@alexiasalavrakos.bsky.social

TALENT ARENA WORKSHOP:

BUILD YOUR FIRST NEURAL NETWORK

Alexia Salavrakos

4 March 2026

ABOUT MYSELF

- Studied physics at Universite Libre de Bruxelles, Belgium
- PhD at Institute of Photonic Sciences (ICFO) in Barcelona, Spain, in quantum technologies
- Worked for three years at Clearpay as a data scientist
- Joined Quandela in June 2022, a startup for photonic quantum computing

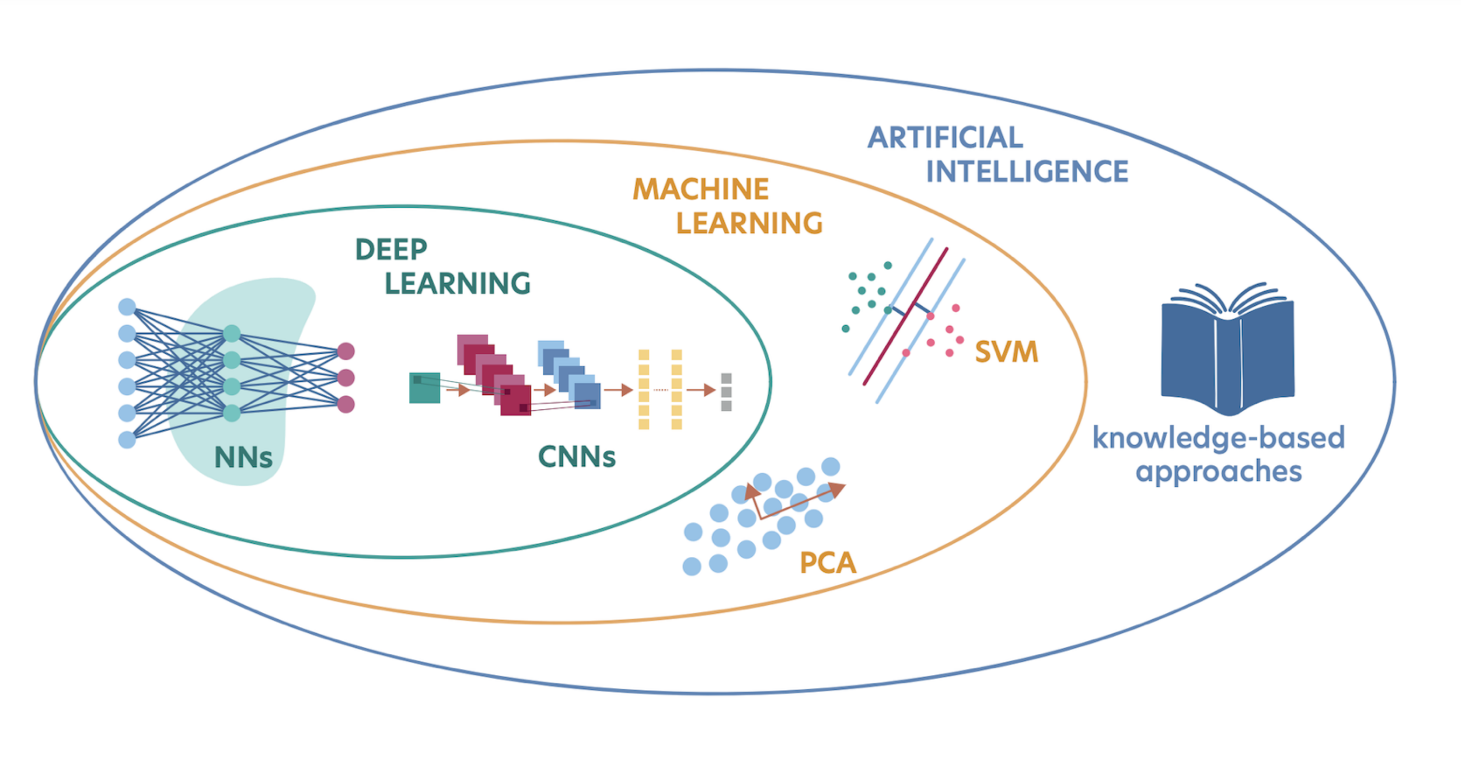
@alexiasalavrakos.bsky.social

Linkedin

CODEWOMEN

MACHINE LEARNING

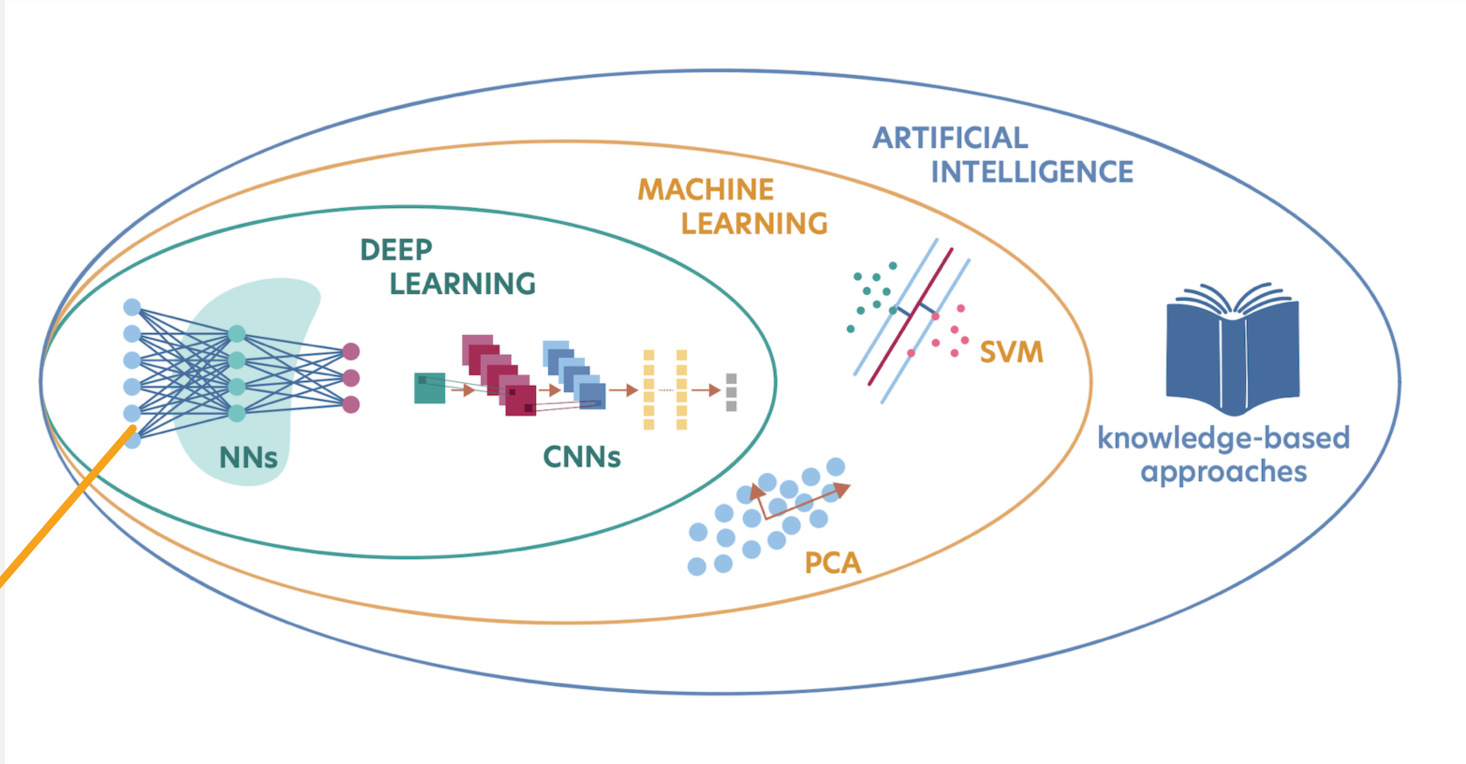
ML: a branch of AI devoted to building methods that learn from the data



A. Dawid et al. arXiv:2204.04198 (2022)

MACHINE LEARNING

ML: a branch of AI devoted to building methods that learn from the data



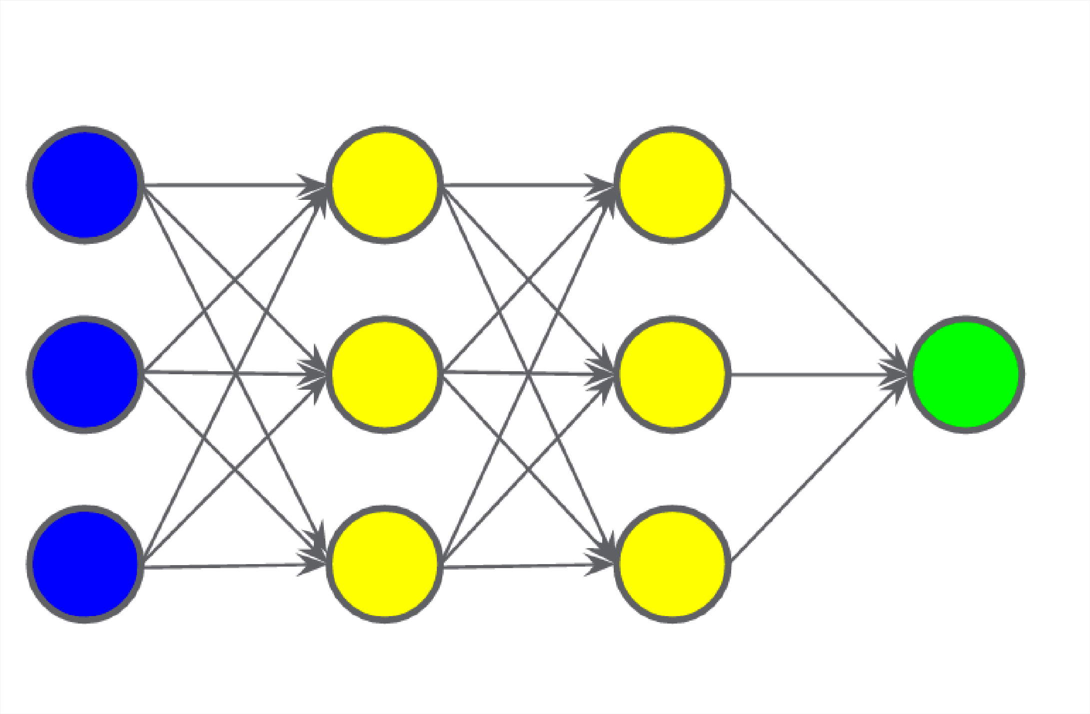
NN: neural networks

A. Dawid et al. arXiv:2204.04198 (2022)

A BRIEF HISTORY OF NEURAL NETWORKS

- **1943:** first mathematical model of a neuron (Pitts and McCulloch)
- **1958:** the perceptron, first neural network proposal (Rosenblatt)
- **1970s-1980s:** backpropagation algorithm (independently developed by several authors)
- **mid 2000s - early 2010s:** GPUs and beginning of deep learning
- **2014:** generative adversarial neural networks (Goodfellow et al.)
- **2017:** first transformer proposal, "attention is all you need" (Google)

STRUCTURE OF A NEURAL NETWORK



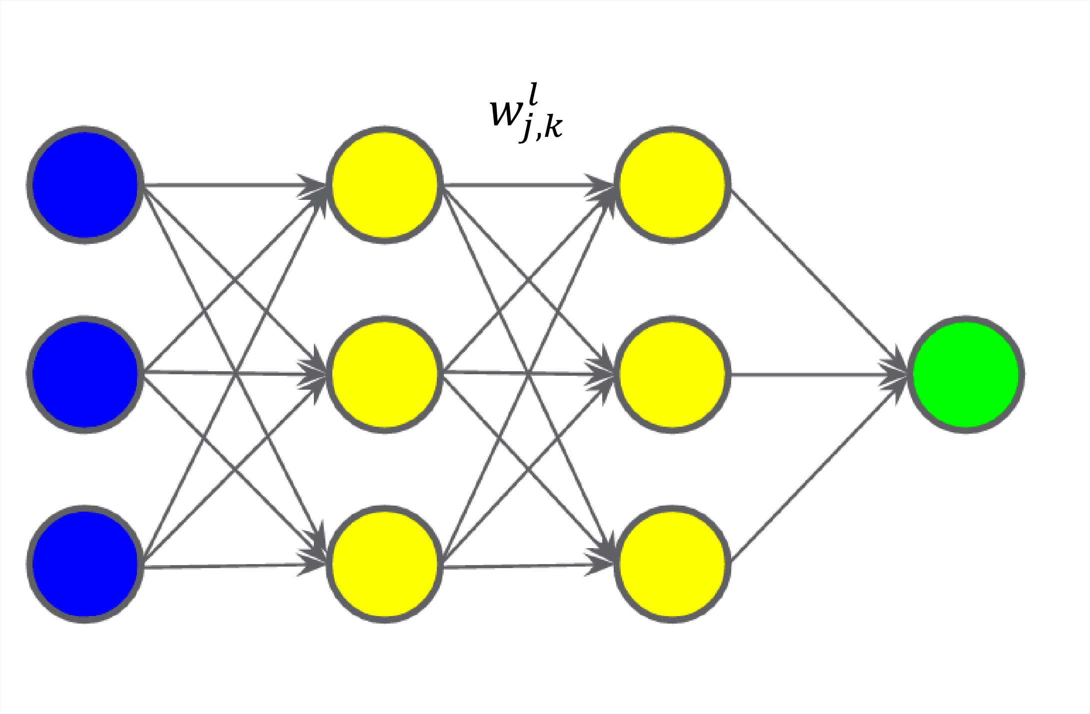
Input layer Hidden layers Output layer

feedforward fully connected network (FFCN)

or

multilayer perceptron (MLP)

STRUCTURE OF A NEURAL NETWORK



Input layer Hidden layers Output layer

activation function

$$a_j^l = \sigma(\Sigma_k w_{j,k}^l a_k^{l-1} + b_j^l)$$

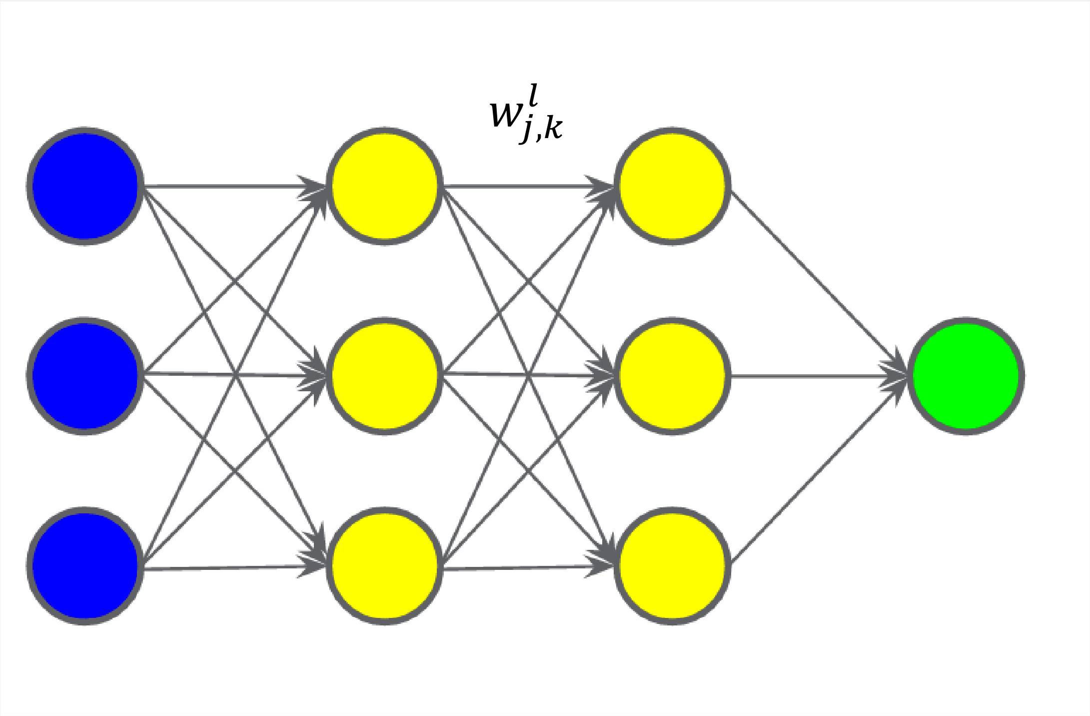
value of neuron j at layer l

weights

bias

value of neuron k at layer l-1

STRUCTURE OF A NEURAL NETWORK



Input layer

Hidden layers

Output layer

activation function

$$a_j^l = \sigma(\Sigma_k w_{j,k}^l a_k^{l-1} + b_j^l)$$

value of neuron j at layer l

weights

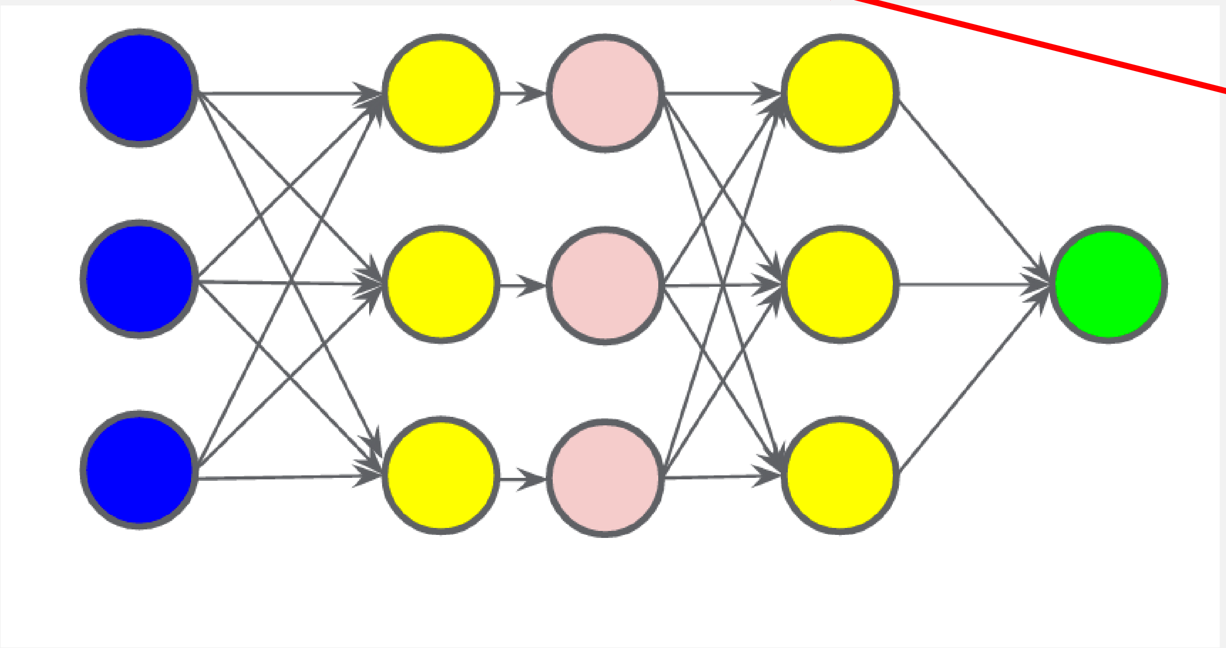
bias

value of neuron k at layer l-1

$$a_1^2 = \sigma(w_{1,1}^2 a_1^1 + w_{1,2}^2 a_2^1 + w_{1,3}^2 a_3^1 + \ \dots)$$

STRUCTURE OF A NEURAL NETWORK

activation function



Input layer

Hidden layers

Output layer

$$a_j^l = \sigma(\Sigma_k w_{j,k}^l a_k^{l-1} + b_j^l)$$

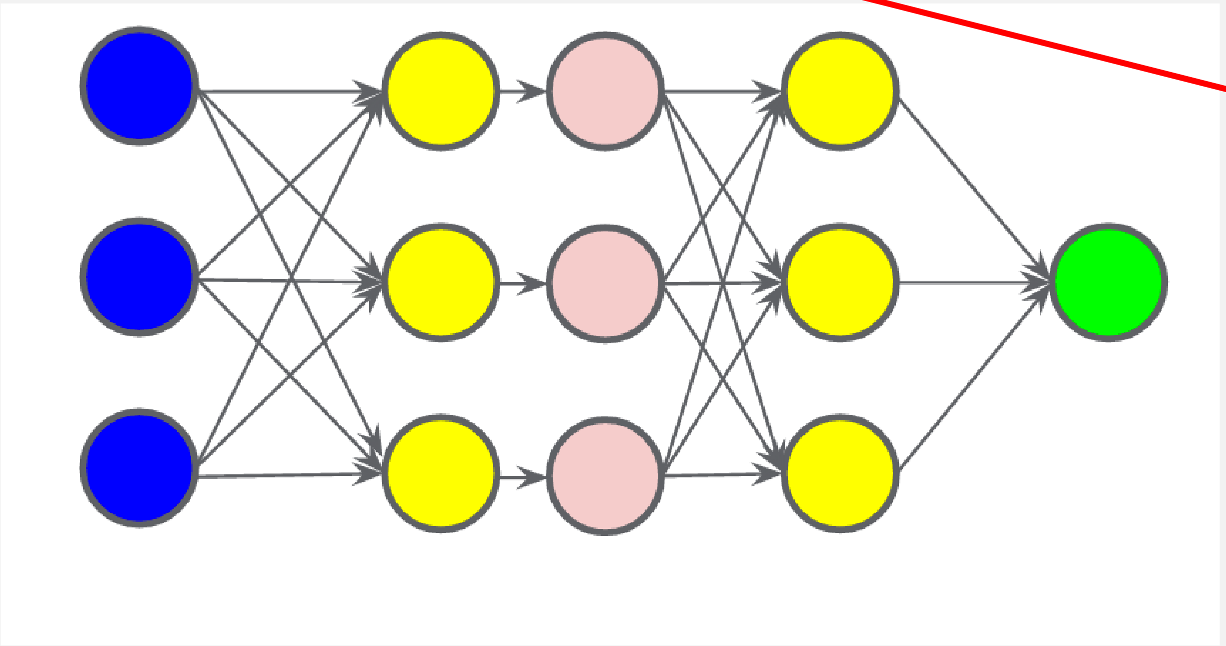
Often used: sigmoid, ReLU

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

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

STRUCTURE OF A NEURAL NETWORK

activation function



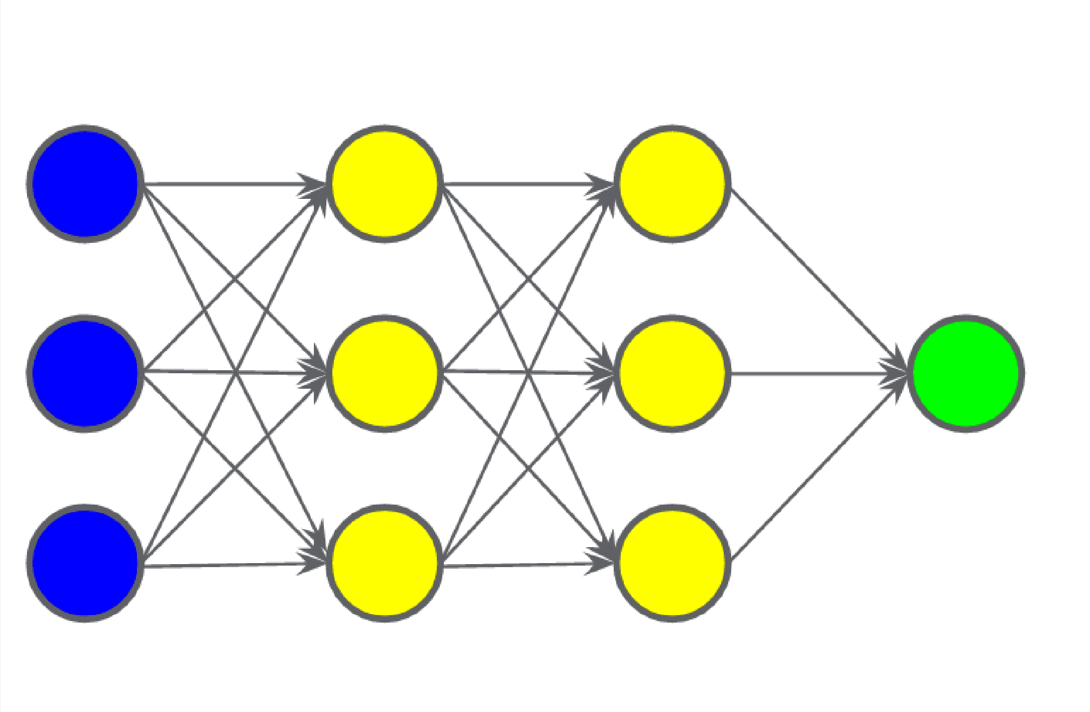
Input layer Hidden layers Output layer

$$a_j^l = \sigma(\Sigma_k w_{j,k}^l a_k^{l-1} + b_j^l)$$

Without the activation functions, a neural network would simply be linear

$$y = ax + b$$

STRUCTURE OF A NEURAL NETWORK



Input layer Hidden layers Output layer

$$a_j^l = \sigma(\Sigma_k w_{j,k}^l a_k^{l-1} + b_j^l)$$

↓

$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^l\right)$$

vector form

STRUCTURE OF A NEURAL NETWORK

$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^l\right)$$

vector form

$$\begin{pmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{1,1}^2 & w_{1,2}^2 & w_{1,3}^2 \\ w_{2,1}^2 & w_{2,1}^2 & w_{2,3}^2 \\ w_{3,1}^2 & w_{3,2}^2 & w_{3,3}^2 \end{pmatrix}\begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} + \begin{pmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{pmatrix}\right)$$

STRUCTURE OF A NEURAL NETWORK

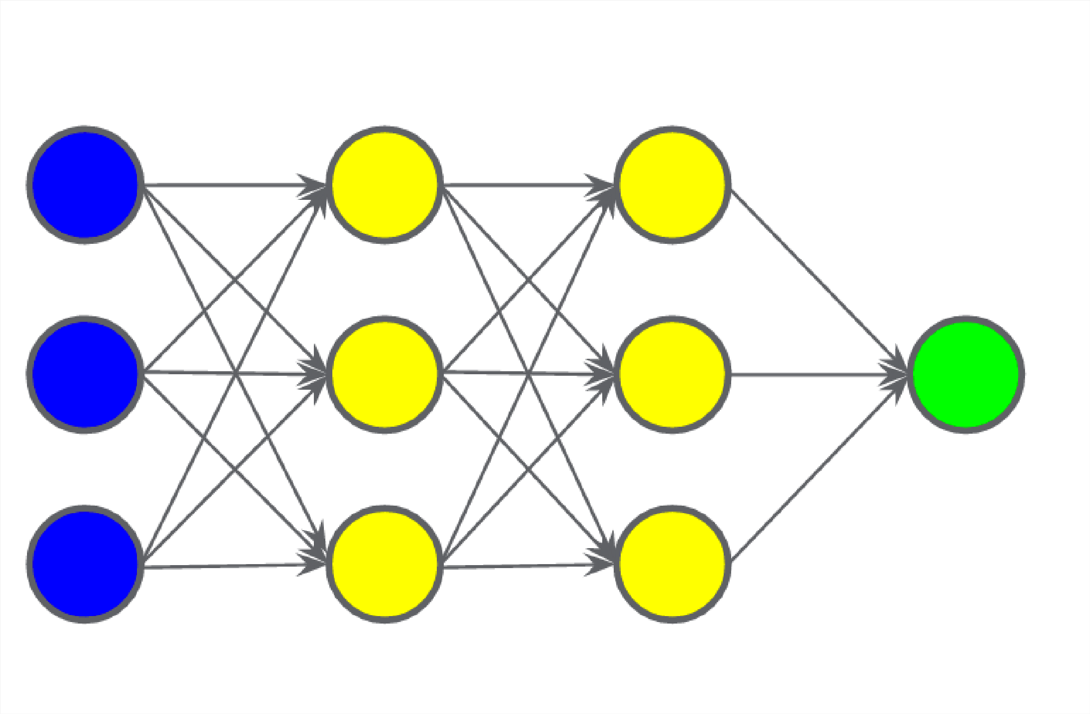
$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^{\mathrm{l}}\right)$$

vector form

$$\begin{pmatrix} a_1^2 \\ a_2^2 \\ a_3^2 \end{pmatrix} = \sigma\left(\begin{pmatrix} w_{1,1}^2 & w_{1,2}^2 & w_{1,3}^2 \\ w_{2,1}^2 & w_{2,1}^2 & w_{2,3}^2 \\ w_{3,1}^2 & w_{3,2}^2 & w_{3,3}^2 \end{pmatrix}\begin{pmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{pmatrix} + \begin{pmatrix} b_1^2 \\ b_2^2 \\ b_3^2 \end{pmatrix}\right)$$

"Neural networks is just matrix multiplication"

STRUCTURE OF A NEURAL NETWORK



Input layer Hidden layers Output layer

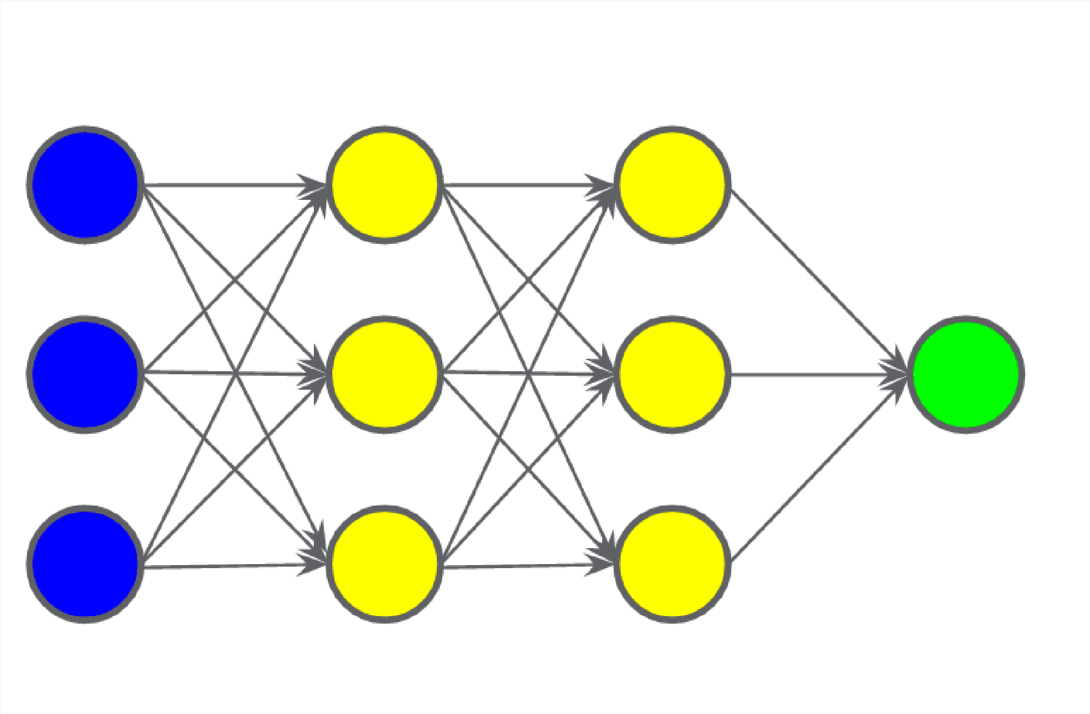
$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^l\right)$$

$$a^l = F_{l-1\to l}\left(a^{l-1}\right)$$

$$a^L = F_{L-1\to L} \bullet \cdots \bullet F_{1\to 2}(a^1)$$

for a network of depth L

STRUCTURE OF A NEURAL NETWORK



Input layer　Hidden layers　Output layer

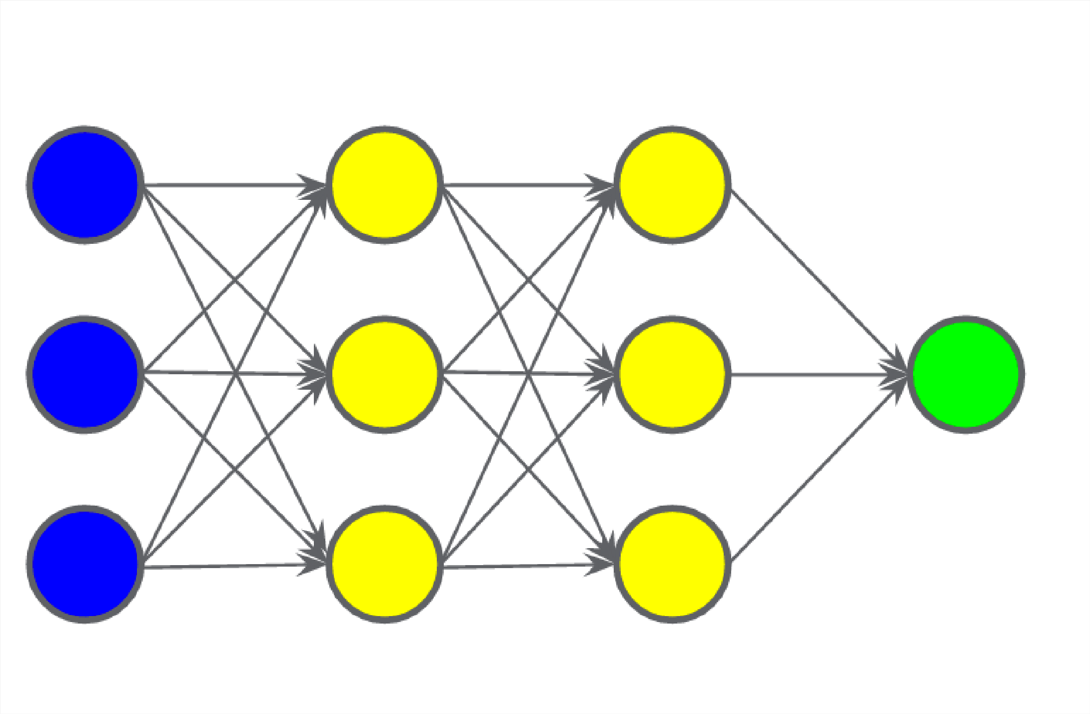
$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^l\right)$$

L: final layer

$$C = C\left(a^L\right)$$

cost function

STRUCTURE OF A NEURAL NETWORK



Input layer Hidden layers Output layer

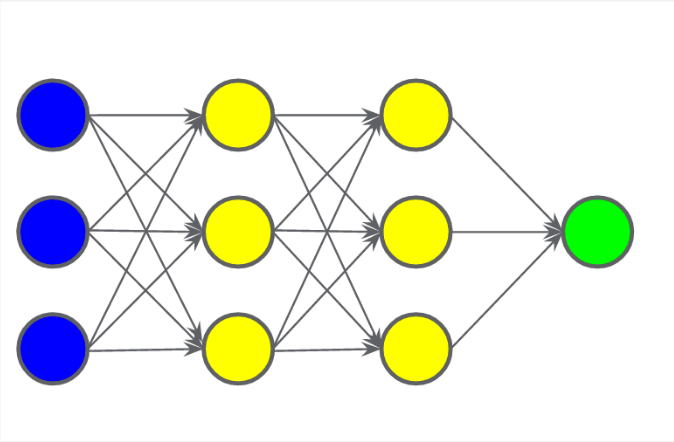
$$a^l = \sigma\left(W^l a^{l-1} + \mathrm{b}^l\right)$$

$$C = \frac{1}{2}\left|\left|y - a^L\right|\right|^2$$

e.g. quadratic cost function

TRAINING A NEURAL NETWORK

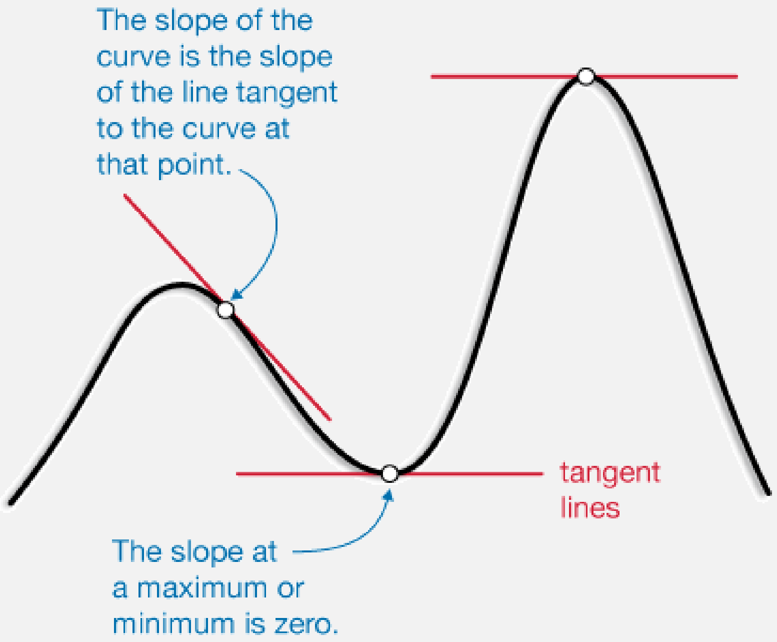
Gradient descent:
update the weights and the biases according to the gradient of the cost function

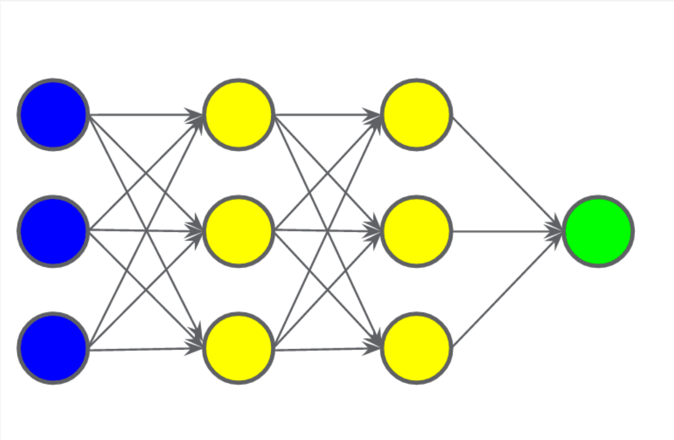


TRAINING A NEURAL NETWORK

The gradient is like the derivative but in several dimensions

$$\frac{d}{dx}(x^2) = 2x$$





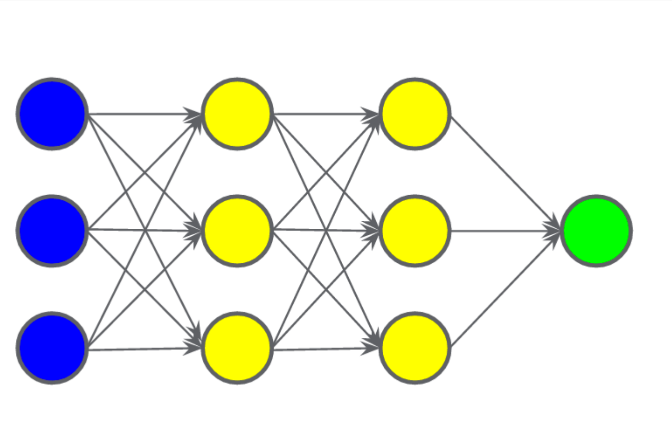
https://xaktly.com/TheDerivative.html

TRAINING A NEURAL NETWORK

Gradient descent:
update the weights and the biases according to the gradient of the cost function

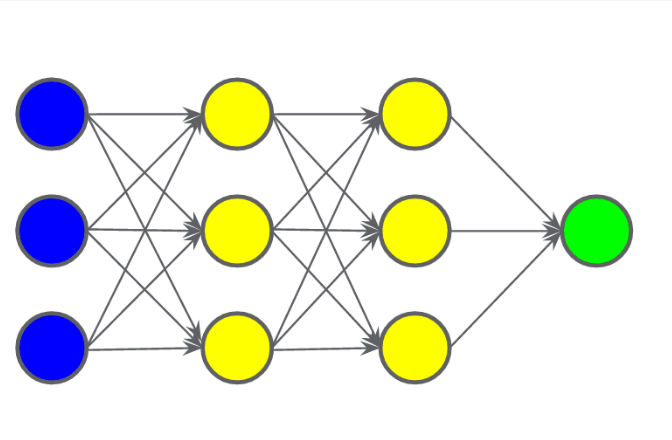
$$w_{j,k}^{l} \rightarrow w_{j,k}^{l\prime} = w_{j,k}^{l} - \eta \frac{\partial C}{\partial w_{j,k}^{l}}$$

$$b_{j}^{l} \rightarrow b_{j}^{l\prime} = b_{j}^{l} - \eta \frac{\partial C}{\partial b_{j}^{l}}$$



TRAINING A NEURAL NETWORK

Gradient descent:
update the weights and the biases according to the gradient of the cost function



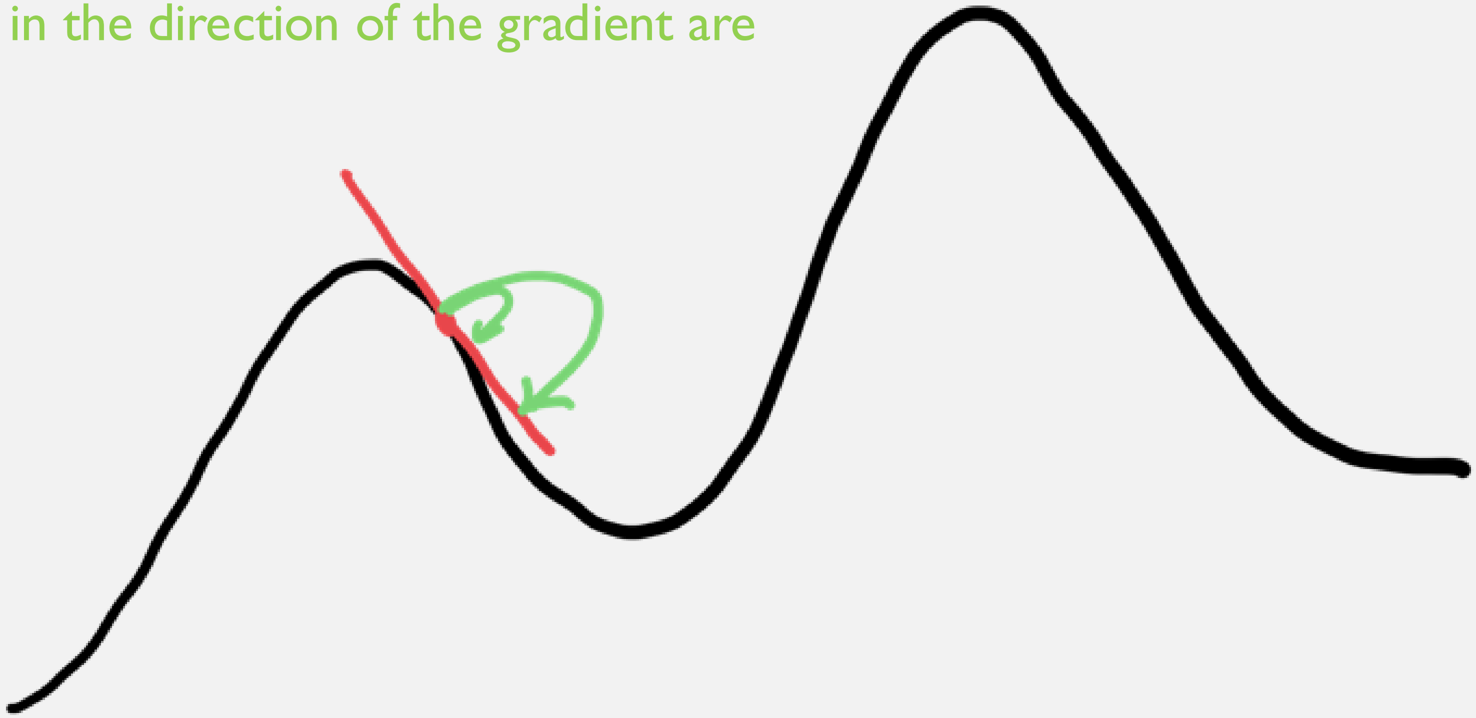
$$w_{j,k}^{l} \rightarrow w_{j,k}^{l\prime} = w_{j,k}^{l} - \eta \frac{\partial C}{\partial w_{j,k}^{l}}$$

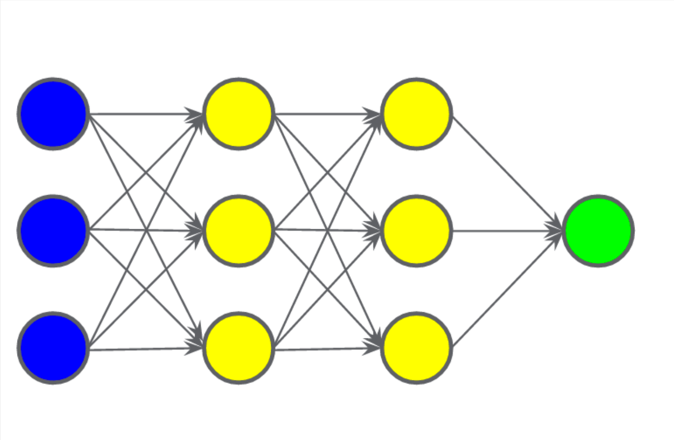
$$b_{j}^{l} \rightarrow b_{j}^{l\prime} = b_{j}^{l} - \eta \frac{\partial C}{\partial b_{j}^{l}}$$

η → learning rate

TRAINING A NEURAL NETWORK

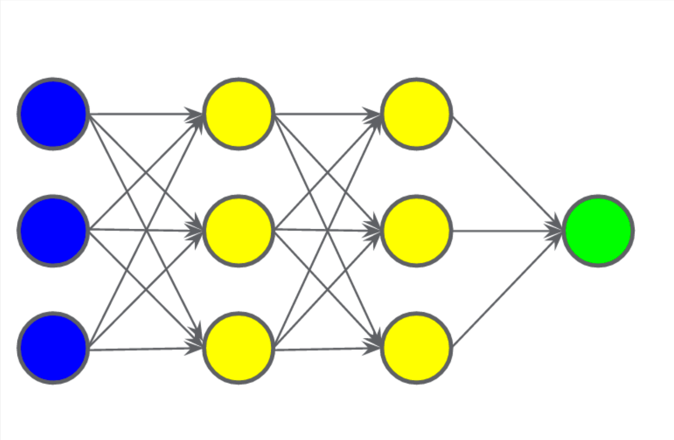
The learning rate tells you how "big" the steps you are taking in the direction of the gradient are





TRAINING A NEURAL NETWORK

Gradient descent:
update the weights and the biases according to the gradient of the cost function



$$w_{j,k}^{l} \rightarrow w_{j,k}^{l\prime} = w_{j,k}^{l} - \eta \frac{\partial C}{\partial w_{j,k}^{l}}$$

$$b_{j}^{l} \rightarrow b_{j}^{l\prime} = b_{j}^{l} - \eta \frac{\partial C}{\partial b_{j}^{l}}$$

How do we compute this gradient?

TRAINING A NEURAL NETWORK

Training algorithm:

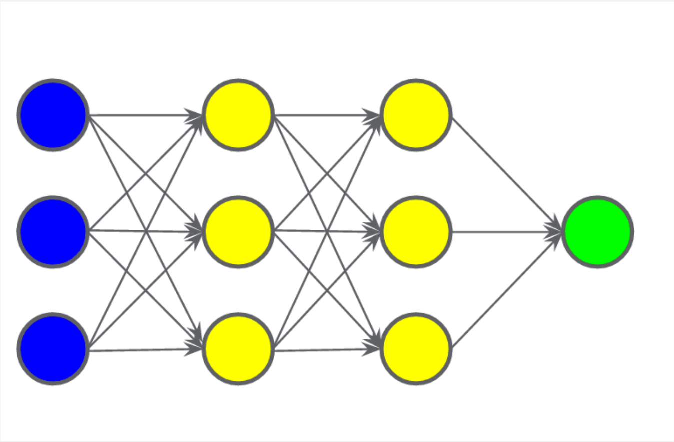
1. Input a set of training examples

2. Feedforward

3. Output error

4. Backpropagate the error thanks to the **chain rule**

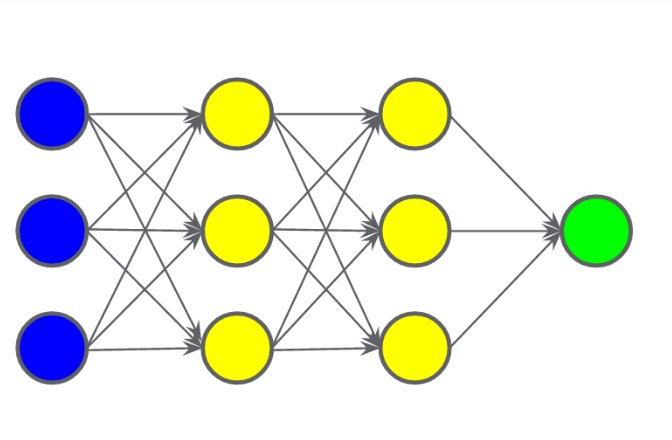
5. Output and gradient descent



TRAINING A NEURAL NETWORK

Training algorithm:

1. Input a set of training examples $x \longrightarrow a^1$
2. Feedforward $l = 2, \dots, L \longrightarrow a^l = \sigma\left(W^l a^{l-1} + b^l\right)$
3. Output error $\frac{\partial C}{\partial a_j^L} \sigma'\left(z_j^L\right)$
4. Backpropagate the error thanks to the **chain rule**
5. Output and gradient descent $l = L-1, \dots, 2$ $\frac{\partial C}{\partial b_j^l}$ $\frac{\partial C}{\partial w_{j,k}^l}$



chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

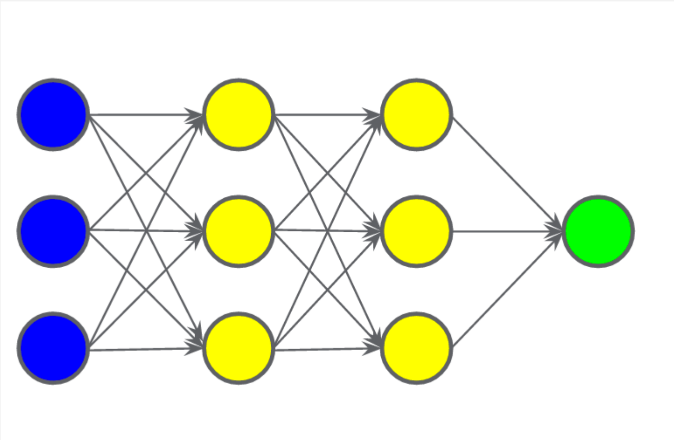
TRAINING A NEURAL NETWORK

Gradient descent:
update the weights and the biases according to the gradient of the cost function

$$w_{j,k}^{l} \rightarrow w_{j,k}^{l\prime} = w_{j,k}^{l} - \eta \frac{\partial C}{\partial w_{j,k}^{l}}$$

$$b_{j}^{l} \rightarrow b_{j}^{l\prime} = b_{j}^{l} - \eta \frac{\partial C}{\partial b_{j}^{l}}$$

values obtained with backprop

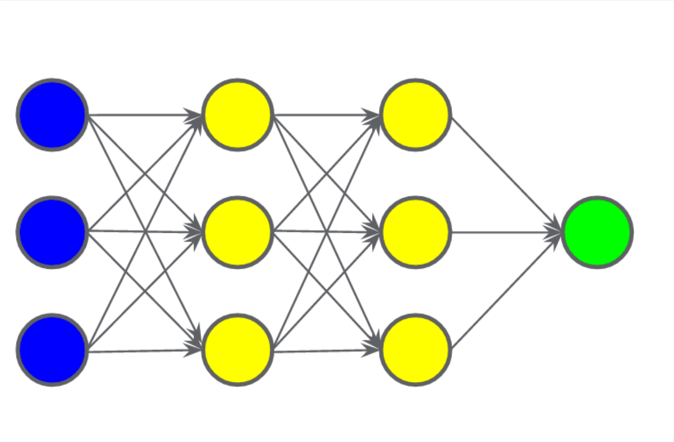


TRAINING A NEURAL NETWORK

Stochastic gradient descent:
mini-batch of m randomly chosen training inputs $X_1, \ldots, X_m$

$$w_{j,k}^{l} \rightarrow w_{j,k}^{l\prime} = w_{j,k}^{l} - \frac{\eta}{m} \sum_{i} \frac{\partial C_{Xi}}{\partial w_{j,k}^{l}}$$

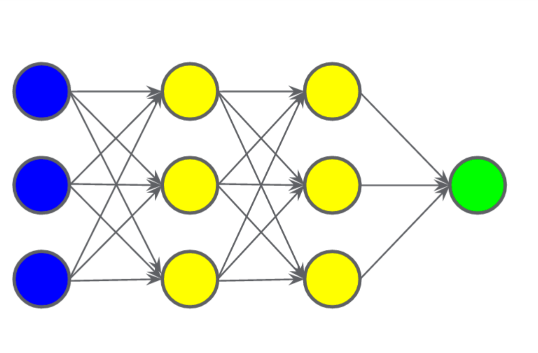
$$b_{j}^{l} \rightarrow b_{j}^{l\prime} = b_{j}^{l} - \frac{\eta}{m} \sum_{i} \frac{\partial C_{Xi}}{\partial b_{j}^{l}}$$



Go through all mini batches = 1 epoch

Speeds up training

TRAINING A NEURAL NETWORK



Why is backpropagation a ”fast” algorithm?

Computing the gradient with finite difference approximation would require as many feedforwards as there are weights → this could mean up to millions of forward passes

$$\frac{\partial C}{\partial w_j} \approx \frac{C(w + \epsilon e_j) - C(w)}{\epsilon}$$

On the other hand with backpropagation, only 1 forward and 1 backward passes

Algorithm first proposed in 1970s
Importance speedup rediscovered 1986

TUNING NEURAL NETWORKS

Choice of cost function (e.g. cross-entropy) and activation functions

Overfitting in neural networks and number of parameters

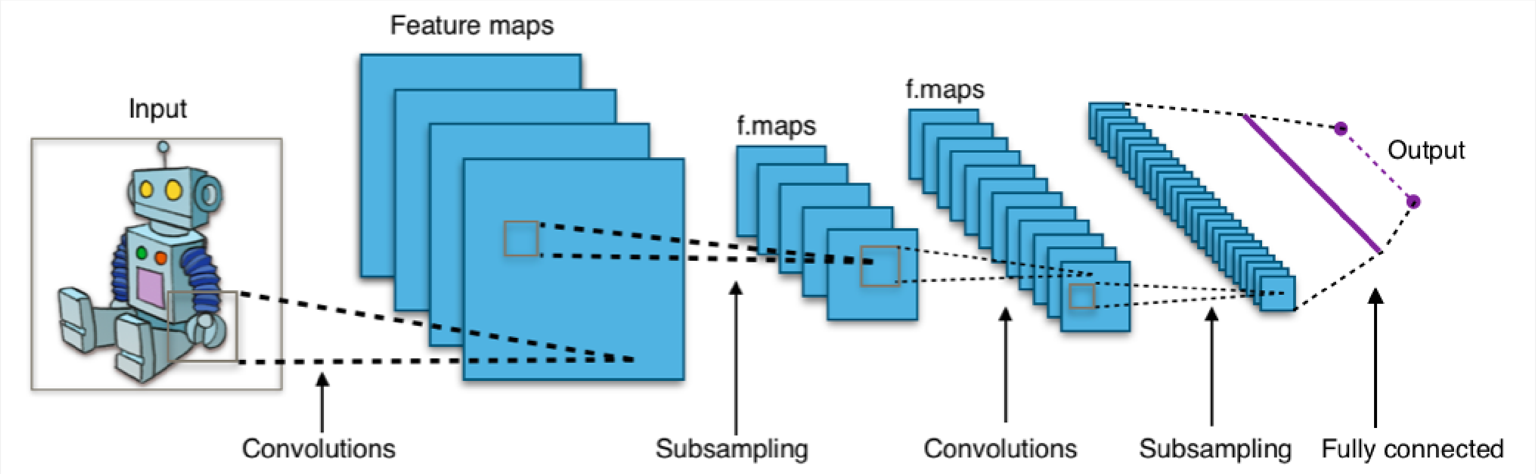
Regularisation: L1 and L2 regularisation, dropout

Weight initialisation

Hyperparameters

OTHER TYPES OF NEURAL NETWORKS

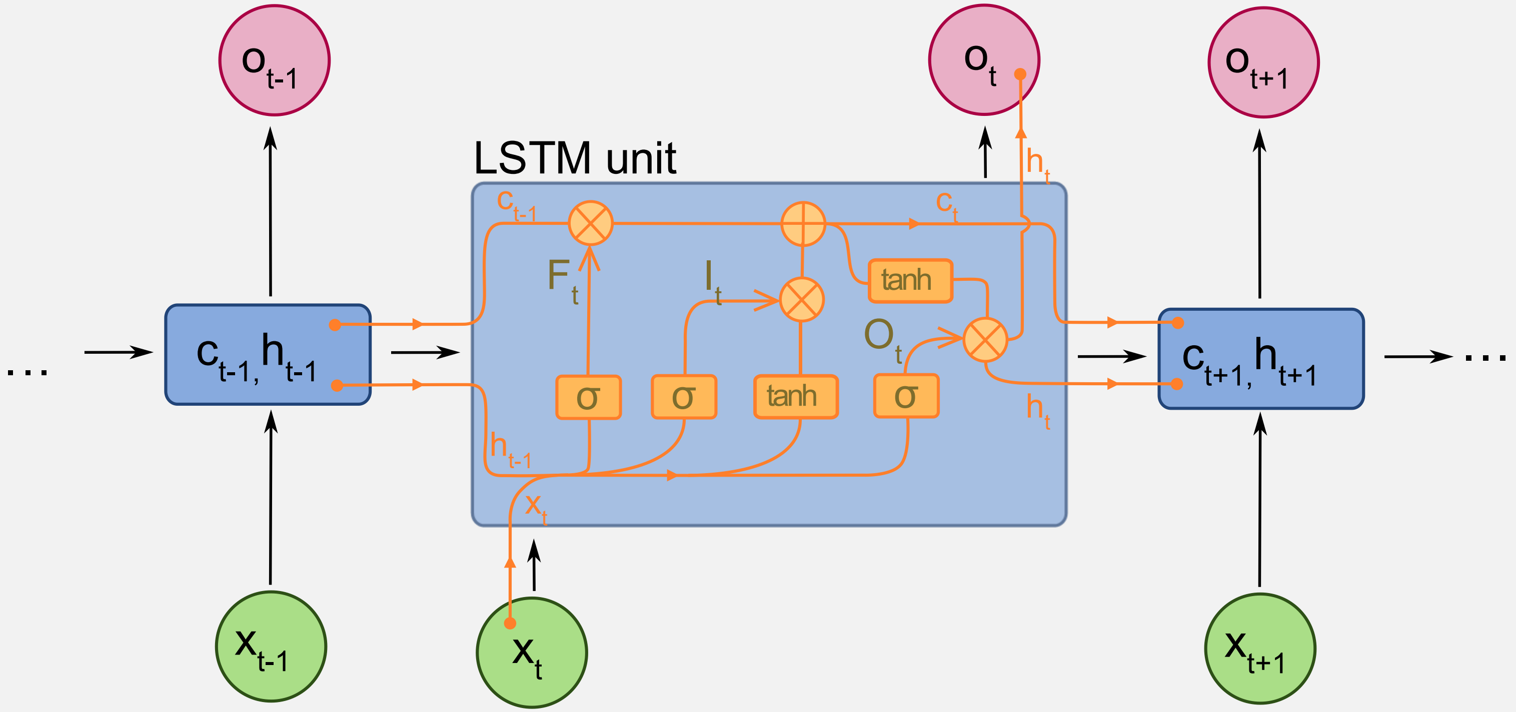
Convolutional neural networks (CNNs) for image data



By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374

OTHER TYPES OF NEURAL NETWORKS

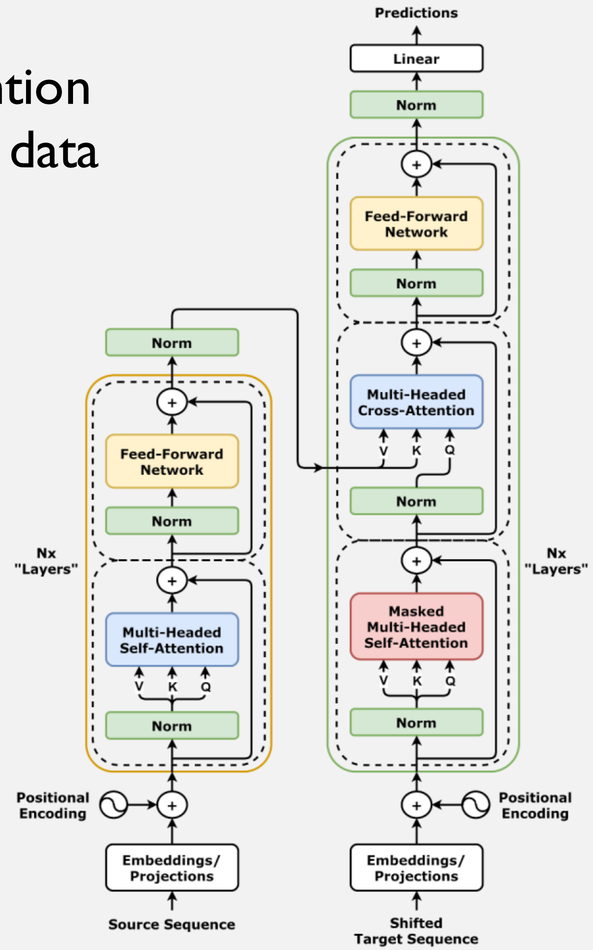
Recurrent neural networks (RNNs) for time series data



By fdeloche - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=60149410

OTHER TYPES OF NEURAL NETWORKS

Transformer architectures with attention mechanism for large-scale sequential data



By dvgodoy - https://github.com/dvgodoy/dl-visuals/?tab=readme-ov-file, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=151216016

IN A NUTSHELL

- Neural networks are a type of machine learning models initially inspired from neurons
- Made of **layers of weights with activation functions** in between to introduce nonlinearities
- Weights are trained with **backpropagation algorithm** based on chain rule for derivatives
- Backpropagation is very efficient
- Specific "biases" can be introduced in the structure of neural networks to better suit certain types of problems

Free online book:
Michael A. Nielsen "Neural networks and Deep Learning".
http://neuralnetworksanddeeplearning.com/

NOW, LET'S TRAIN A NEURAL NETWORK IN PYTORCH!

https://colab.research.google.com/drive/1ussR4iC0CubF8bxP5HTxSsr-fO-OmVDO?usp=sharing

https://tinyurl.com/2petju2w

Go to File --> Save a copy in Drive

Based on
https://docs.pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

EXERCISE

Try to improve the model performance by playing around with model architecture, hyperparameters, etc

What do you observe?

Suggestions:

- Layer sizes
- Activation functions
- Number of epochs
- Batch size

- More advanced: use a CNN

APPENDIX

NOW, LET'S TRAIN A NEURAL NETWORK IN PYTORCH!

https://github.com/alexiasalavrakos/codewomen-pytorch

Based on
https://docs.pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html