

Introduction to Artificial Neural Networks

Pattern recognition

5.3. 2019

Outline

- ▶ Biological neural networks
 - ▶ Motivation
 - ▶ Biological neurons
- ▶ Artificial neural networks
 - ▶ Symbolism vs. Connectionism
 - ▶ Historical background & bio-inspiration
 - ▶ First artificial neurons
 - ▶ XOR problem
 - ▶ MLP
 - ▶ Back-propagation
 - ▶ Types of ANN and learning

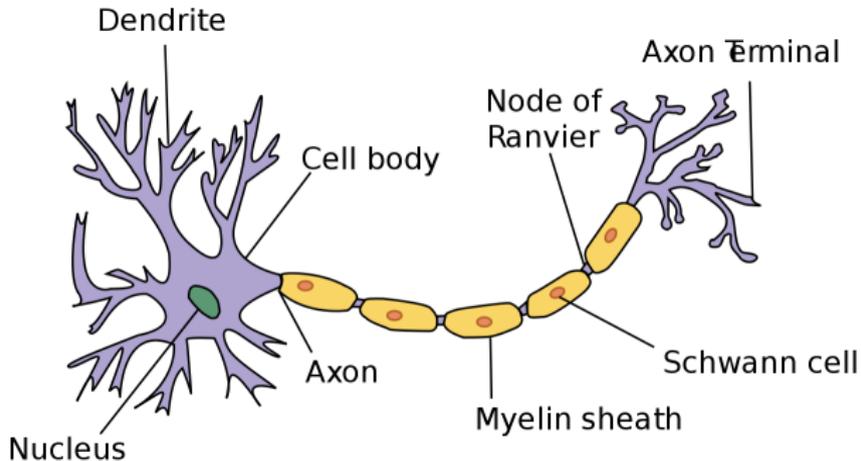
Biological Neural Networks

Why bio-inspiration

- ▶ Success in many other fields
 - ▶ basically, “nature” had some time to develop some pretty good systems (\sim evolution)
- ▶ Human brain:
 - ▶ neurons are slow compared to logic gates/transistors (order of milliseconds)
 - ▶ yet the brain is faster and more accurate on some tasks
 - ▶ massive parallelism
 - ▶ $\sim 10^{11}$ neurons
 - ▶ $\sim 10^{15}$ synapses
 - ▶ robustness & good generalization

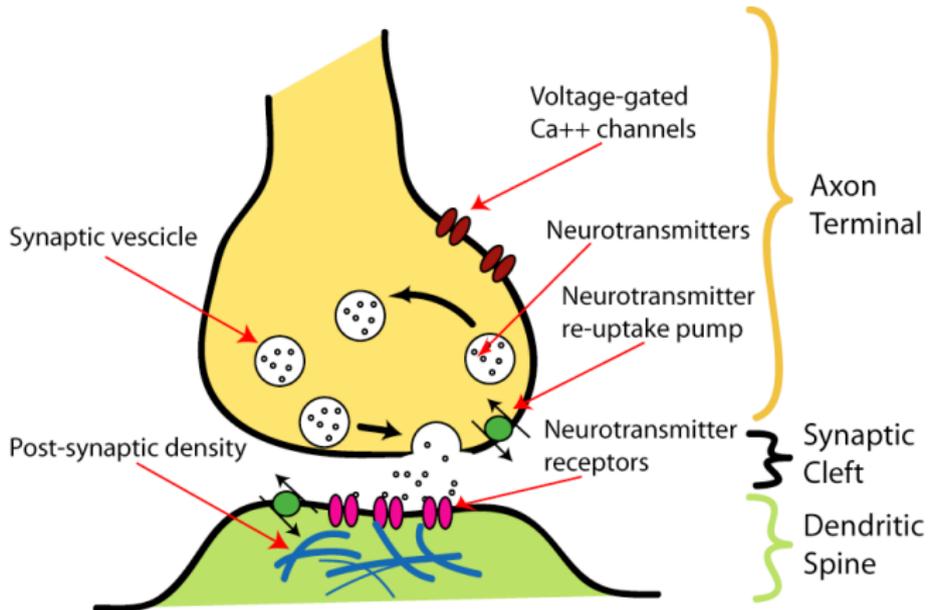
Biological neuron

- ▶ *Dendrites* collect information (electro-chemical signals)
- ▶ *Soma* (“body”) integrates information
- ▶ If threshold is reached → the neuron fires
 - ▶ Information is transmitted mostly via frequency modulation
- ▶ Firing transmitted via *axon*



Biological neuron – synapse

- ▶ Mediates the interaction between neurons
- ▶ The signal is converted from electrical activity to chemical (release of neurotransmitter) and back to electrical (activation of neurotransmitter receptors)
- ▶ Can have excitatory or inhibitory effect



Artificial Neural Networks

Connectionism

- ▶ Information processing theory
- ▶ **Symbolism:**
 - ▶ Entities in the world are represented using symbols
 - ▶ Information processing done by manipulating the symbols
 - ▶ E.g., propositional logic
 - ▶ View of human brain:
 - ▶ Humans are symbolic species – the symbolic part distinguishes us from the rest of the animals
 - ▶ Brain runs “a program” – the substrate is irrelevant
 - ▶ AI started as a symbolic field (many actual AI systems deployed in the field are symbolic even today)
- ▶ **Connectionism:**
 - ▶ Inspired by (human) brain – huge network of simple processing units
 - ▶ Information processing done via interaction between the units
 - ▶ View of human brain:
 - ▶ The form of human brain is important
 - ▶ Function of the brain cannot be understood without the connectionist view
 - ▶ Thanks to the most well known connectionist model – artificial neural network – connectionism is rapidly taking over the field of practical AI (e.g. search engines, recognition systems)

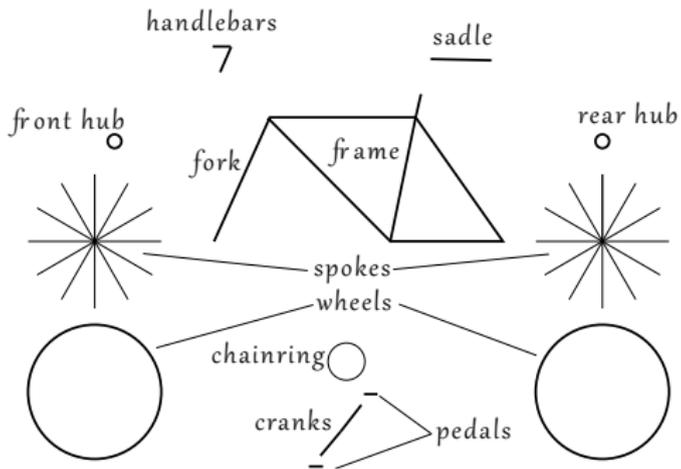
Connectionism vs. Symbolism

- ▶ *An oversimplified analogy*

What a symbolist sees:



What a connectionist sees:



History – first neural model

- ▶ Warren McCulloch and Walter Pitts (1943) – Threshold Logic Unit
- ▶ Binary firing function
- ▶ Excitation x^{exc} and inhibition x^{inh} inputs, firing threshold θ

$$\xi = \sum_{i=1}^n x_i^{exc} - \sum_{j=1}^m x_j^{inh} \quad (1)$$

$$y = \begin{cases} 1 & \xi \geq \theta \\ 0 & \xi < \theta \end{cases} \quad (2)$$

History – first neural model

- ▶ equation (1) can be rewritten with weights:

$$\xi = \sum_{i=1}^n x_i * w_i, w_i \in \{-1, 1\} \quad (3)$$

- ▶ equation (2) can then be changed to:

$$y = \begin{cases} 1 & \xi - \theta \geq 0 \\ 0 & \xi - \theta < 0 \end{cases} \quad (4)$$

- ▶ Putting it together and rearranging a bit – we get a spoiler of the future ANN model:

$$y = \text{step} \left(\sum_{i=1}^n x_i * w_i - \theta \right) \quad (5)$$

- ▶ There is still no learning
- ▶ Only simple binary functions

Hebbian learning rule

- ▶ The famous phrase:

“Neurons that fire together wire together”

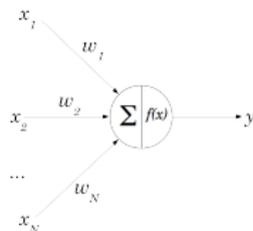
- ▶ Donald Hebb (1949) – “The Hebbian learning rule”
- ▶ Weights are changed as the neuron fires
 - ▶ correct firing \Rightarrow responsible connections (weights) are strengthened
 - ▶ incorrect firing \Rightarrow responsible connections (weights) are weakened

$$w_{ij}(t+1) = w_{ij}(t) + \eta o_i o_j \quad (6)$$

where w_{ij} is weight (strength of connection) between neurons “ i ” and “ j ” and o_i, o_j are outputs of these neurons; η is the learning rate (usually $\ll 1$)

Perceptron

- ▶ Frank Rosenblatt (1957) neural model – perceptron
- ▶ Combination of the McCulloch & Pitts model and Hebbian learning



$$y = f \left(\sum_{i=1}^n x_i w_i - \theta \right) \quad (7)$$

or

$$y = f \left(\sum_{i=1}^{n+1} x_i w_i \right), x_{n+1} = -1 \quad (8)$$

- ▶ The **activation function** f has to be *bounded, monotonous, differentiable*; e.g. the sigmoid function σ

Perceptron learning

- ▶ Activation:

$$y = f \left(\sum_{i=1}^n x_i w_i \right)$$

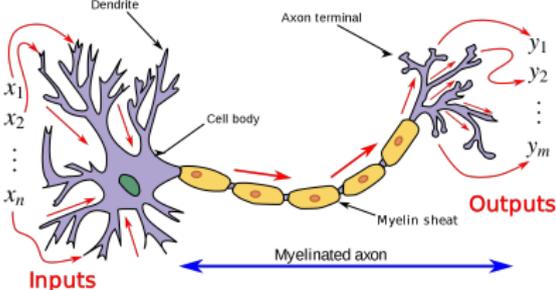
- ▶ Target output: \hat{y}
- ▶ Error:

$$e(w_i) = \frac{1}{2} (\hat{y} - y)^2 \quad (9)$$

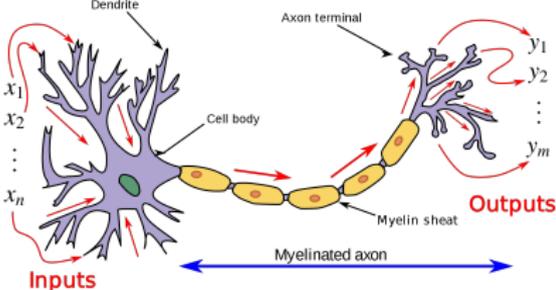
- ▶ Learning rule:

$$w_i(t+1) = w_i(t) + \alpha (\hat{y} - y) x_i \quad (10)$$

Bio-inspiration of the artificial neuron

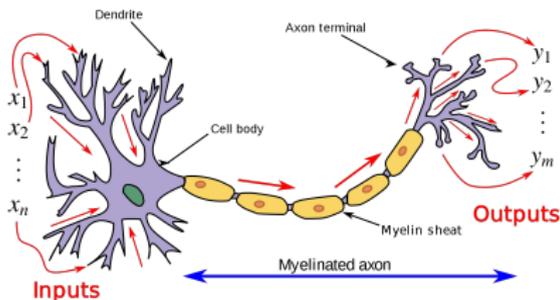


Bio-inspiration of the artificial neuron



This is wrong!

Bio-inspiration of the artificial neuron

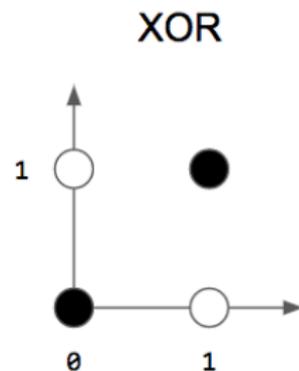
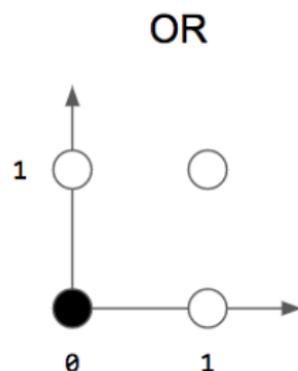
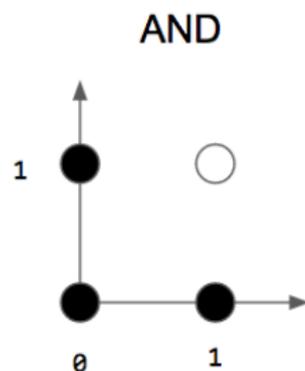


This is wrong!

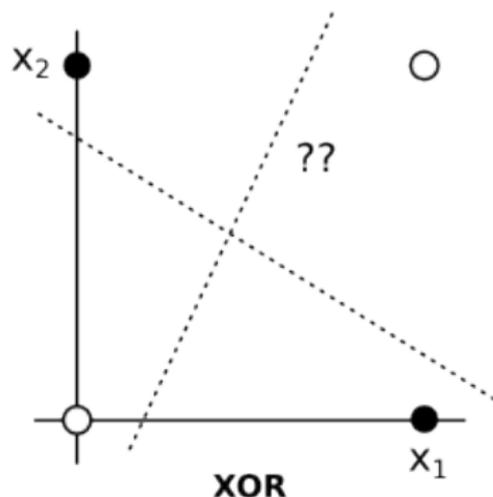
- ▶ More accurate view: a single mathematical model of neuron (a.k.a. artificial neuron) represents (i.e. model the behavior of) a group of biological neurons
- ▶ But mathematical models had diverged from the biological neurons for the sake of computational efficiency and usability
 - ▶ modeling the actual behavior of a neuron is quite complex and costly
- ▶ Remember it is *bio-inspired*, not biologically accurate
 - ▶ There is another field of research trying to develop models that would be almost as efficient as ANNs but more biologically plausible

The XOR problem

- ▶ Marvin Minsky and Seymour Papert (1969)
- ▶ It is impossible for perceptron to learn the XOR function
- ▶ A single perceptron can only classify linearly separable classes
- ▶ Logical functions:



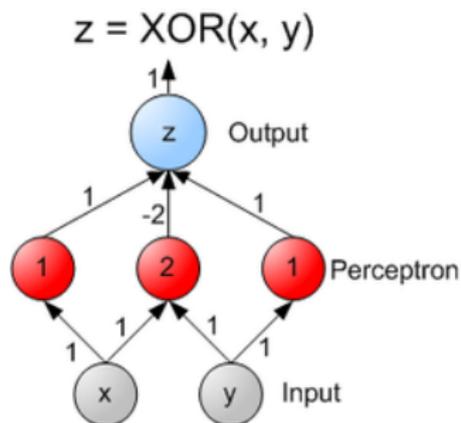
The XOR and the learning problem



- ▶ A good solution would be to stack multiple layers of perceptrons – splitting the problem into sub-problems
 - ▶ but how to adjust the weights?
- ▶ ANNs dormant for many years:
 - ▶ needed the invention of back-propagation learning algorithm
 - ▶ and more computational power

Multilayer perceptron

- ▶ Multilayer perceptron (MLP) – universal function approximator
- ▶ Rumelhart, Hinton, and Williams (1986)



Back-propagation – gradient descent

- ▶ Gradient descent
 - ▶ Adjust weights in the direction of steepest gradient of the error/cost function
 - ▶ Ideally, error computed from every sample
 - ▶ In reality: computationally infeasible (e.g. memory problems)
 - ▶ → stochastic gradient descent (mini-batches)
- ▶ Compute gradient for the error/cost function:

$$J = \sum_{i=1}^n \frac{1}{2} (y_i - \hat{y}_i)^2 \quad (11)$$

- ▶ Activation function (sigmoid function):

$$y_i = \sigma \left(\sum_{j=1}^m x_j w_{ij} - \theta_i \right) \quad (12)$$

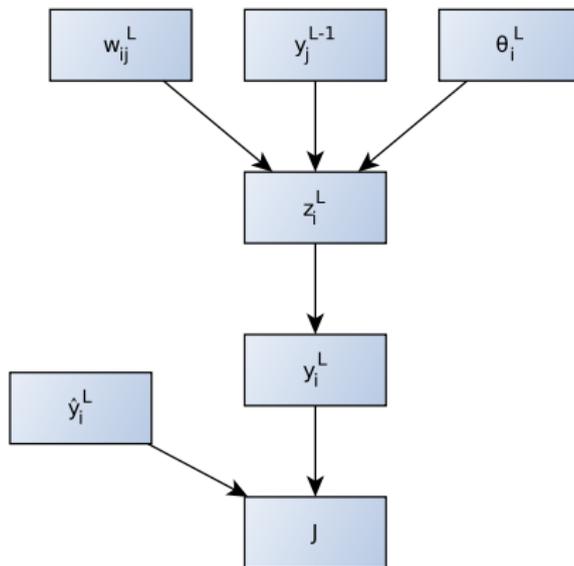
- ▶ Separate the sum from the function:

$$y_i = \sigma (z_i) \quad (13)$$

where

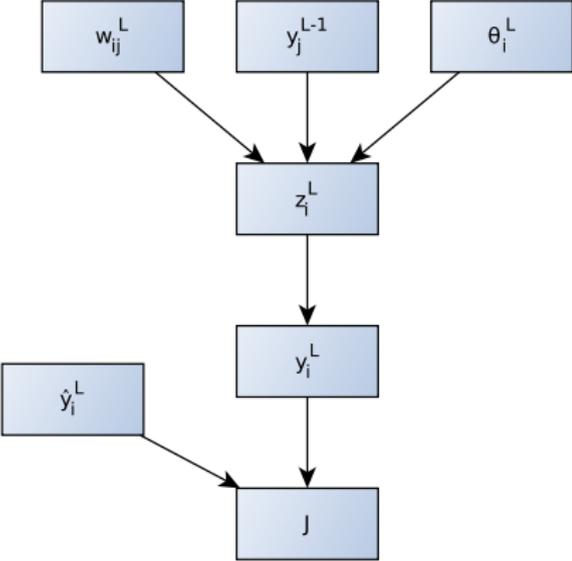
$$z_i = \sum_{j=1}^n x_j w_{ij} - \theta_i \quad (14)$$

Back-propagation – gradient descent



- ▶ Recommended to watch:
3Blue1Brown Series S3 - E4: Backpropagation calculus

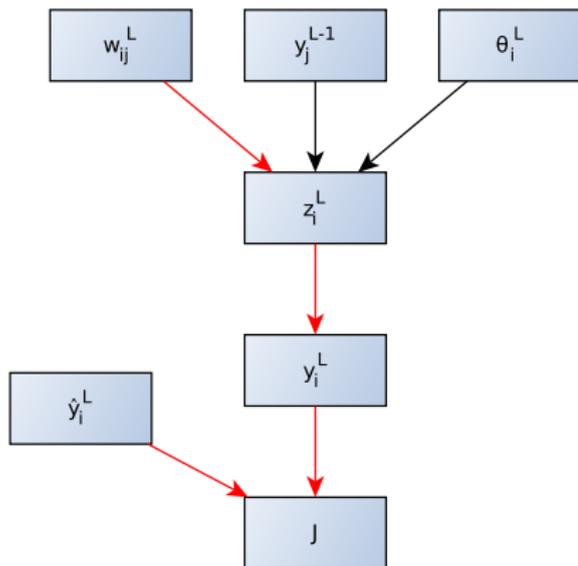
Back-propagation – gradient descent



Back-propagation – gradient descent

$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial z_i^L}{\partial w_{ij}^L} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$

$$\frac{\partial J}{\partial w_{ij}^L} = \sum_{j=1}^m y_j^{L-1} \sigma' (z_i^L) (y_i^L - \hat{y}_i)$$



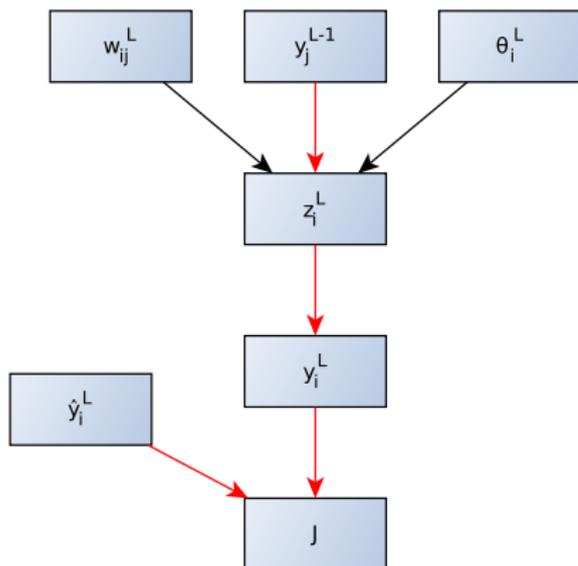
Back-propagation – gradient descent

$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial z_i^L}{\partial w_{ij}^L} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$

$$\frac{\partial J}{\partial w_{ij}^L} = \sum_{j=1}^m y_j^{L-1} \sigma' (z_i^L) (y_i^L - \hat{y}_i)$$

- ▶ Computation of error belonging to the previous layer:

$$\frac{\partial J}{\partial a_j^{L-1}} = \frac{\partial z_i^L}{\partial a_j^{L-1}} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$



Back-propagation – gradient descent

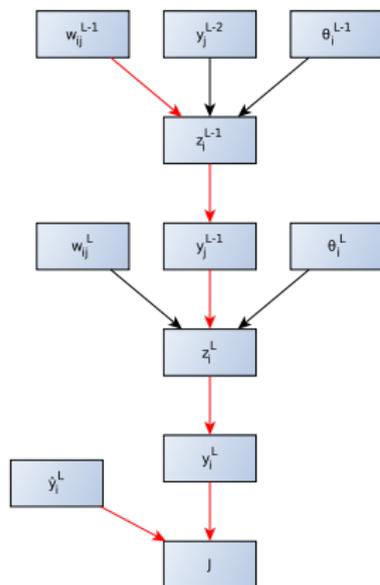
$$\frac{\partial J}{\partial w_{ij}^L} = \frac{\partial z_i^L}{\partial w_{ij}^L} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$

$$\frac{\partial J}{\partial w_{ij}^L} = \sum_{j=1}^m y_j^{L-1} \sigma' \left(z_i^L \right) (y_i^L - \hat{y}_i)$$

- ▶ Computation of error belonging to the previous layer:

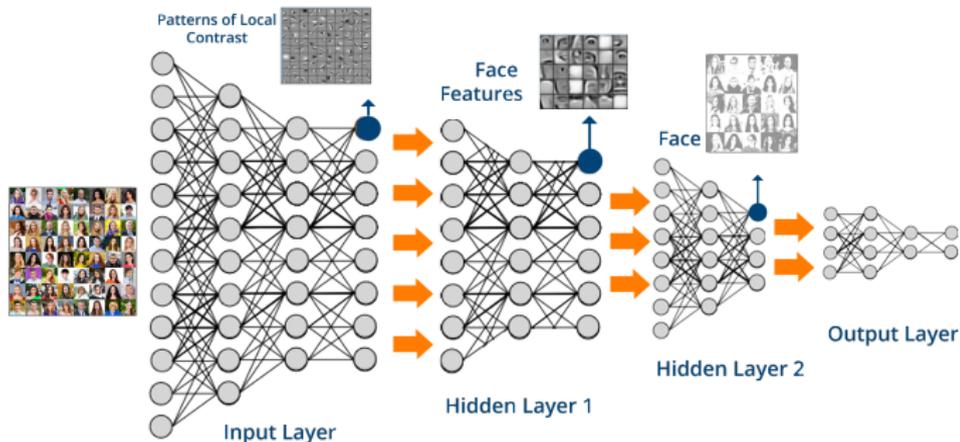
$$\frac{\partial J}{\partial a_j^{L-1}} = \frac{\partial z_i^L}{\partial a_j^{L-1}} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$

$$\frac{\partial J}{\partial a_j^{L-1}} = \sum_{i=1}^n \frac{\partial z_i^L}{\partial a_j^{L-1}} \frac{\partial y_i^L}{\partial z_i^L} \frac{\partial J}{\partial y_i^L}$$



Deep neural networks

- ▶ Hinton (2006)
- ▶ number of layers > 3 ; today $\gg 3$
- ▶ Combination of unsupervised (lower layers) and supervised (higher layers)



Types of learning in ANN

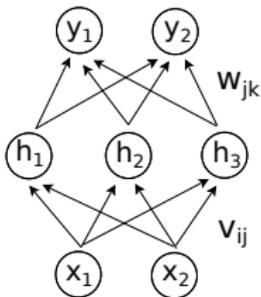
- ▶ **Supervised** learning: weights are adjusted according to the **desired** output (perceptron, MLP, RBF, RNN)
 $s = s(\mathbf{w}, \mathbf{x}, \hat{y})$
- ▶ **Reinforcement** learning: extension of the supervised learning; weights are adjusted according to the **reward** (MLP, RBF)
 $s = s(\mathbf{w}, \mathbf{x}, r)$
- ▶ **Unsupervised** learning: weights are adjusted according to the **statistics** of the input (Hebbian net, SOM)
 $s = s(\mathbf{w}, \mathbf{x})$

Basic types of ANN

Multi-layer perceptron MLP

(Rumelhart, 1986)

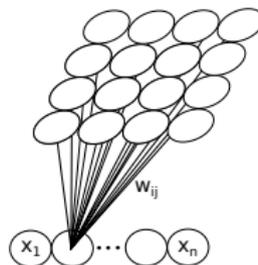
- ▶ hidden layer - allows to solve linearly inseparable problems
- ▶ supervised learning - **error back-propagation (BP)**
- ▶ universal function approximator
- ▶ typical task: input-output mapping, classification, regression, anomaly detection, association



Self-organising map SOM

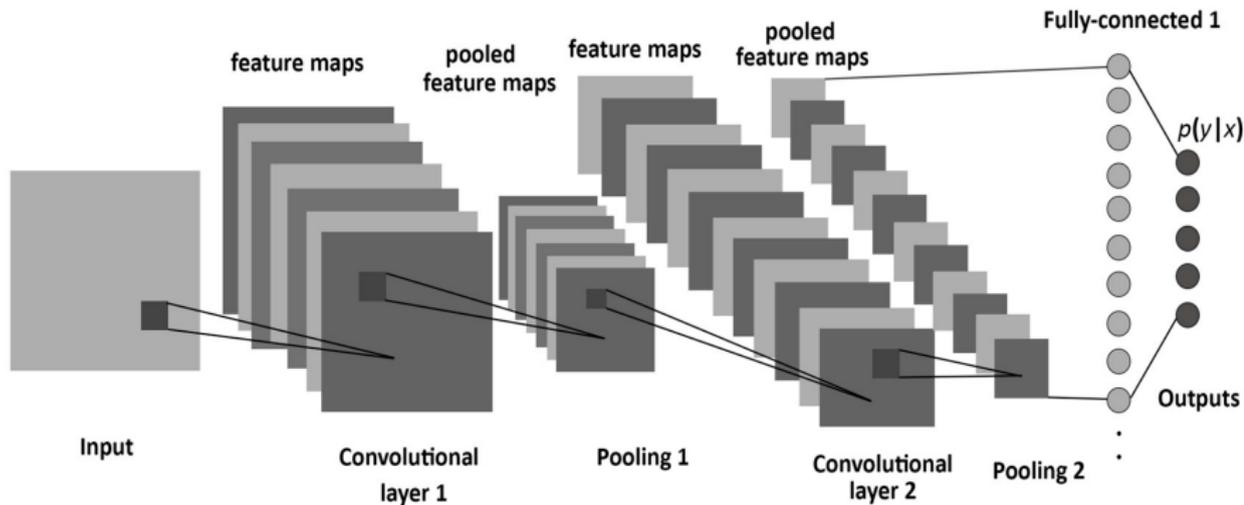
(Kohonen, 1997)

- ▶ unsupervised learning
- ▶ a representation of a data category is usually computed as a **local average of the data**
- ▶ similar data cluster together to same locations in the map - data topology
- ▶ typical tasks: clustering of multidimensional data, categorisation



Convolutional neural networks

- ▶ Slight shift in the input changes the output drastically
- ▶ Especially problematic in images
- ▶ Solution: move (convolve) the “receptive field” of the neuron across the image and accumulate the outputs
- ▶ \Rightarrow convolutional neural networks



Recurrent neural networks

- ▶ Outputs from neurons are used as their own inputs (alongside with other inputs)
- ▶ Very powerful tool for sequential data (NL parsing, signals...)
- ▶ We are not going into it...

ANN problems

- ▶ Under and over-fitting
 - ▶ Over-fitting: network represents the training data too well
 - ▶ bad performance on testing data
 - ▶ stop training when error on testing data increases
 - ▶ **dropout** – deactivate certain neurons during training
 - ▶ Under-fitting
 - ▶ possible causes: slow learning rate or overshooting the local minima – learning rate decay
- ▶ Vanishing gradient
 - ▶ For deep networks, the gradient in back-prop can vanish
 - ▶ Cannot be used for training
 - ▶ Some activation functions (e.g. ReLU) diminishes this effect
- ▶ Scaling
 - ▶ Models use usually small numbers of units
 - ▶ Solutions that work well for small networks with narrow focus fail to deal with large input spaces and multiple tasks
- ▶ Data
 - ▶ Huge amount of parameters (deep networks) = requires huge amount of data

A puppy or a muffin?



Example

<https://playground.tensorflow.org>

Thank you for your attention