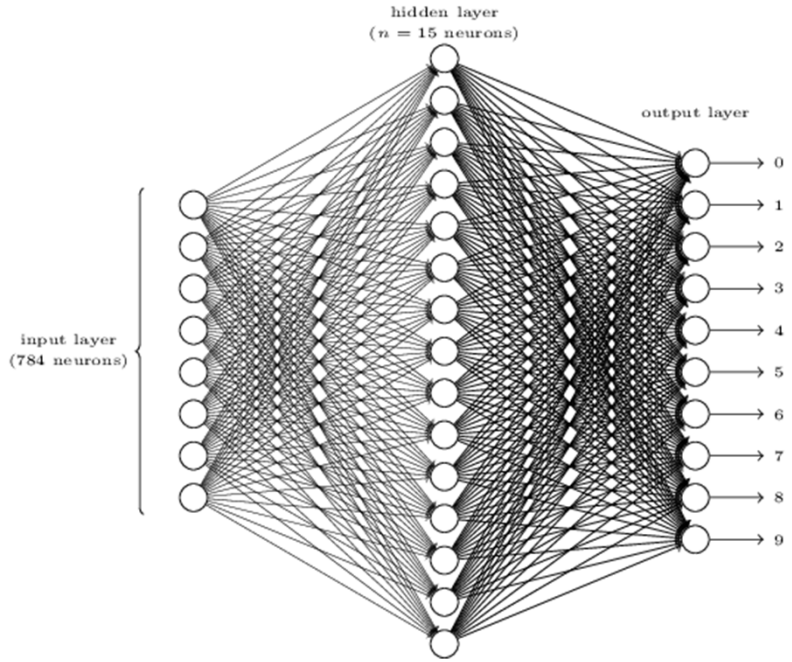
- Example:
  - 1 hidden layer
  - 15 neurons
  - Fully connected with input and output layer
- Number of parameters:
  - Weights: 784*15+15*10 = 11910
  - Biases: 15+10=25
- Result:
  - Accuracy > 90% on MNIST dataset



[5]

Design Considerations

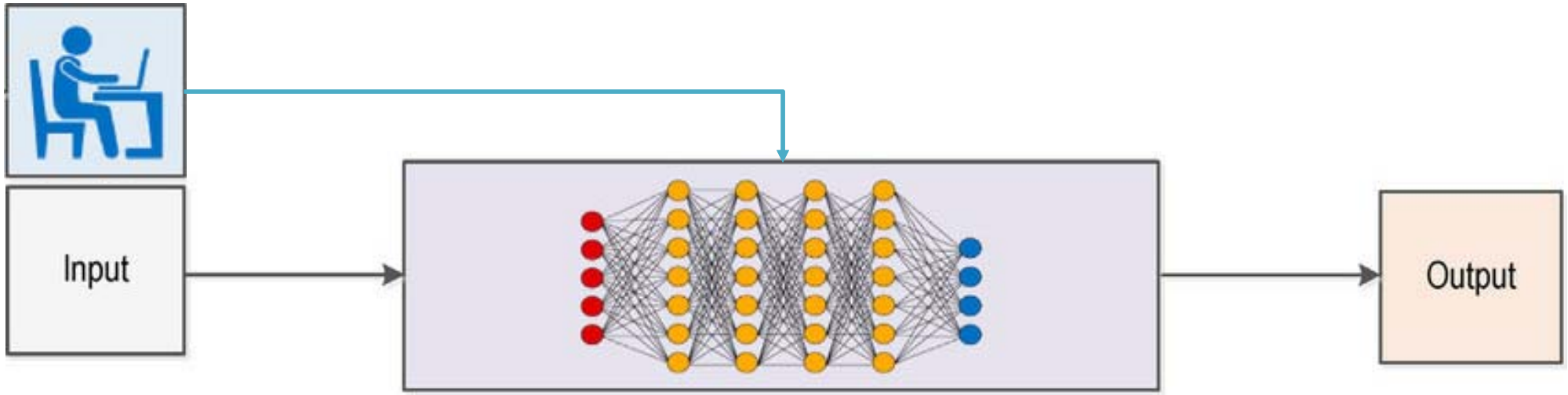
- Open questions:
  - Why 10 output neurons?
  - Why not 15 hidden neurons?
  - Why not 2 hidden layers
- Non-trivial problem, one of the most difficult problems with neural networks
- No rule of thumb for selecting the number of layers and the sizes of the layers (*hyper parameters*)
- Network design is one of the **core challenges** today!



Design Tradeoff

› Traditional design approach: *hand-crafted features*

› Neural network design approach: *hand crafted-networks*



OUTLOOK

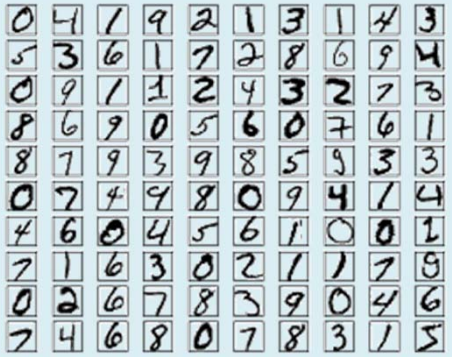
Limitations of MLP

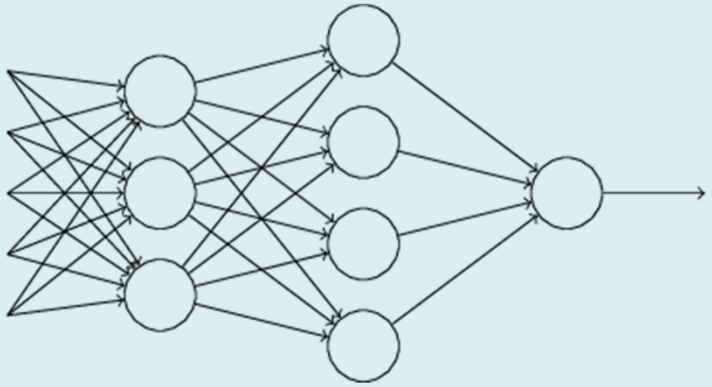
Further network architectures

Limitations of MLP

- MLP architecture
  - Good for structured data (e.g. demographic data – predict income)
  - Suboptimal for unstructured data: images, videos, audio/time series

MLP





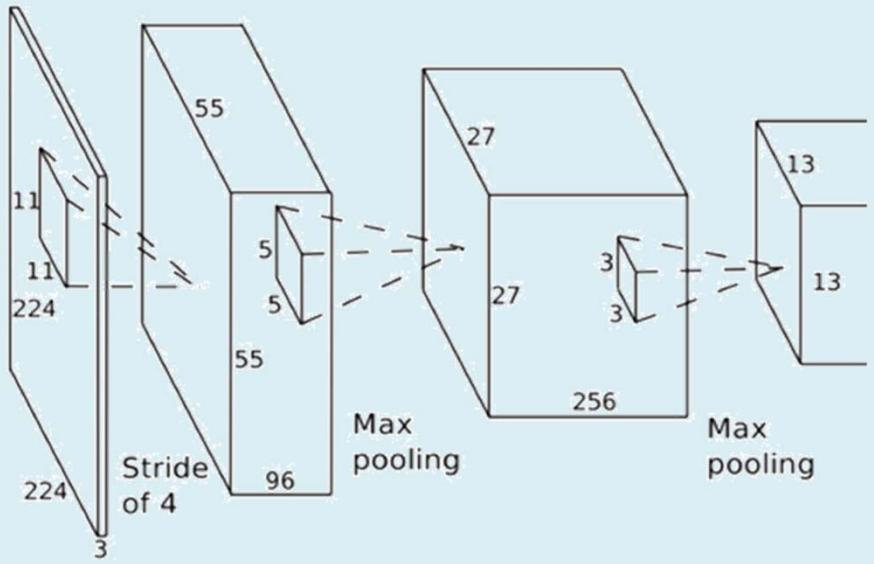
Independent inputs

No spatial/temporal relations

Scales badly with input size

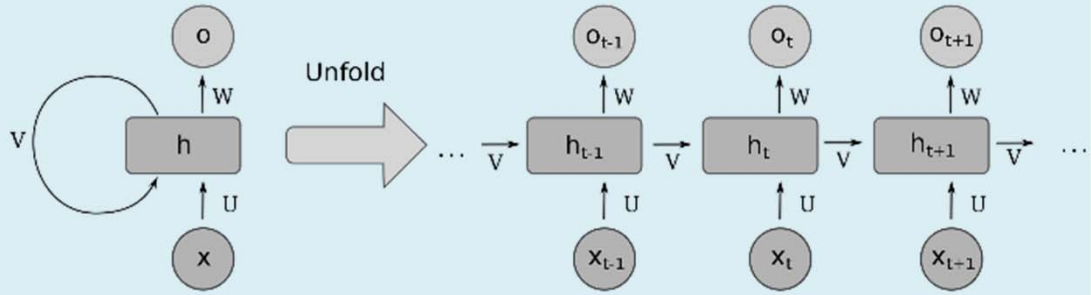
Outlook: More Powerful Network Architectures

Convolutional Neural Networks (CNNs)



Spatial data (e.g. image, video)

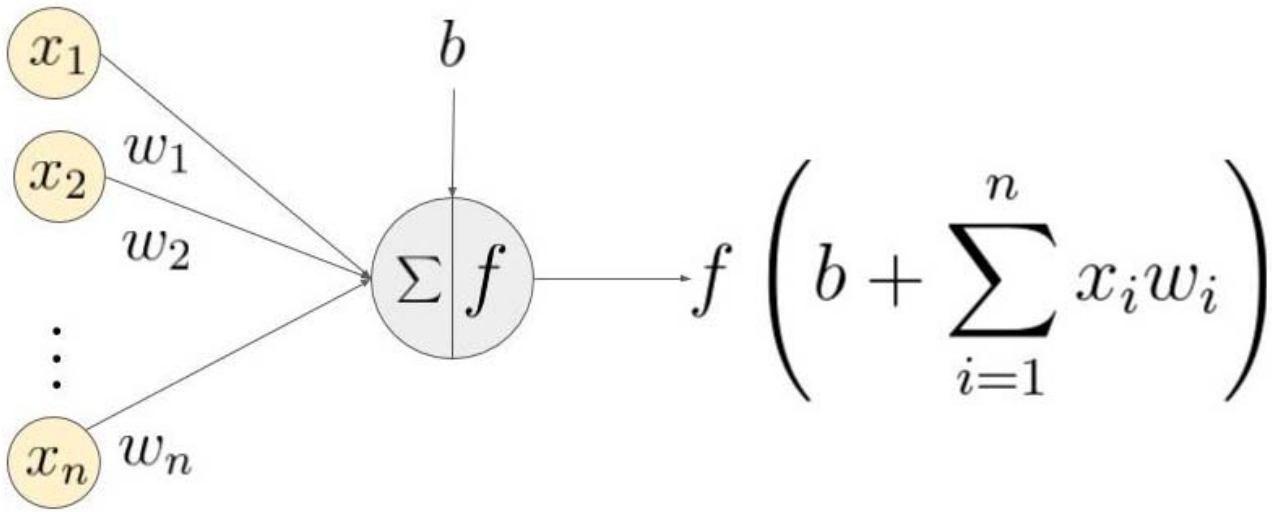
Recurrent Neural Networks (RNNs)



Temporal data (e.g. audio, time series)

Conclusion

- Introduction to neural networks
- Neurons: the perceptron, the sigmoid neuron
- Basic intuition of networks training
- Layers, multi-layer perceptron (MLP)
- Example: Designing a network
- Outlook



[6]

Contact:
matthias.zeppelzauer@fhstp.ac.at

Credits & References

[1] https://fractalfoundation.org/OFC/OFC-1-6.html

[2] https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[3] https://www.sciencedirect.com/science/article/abs/pii/S0278612518300037

[4] https://pixabay.com/es/users/cherebus0-1373501/

[5] http://neuralnetworksanddeeplearning.com/chap1.html

[6] image credit: Aditya Sharma