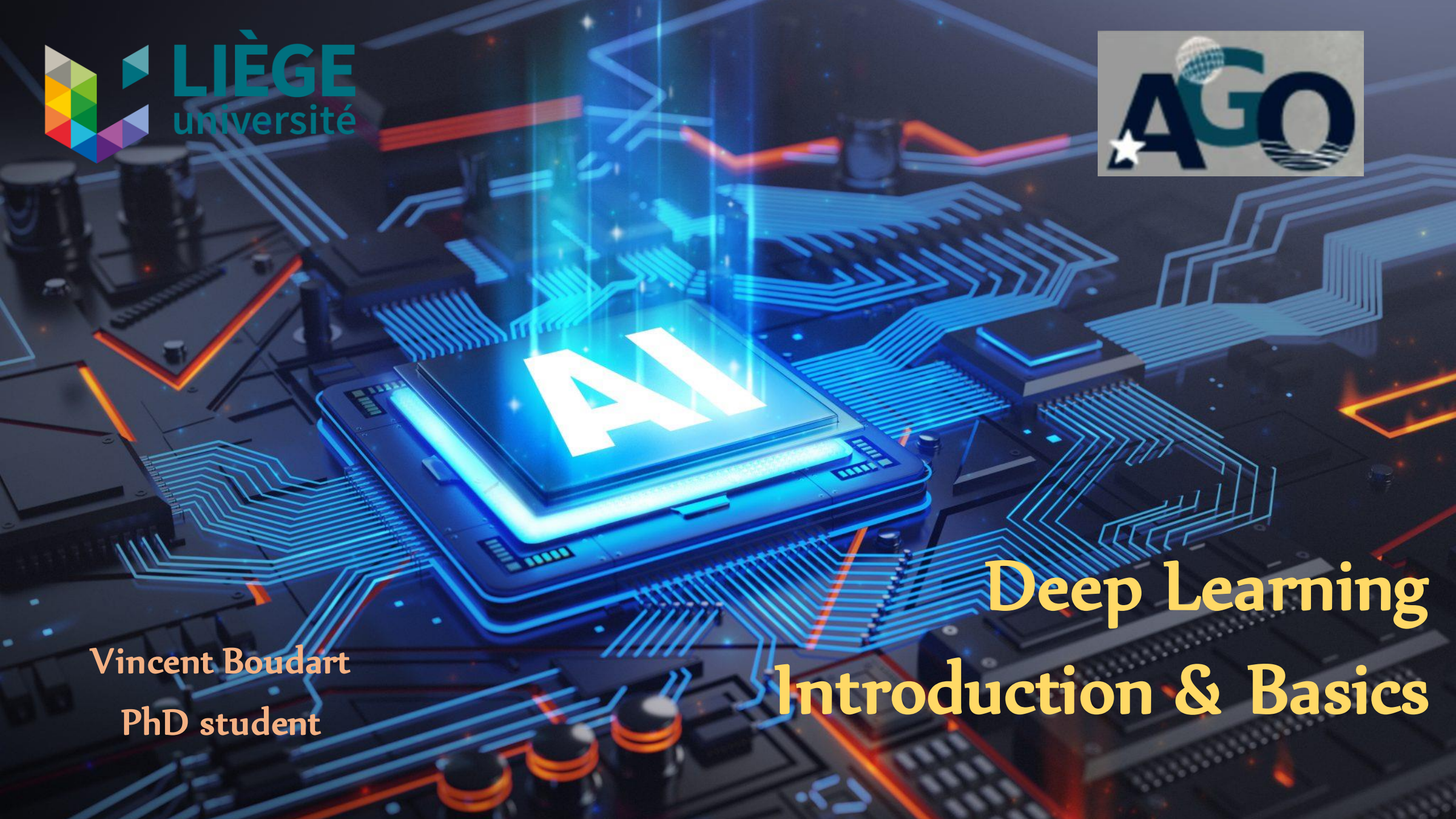


LIÈGE université

AGO

Deep Learning
Introduction & Basics

Vincent Boudart
PhD student

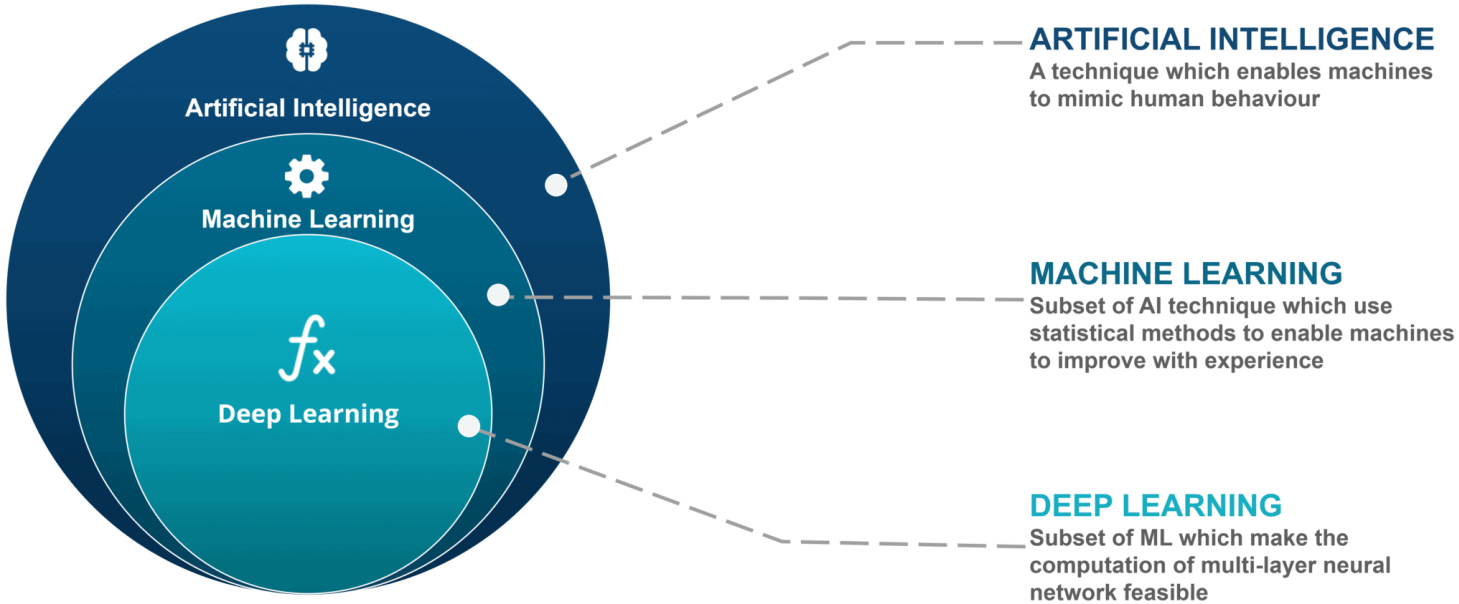
Course 1 : Table of contents

1) What is deep learning ?

- Artificial Intelligence vs Machine Learning vs Deep Learning

(http://wiki.pathmind.com/ai-vs-machine-learning-vs-deep-learning)

- pile of if-then statements
- statistical model mapping raw sensory data to symbolic categories
- ...



- optimization algorithms
- decision tree, random forest, K-means linear regression, support vector machine, ...

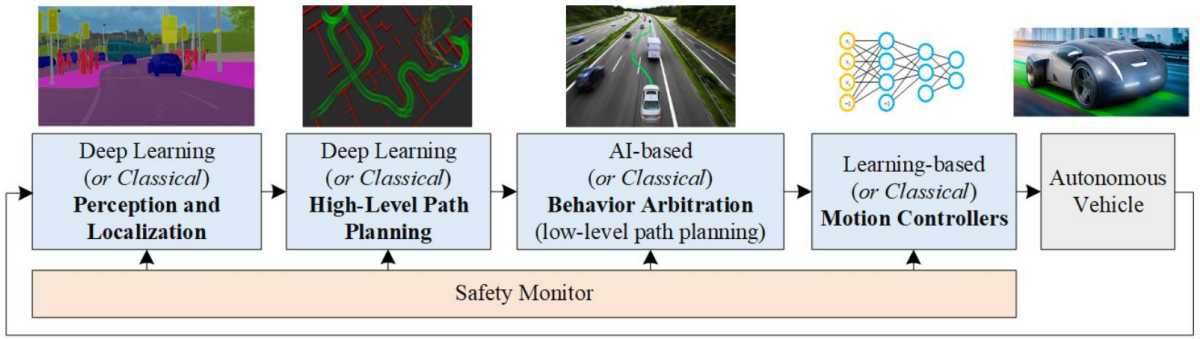
- neural networks
- performs exceptionally well on lots of tasks

2) How it is used nowadays ?

- Self-driving cars (Tesla, Google)

- Lots of data to process (camera, LiDAR, RADAR, GPS, various sensors, ...)

- Lots of action to take (breaking, turning, prediction of human and vehicle behavior, ...)



©Google self-driving car

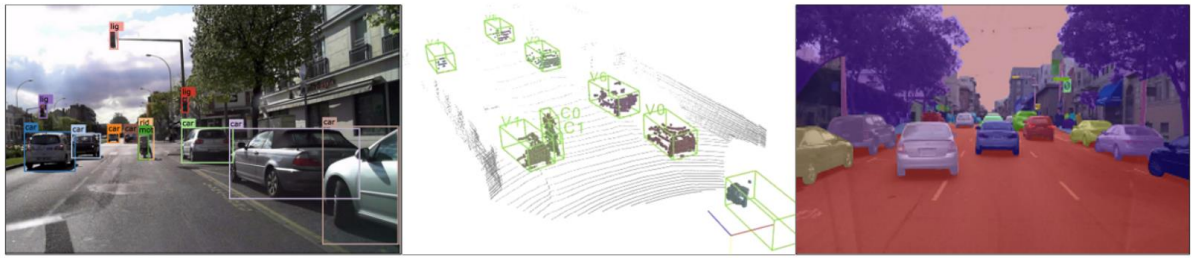
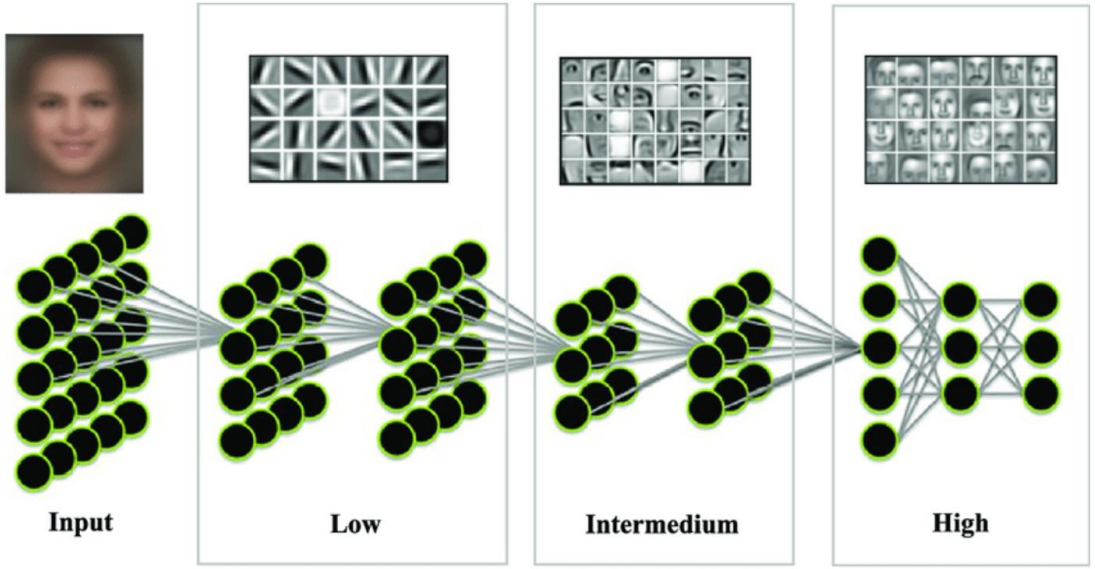


Figure 3: **Examples of scene perception results.** (a) 2D object detection in images. (b) 3D bounding box detector applied on LiDAR data. (c) Semantic segmentation results on images.

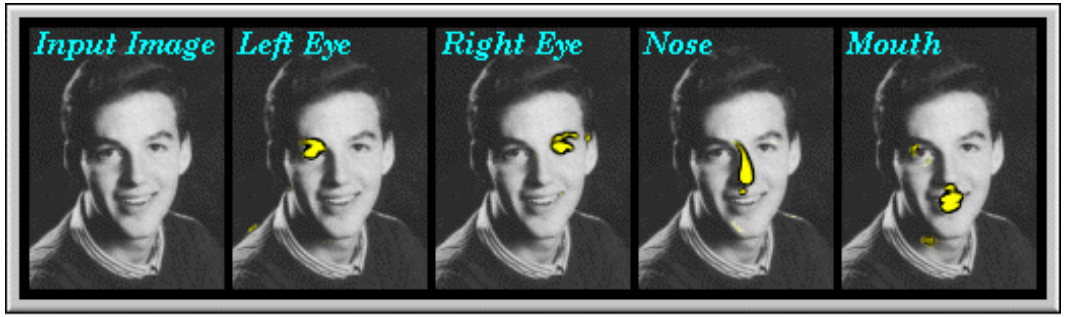
Source : https://arxiv.org/pdf/1910.07738

2) How it is used nowadays ?

- Face recognition



Li, Xiang et al. Computer Methods in Applied Mechanics and Engineering. 347. 10.1016/j.cma.2019.01.005. (2019).

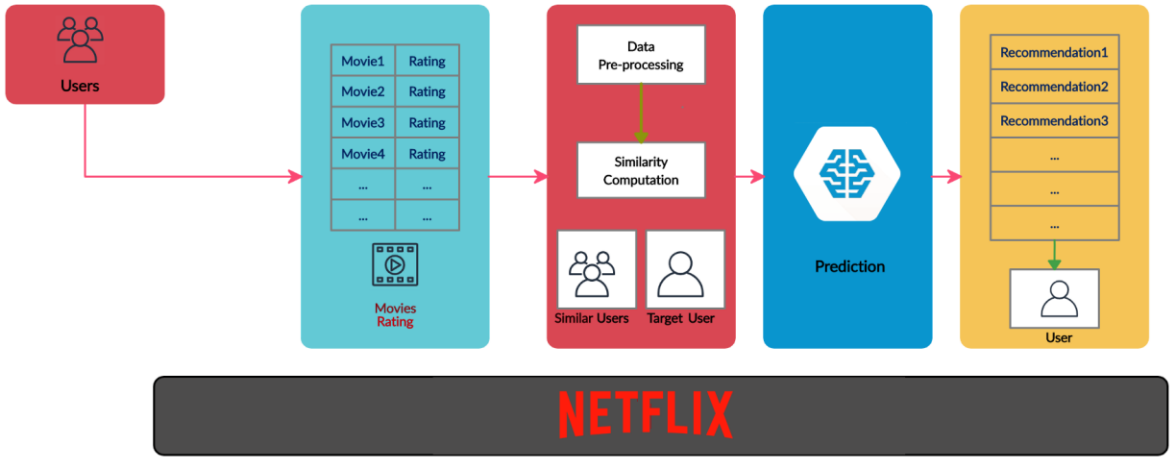


Neural network outputs for a previously unseen face

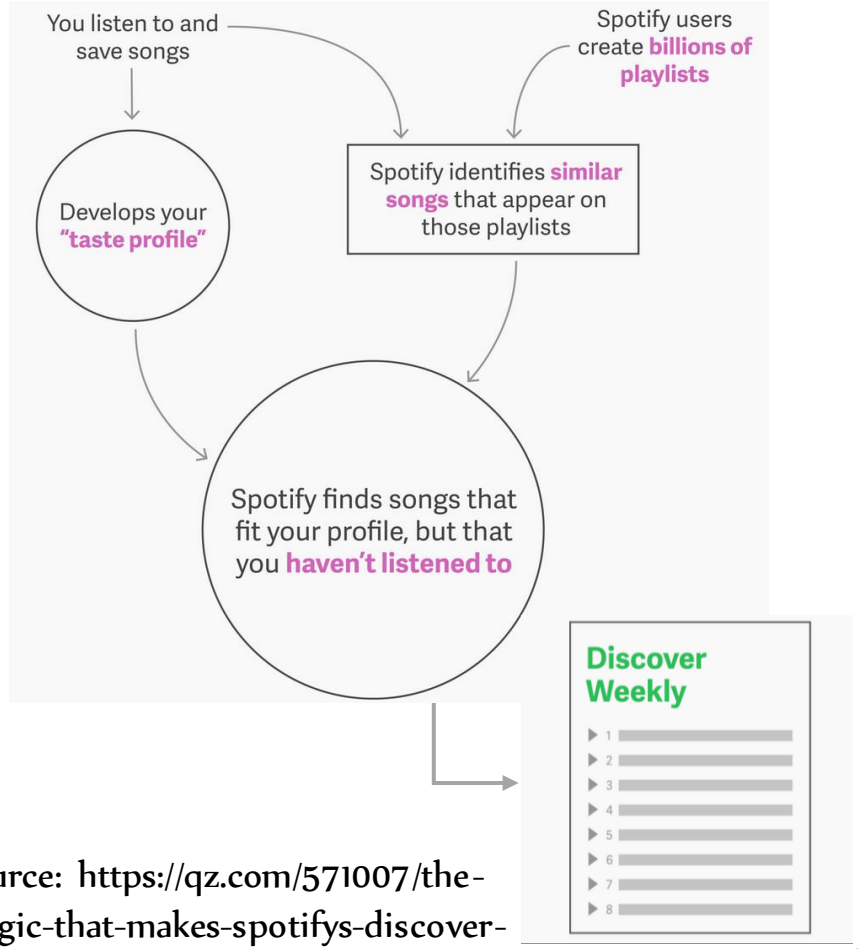
Paul Debevec. *A Neural Network for Facial Feature Location.* UC Berkeley CS283 Project Report, December 1992. http://www.debevec.org/FaceRecognition/

2) How it is used nowadays ?

- Recommendation systems (Netflix, Amazon, Spotify, ...)



"75% of what people are watching on Netflix comes from recommendations" McKinsey & Company

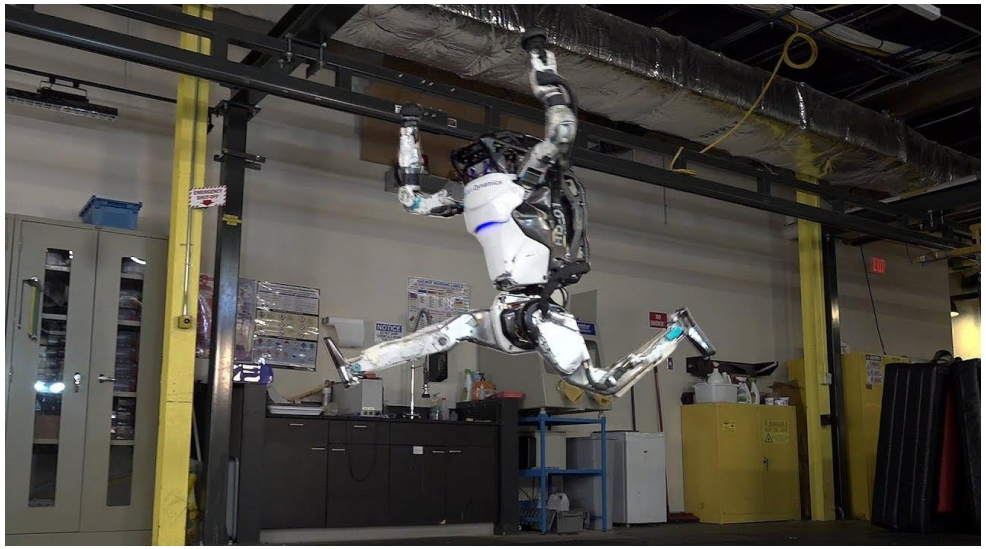


Source: https://qz.com/571007/the-magic-that-makes-spotifys-discover-weekly-playlists-so-damn-good/

2) How it is used nowadays ?

- Self-learning robots

This robot dog has an AI brain and taught itself to walk in just an hour, University of California, Berkeley



Atlas robot (Boston Dynamics) performing a sequence of dynamic maneuvers that form a gymnastic routine

2) How it is used nowadays ?

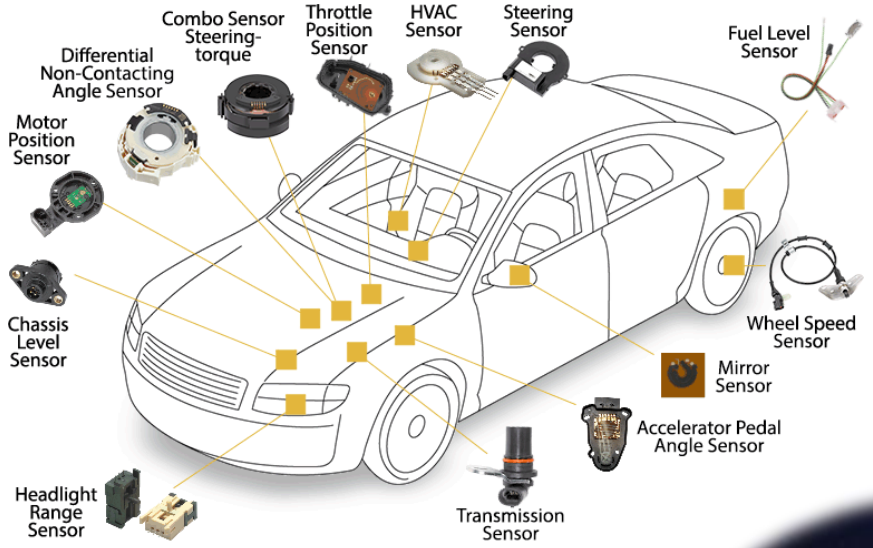
- Automatic detection of fruits/vegetables ..... and metal waste !

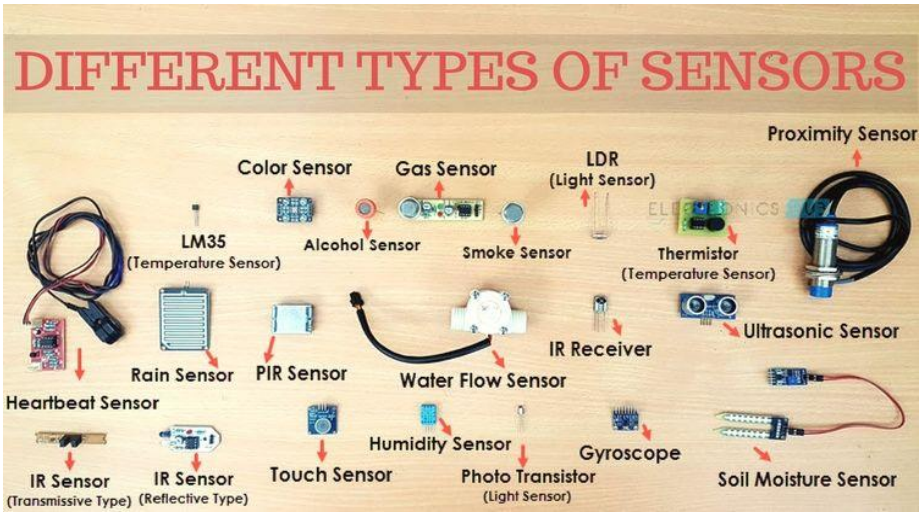
AI camera recognizes more than 120 fruits and vegetables with a 97% precision (Robovision) In .... **Courtrai/Kortrijk, Belgium** !!!

World first: metals sorted by robots. GeMMe laboratory (Faculty of Applied Sciences at **ULIEGE**) & Citius engineering & COMET Group

3) Why it will become vital in the future

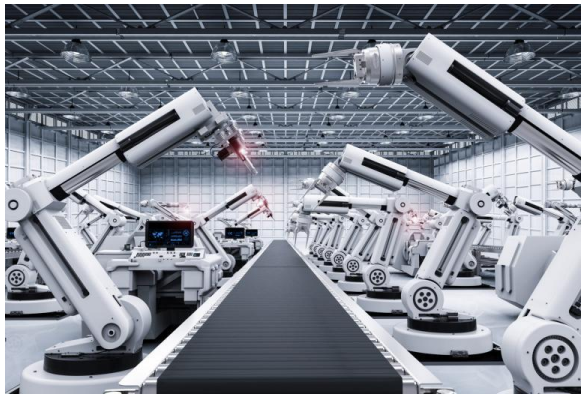
- Sensors are everywhere, data keep growing !



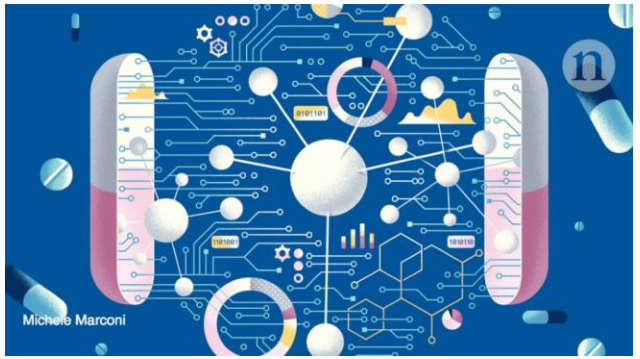


3) Why it will become vital in the future

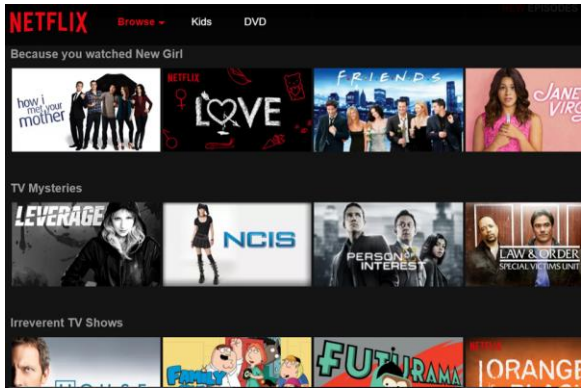
- Performance achieved by AI in many domains is now state-of-the-art



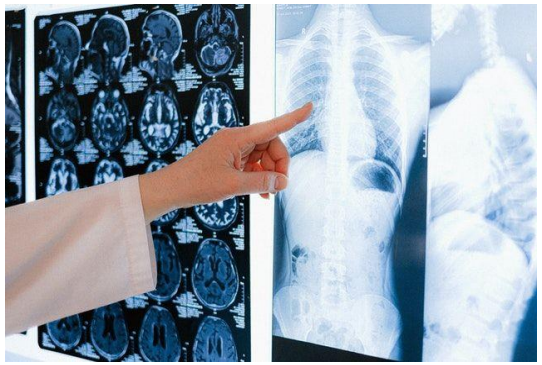
Manufacturing



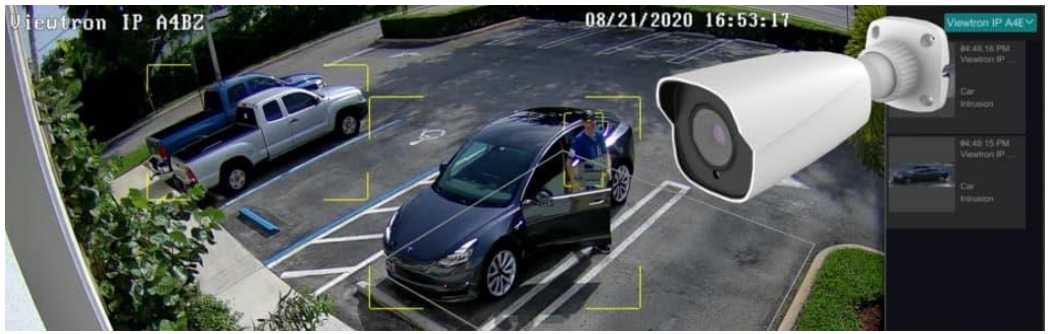
Drug discovery



Netflix



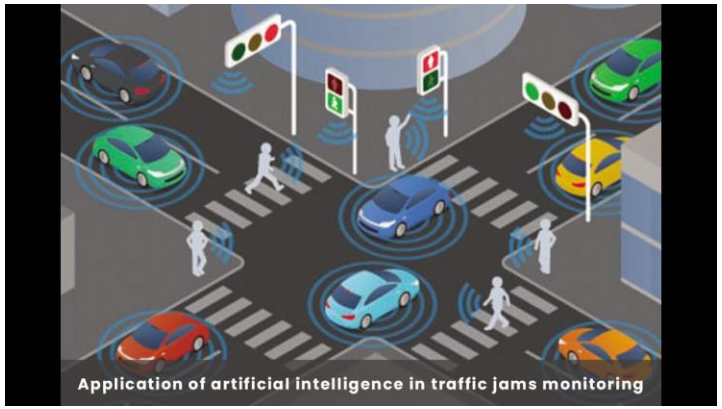
Medical diagnosis



Security Camera

3) Why it will become vital in the future

- Some problems have found a new solution



Traffic management

Automatic translation

Rapid fraud detection

Protein folding

Books digitalization

4) Objectives of this course

- This course is only an introduction but you should be able to...

- understand most of the key words/jargon of ML and DL

- be able to criticize papers in your field where deep learning is used

- understand problems that can happen during NN training

- create and train basic networks (both on tabular data, time series and images)

4) Objectives of this course

- The deep learning jargon...

GradCAM

Batch size

Overfitting >< underfitting

Convolution layers

Pooling

Vanishing gradients

Loss function

Activation function

Transfer learning

RNN

Learning rate

Normalization

Optimizer

Transposed convolution

Backpropagation

Gradient descent

Fully-connected

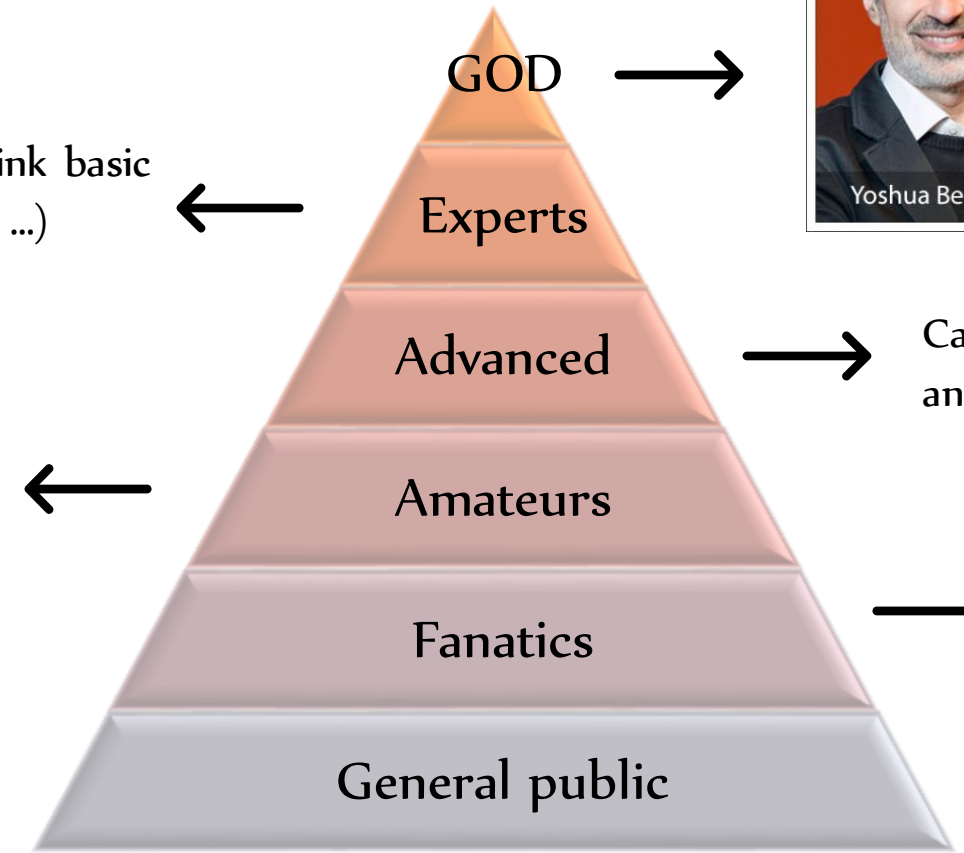
ReLU

YOLO

Initialization

4) Objectives of this course

- My pyramid of deep learning levels....



Create new architecture, rethink basic principles (Google, Microsoft, ...)

Can manipulate neural networks and adapt them to their need

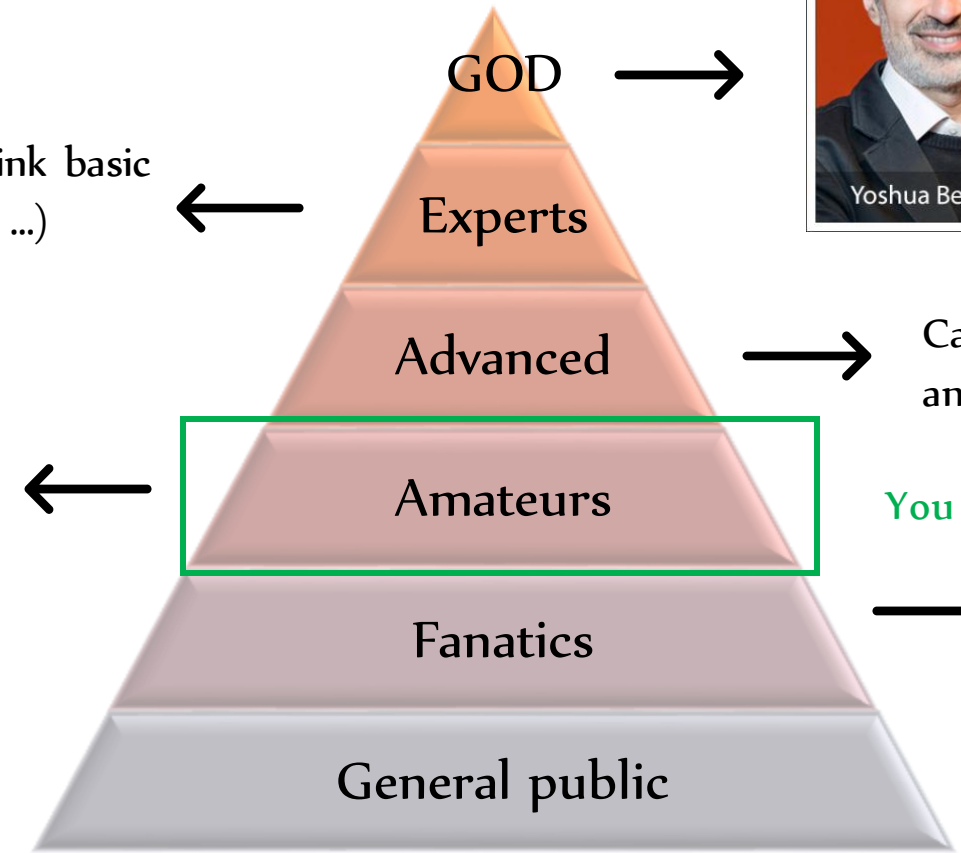
Understand basic concepts and can implement simple neural networks

Black box doing amazing things

Terminator, iRobot, self-driving cars, ...

4) Objectives of this course

- My pyramid of deep learning levels....



Create new architecture, rethink basic principles (Google, Microsoft, ...)

Can manipulate neural networks and adapt them to their need

Understand basic concepts and can implement simple neural networks

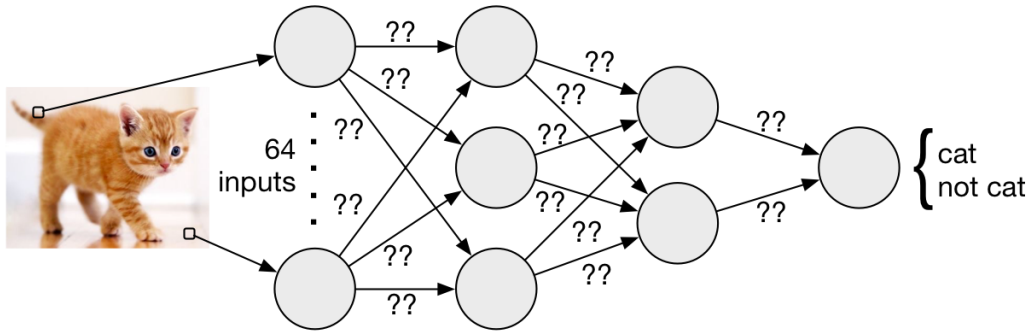
You can reach this level !

Black box doing amazing things

Terminator, iRobot, self-driving cars, ...

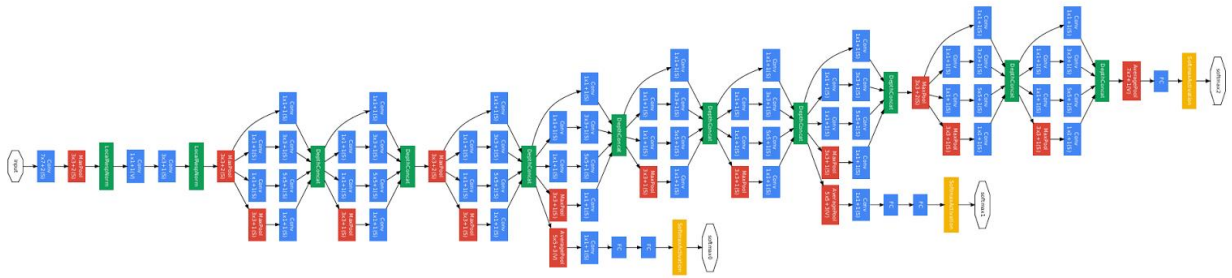
4) Objectives of this course

- You will be able to do AND understand...

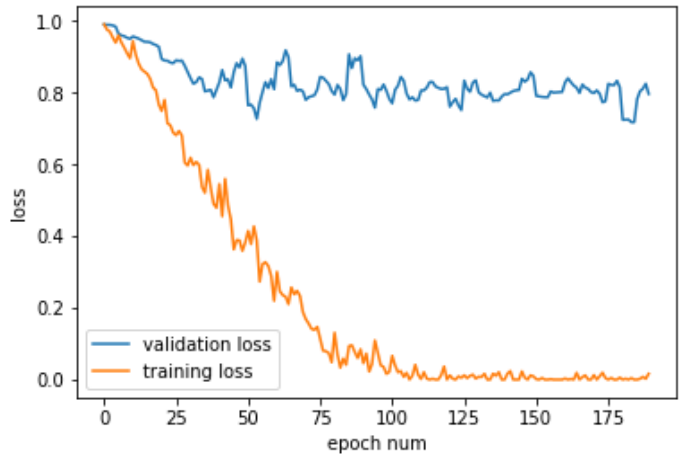


Pet
Cat
Dog
Turtle
Fish
Cat

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

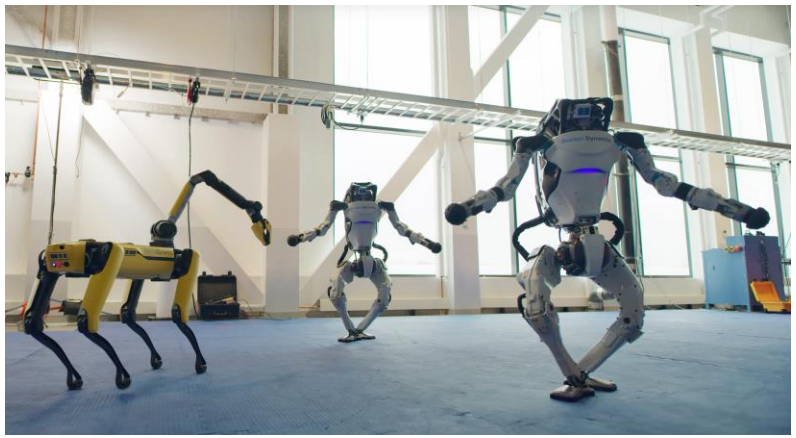


GoogLeNet (Google Brain team)



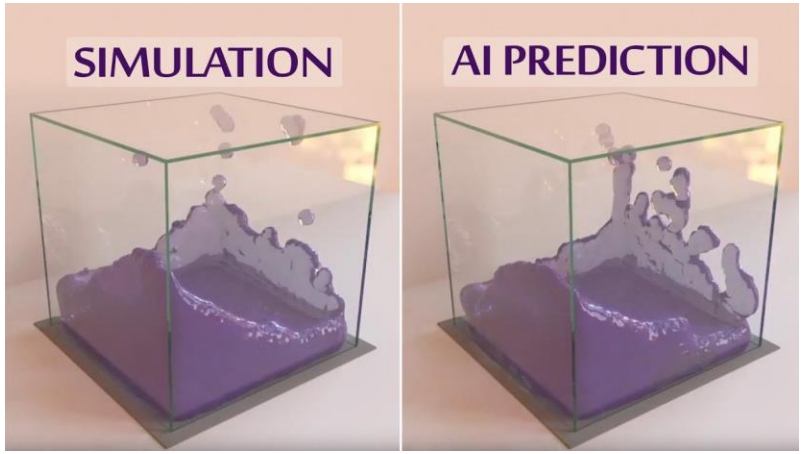
4) Objectives of this course

- You will **NOT** be able to do this…



Robot dancing
(Boston dynamics)

Faces created by using ProGAN.
These people do not exist.

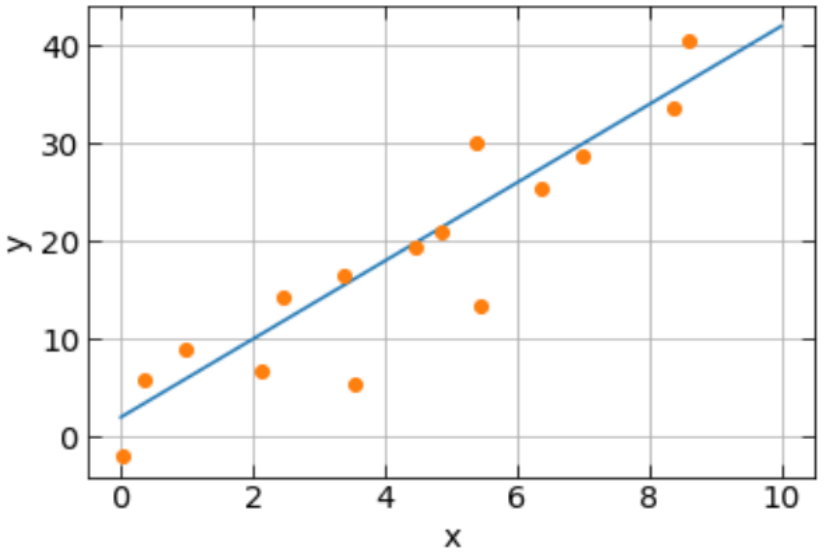


Simulate and find laws of physics
(Google DeepMind, 2min55)

5) From linear regression to neural network

- Let us settle a very simple problem: linear regressior

- You have N data points with coordinates (x,y)
- You want to fit a line to these data
- AND predict new y for new x coordinates



Graph taken from Grégory Baltus thesis

- Model : $y = w*x + b$
- Parameters : w and b

- To find the best w and b, we need a mathematical expression to assess how close is our model with respect to the existing data :

$$L = \sum_{i=0}^{N} e_i^2 = \sum_{i=0}^{N} (y_i - y_{pred,i})^2 = \sum_{i=0}^{N} (y_i - (w\,x_i + b))^2$$

5) From linear regression to neural network

- The problem becomes an optimization problem !

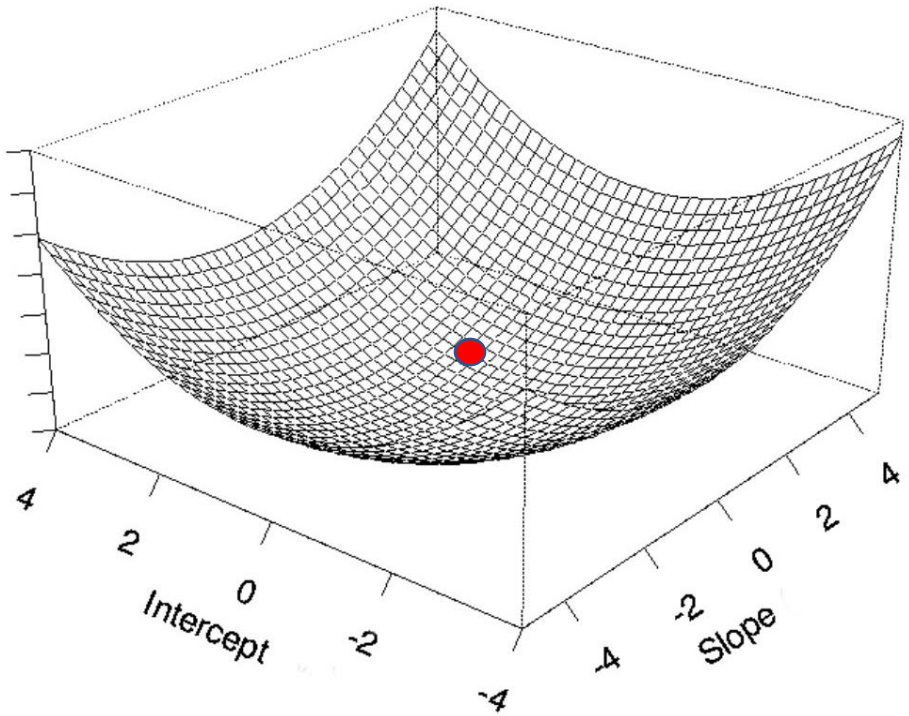
- find w and b such that $L = \sum_{i=0}^{N} (y_i - (w\,x_i + b))^2$ is minimal.

- We can solve this problem with a 3D graph

- Out of any possible combination of w and b, only 1 gives the minimum of the function.

- Local minimum = global minimum

- This is the ordinary least square regression for which analytical solutions exist (for polynomial regression)



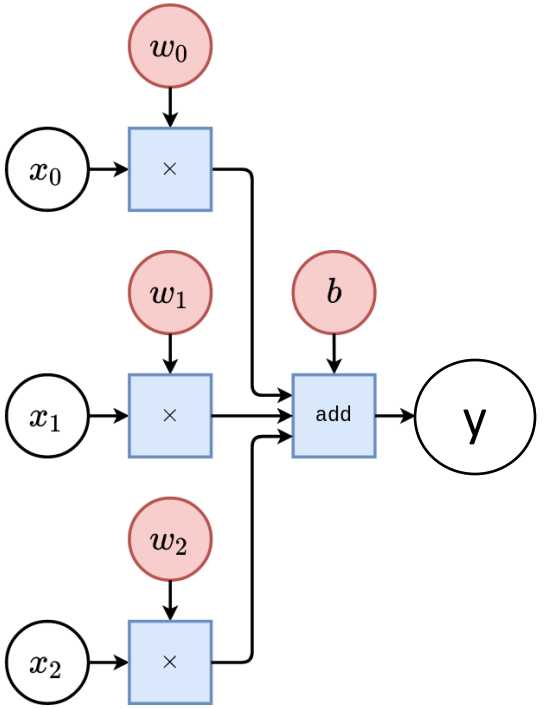
5) From linear regression to neural network

- Let us rewrite our problem… with some adaptations

- You now have N data points with coordinates (x_0, x_1, x_2, y)

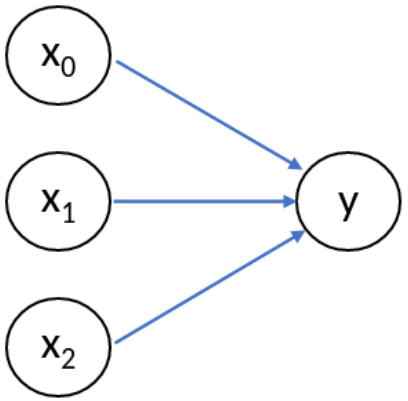
- You want to predict new y from new (x_0, x_1, x_2) couples

- Let us write our model : $$y = w_0\,x_0 + w_1\,x_1 + w_2\,x_2 + b$$ $$= W^T\,X + b$$



Adapted from Louppe, G.
Deep Learning

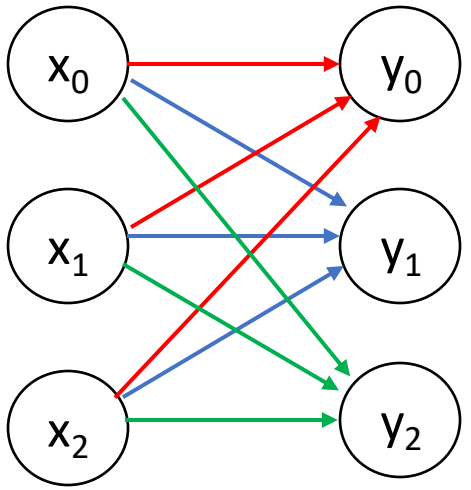
- The graph can be summarized as :



5) From linear regression to neural network

- Let us rewrite our problem... with some adaptations

- You now have N data points with coordinates (x_0, x_1, x_2, y_0, y_1, y_2)
- You want to predict new (y_0, y_1, y_2) from new (x_0, x_1, x_2) couples
- Every y might need the input from all x (namely x_0, x_1 and x_2)



- The model becomes : $Y = W^T X + B$

- fully linear
- some inputs (x) might be useless for some outputs (y)

(e.g. y_2 depends only on x_0 and x_2)

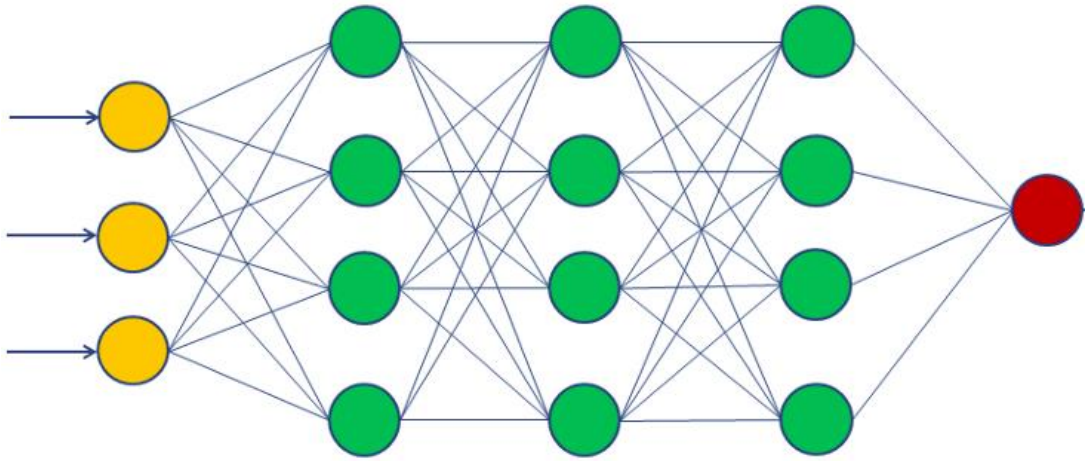
- How can we learn more complex relation between the data ?

5) From linear regression to neural network

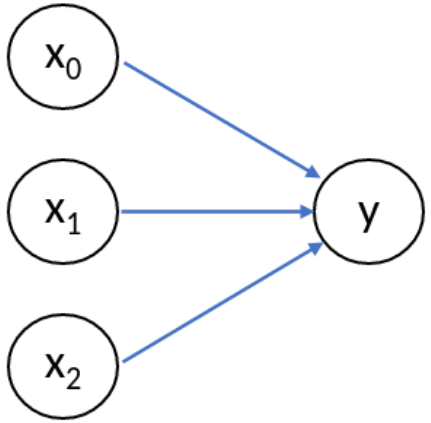
- How can we learn more complex relation between the data ?

- First guess : add more layers ?

➡ composition of linear functions = linear function



=



5) From linear regression to neural network

- How can we learn more complex relation between the data ?

- Second guess : use polynomial models

- Replace the simple linear regression with higher order models

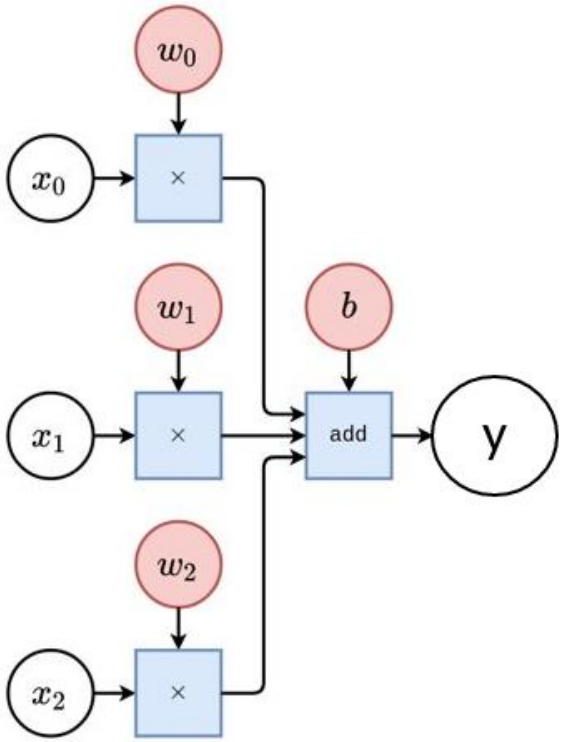
$$L = \sum_{i=0}^{N} (y_i - (w\,x_i + b))^2$$

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

- Two problems :

- require some knowledge of the solution

- computationally more expensive



Adapted from Louppe, G.
Deep Learning

5) From linear regression to neural network

- How can we learn more complex relation between the data ?

- Third guess : add some non-linearities

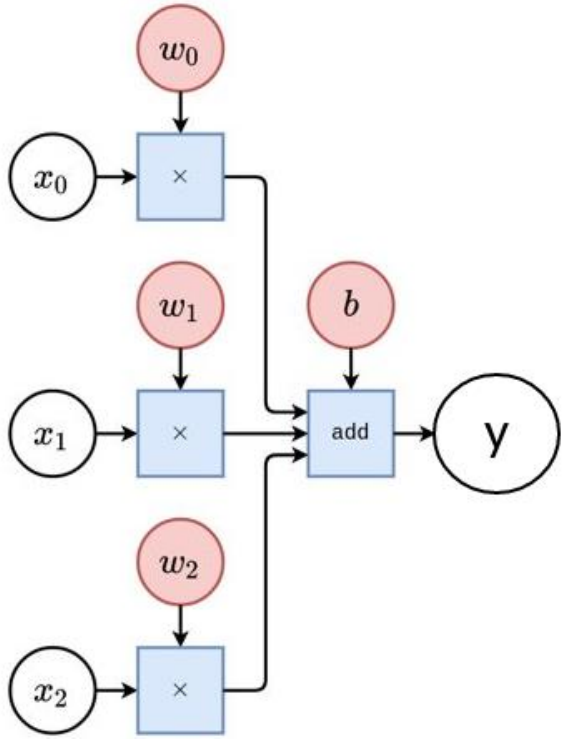
➥ Where ?

- Easy answer : at the output $Y = AF(W^T X + B)$

- we keep a simple linear relation

- and add non-linearity at the end

- This non-linear function is called an activation function

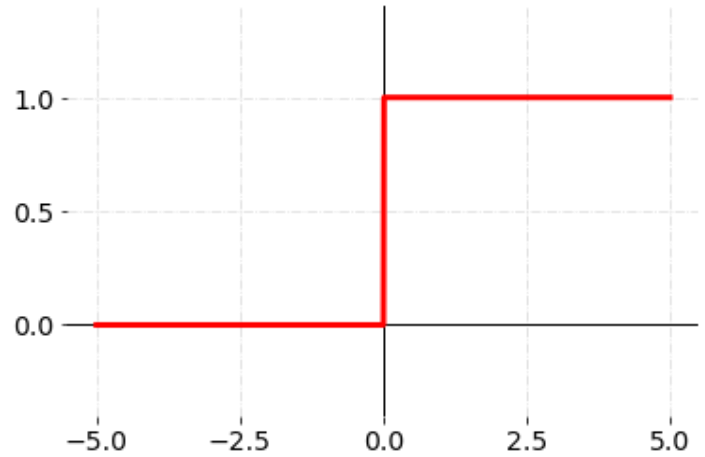


Adapted from Louppe, G.
Deep Learning

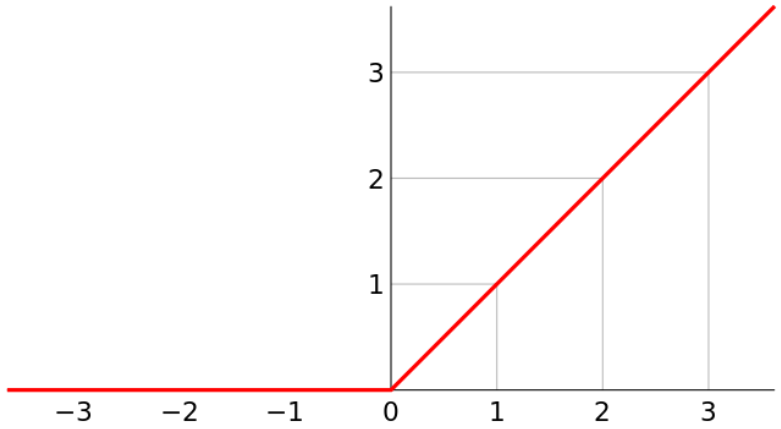
5) From linear regression to neural network

- Simplest activation function

Step function



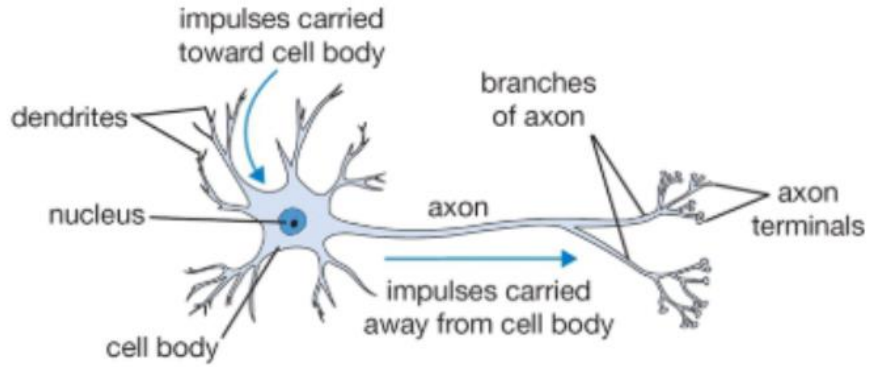
Rectified Linear Unit (ReLU)

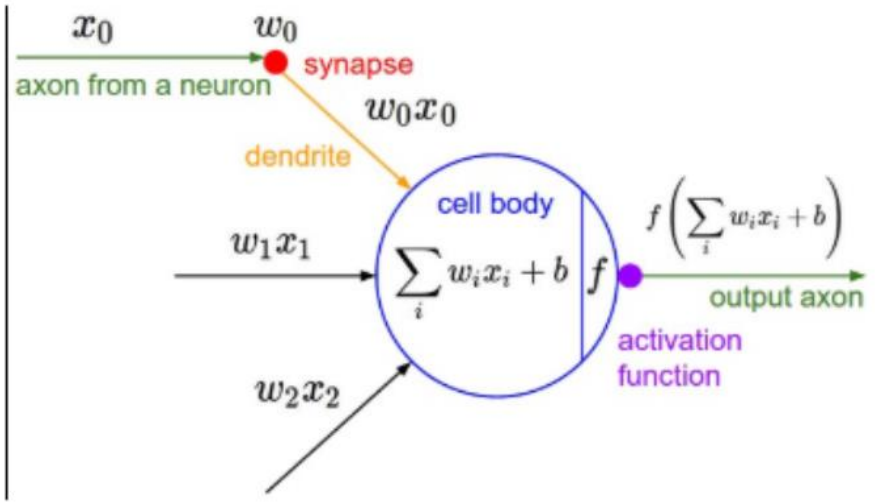


- These simple functions, added to a linear model, can lead to impressive results !

5) From linear regression to neural network

- The unit we have built is the basic mathematical formulation of a neuron





- The activation function allows to cancel the contribution from some neurons

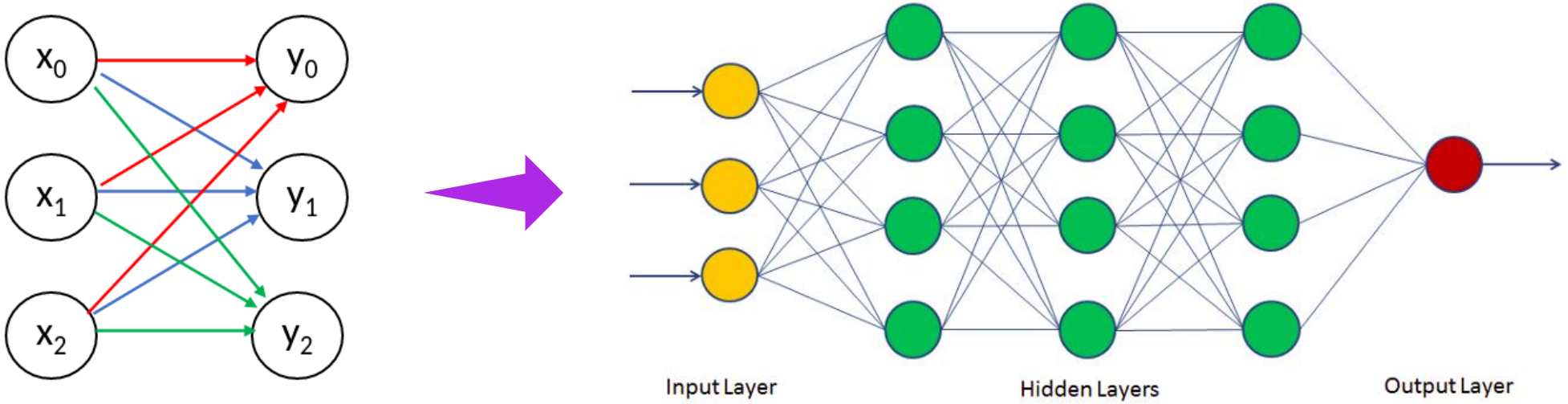
➥ Information is sometimes not relevant

5) From linear regression to neural network

- Even if our model is better than the multilinear model, it still cannot handle the underlying complexity of the relationship between inputs and output

- Let us consider the output of our model as intermediate features

- We can repeat this pattern and build new layers, with different weights (and activation ?)



5) From linear regression to neural network

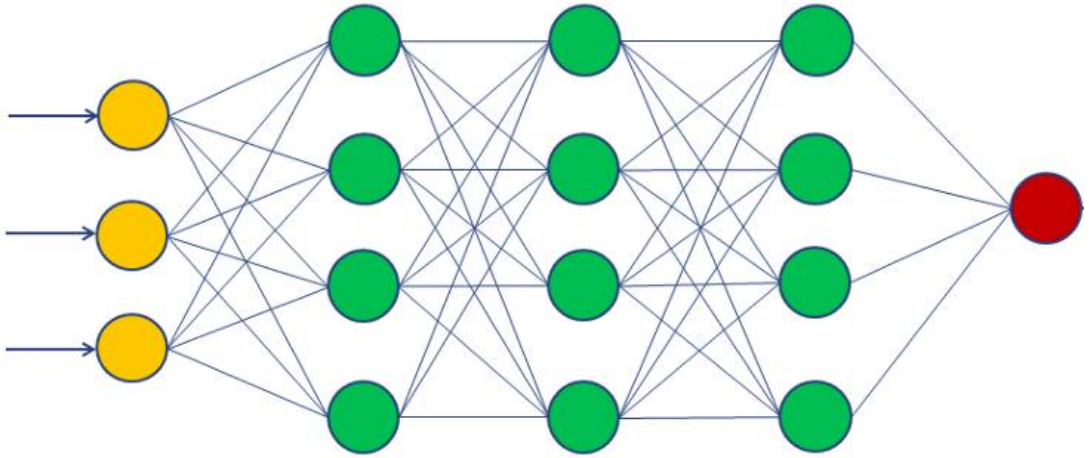
- We have built our first neural network, called the multi-layer perceptron (MLP)

- It is made up of :

- simple model to tune

- non-linearities

- multiple layers (fully-connected or linear layers)



- Disclaimer 1 : all the neural networks do not fit the template described above
- Disclaimer 2 : once defined, the model still need to be fitted (optimization)

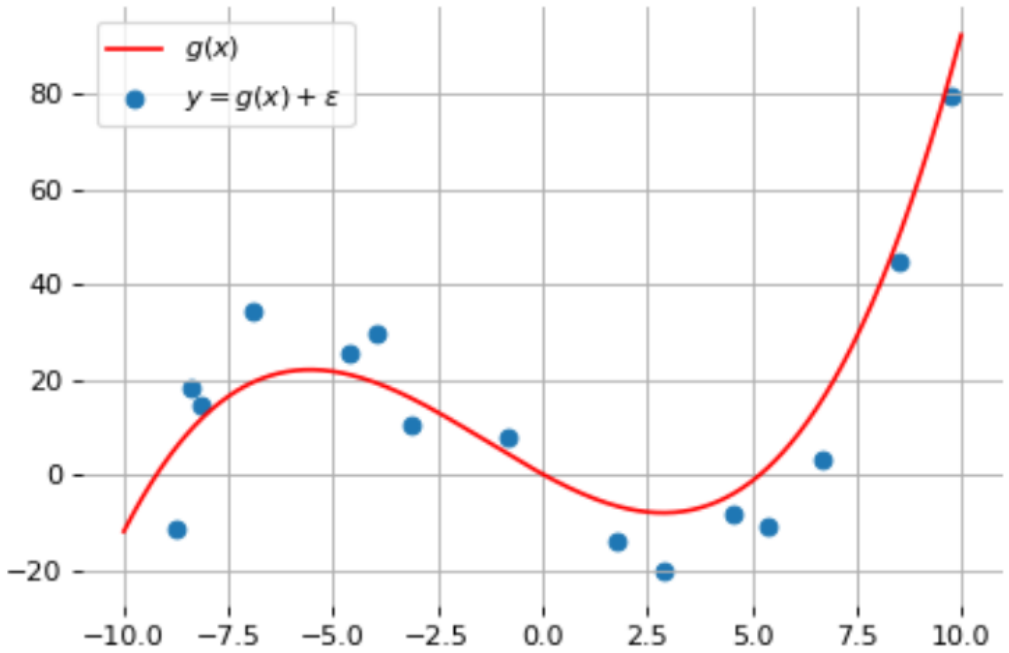
6) Analogy with the polynomial regression

- Let us consider a set of points that you want to fit with a polynomial

- you don't know the exact relation between the data

- you decide to go for "trial and error" and solve the least square regression with a polynomial of degree ***d***

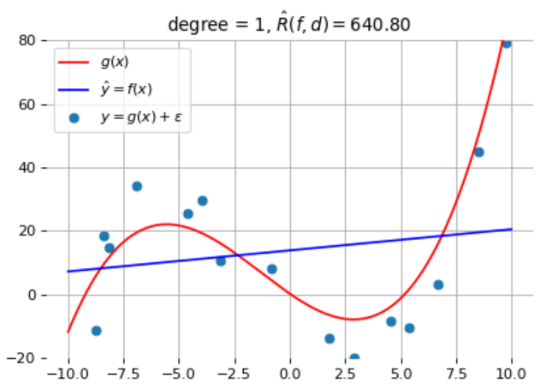
- your final choice is based on the total error

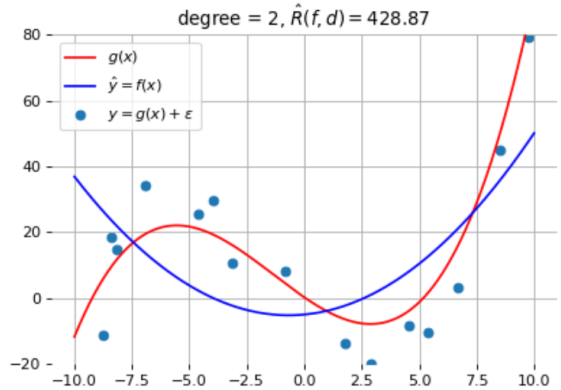


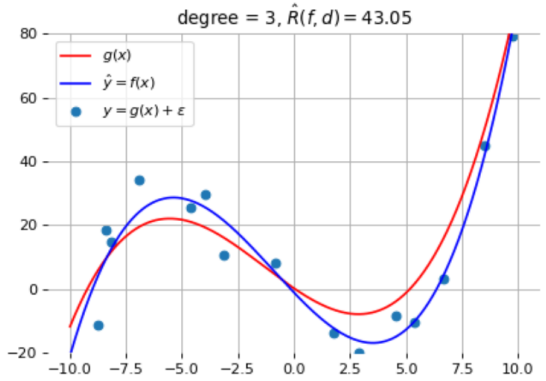
Taken from Louppe, G. *Deep Learning*

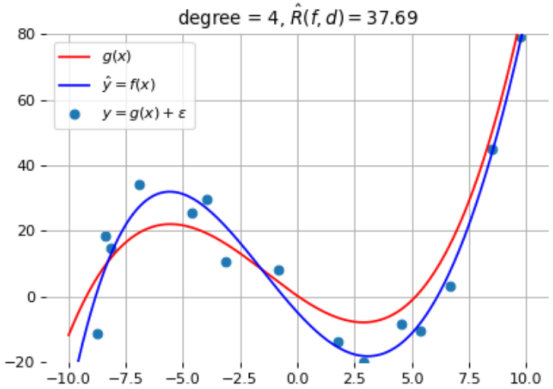
6) Analogy with the polynomial regression

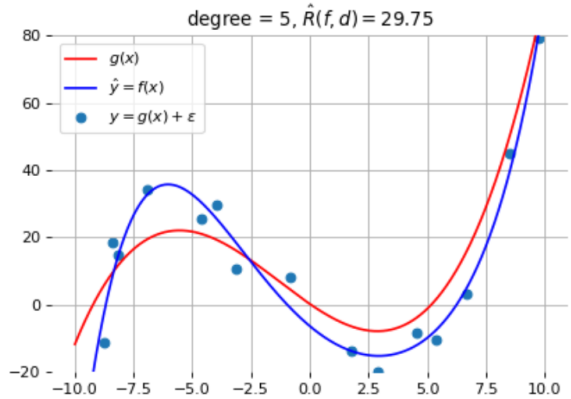
- Fit and total error for some polynomial degrees

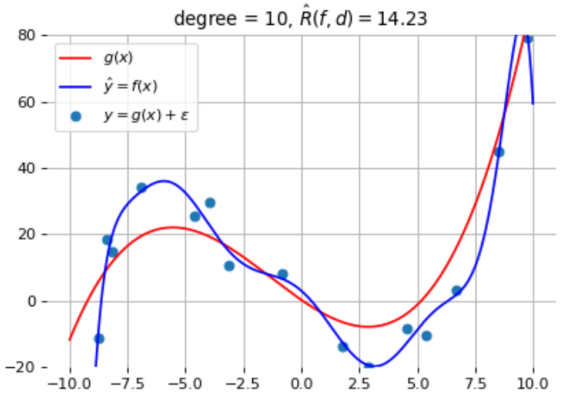








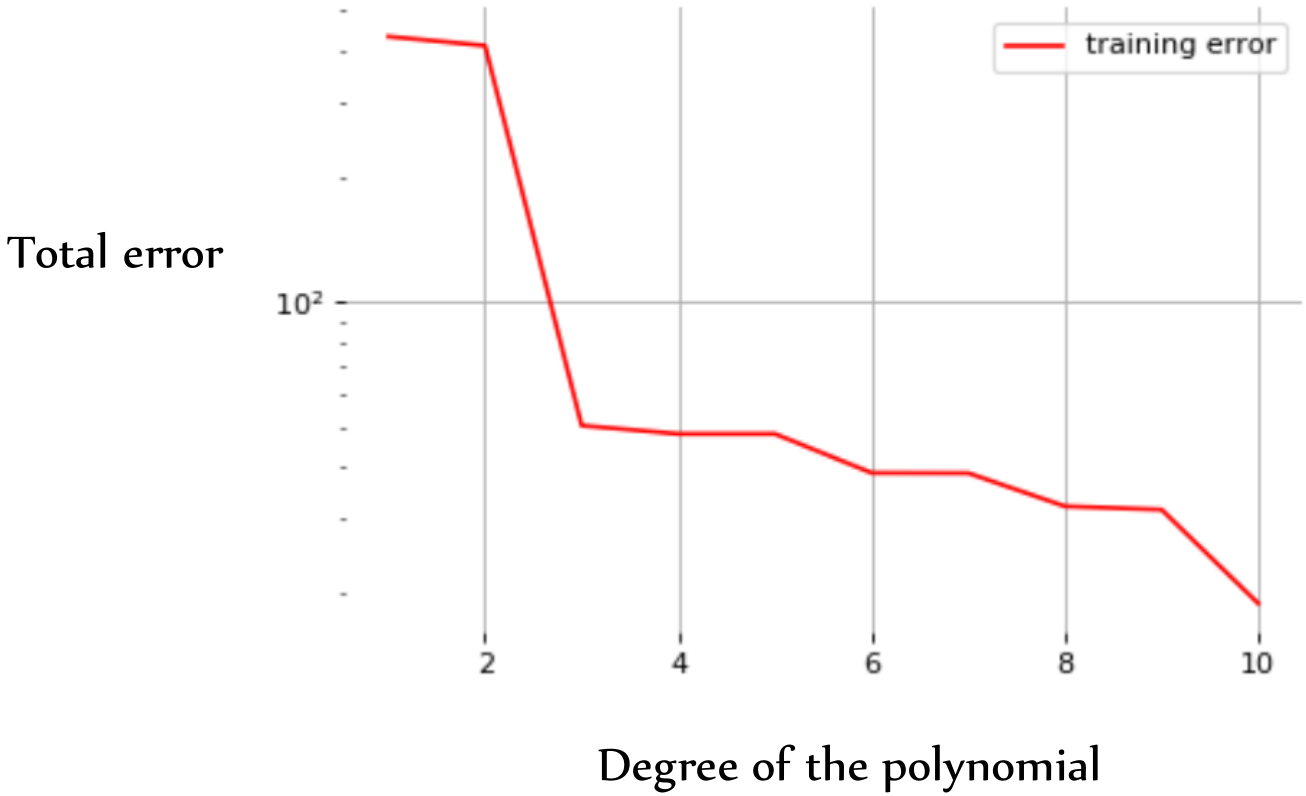




Taken from Louppe, G. *Deep Learning*

6) Analogy with the polynomial regression

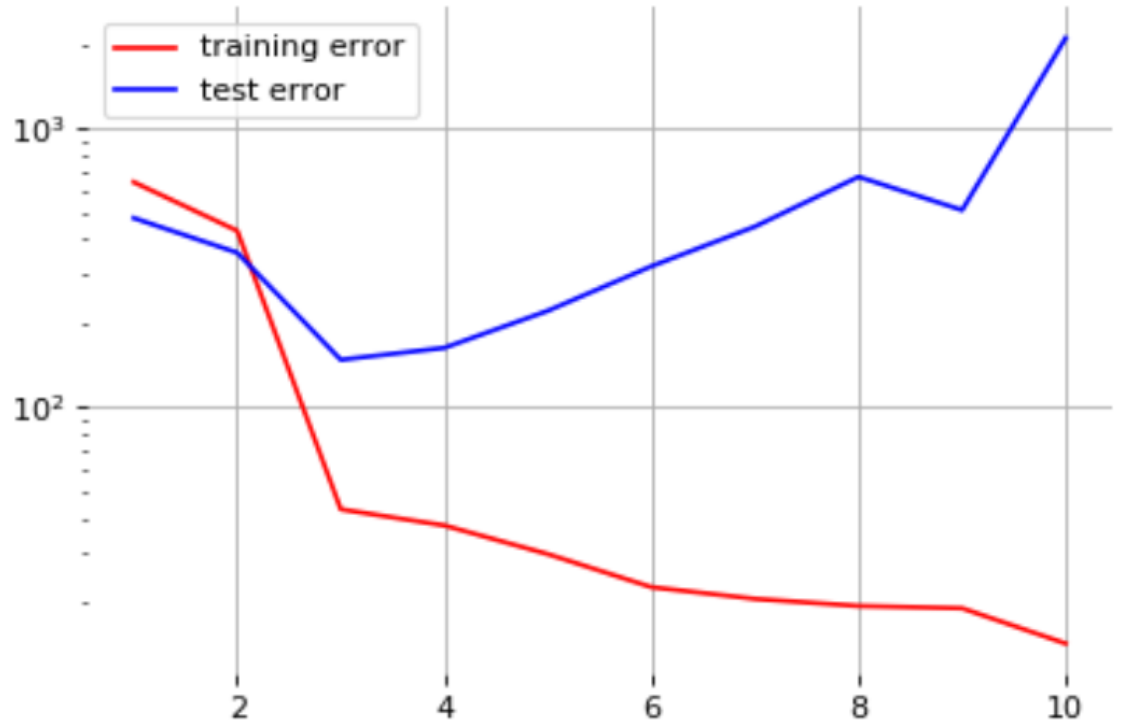
- The best solution seems achieved when ***d=10***



Taken from Louppe, G. *Deep Learning*

6) Analogy with the polynomial regression

- But when you add data points that were left aside...



Taken from Louppe, G. *Deep Learning*

6) Analogy with the polynomial regression

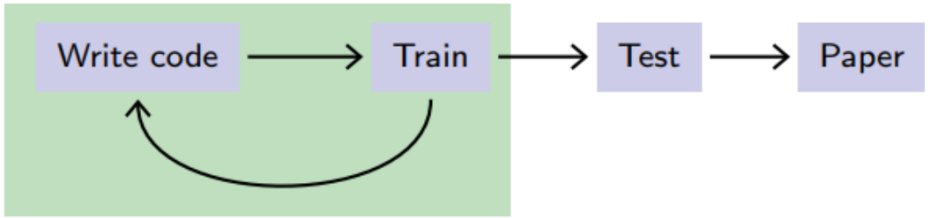
- Capacity is the ability to find a good model

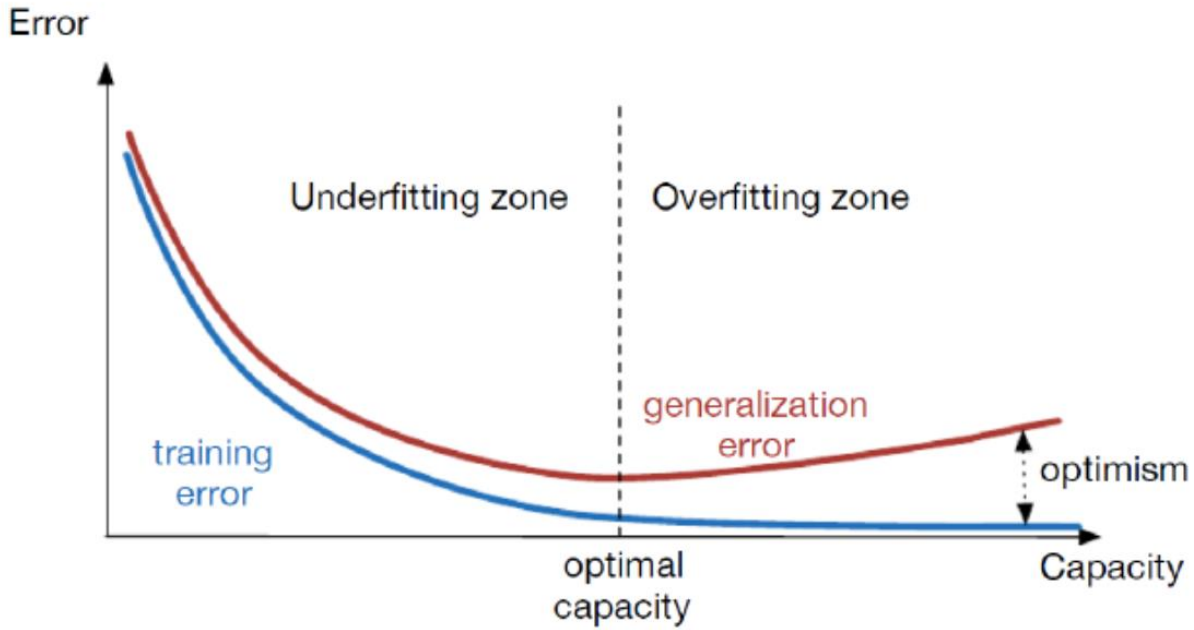
- polynomial regression : degree ***d***

- neural network : number of layers, number of training steps, regularization terms, ...

- Underfitting >< overfitting

- Separate training and testing data



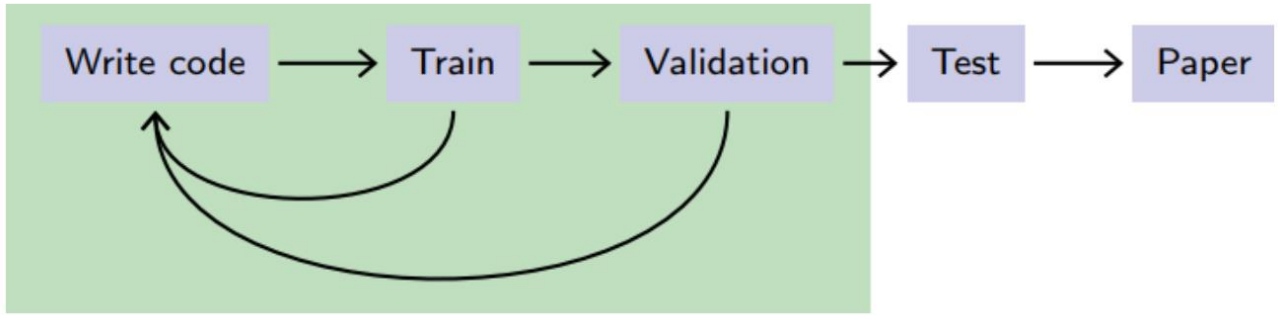


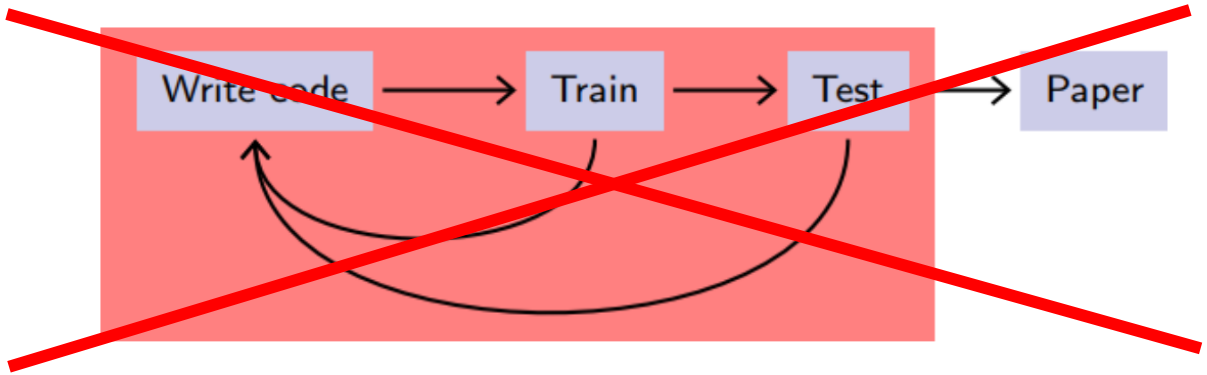
6) Analogy with the polynomial regression

- You always have to test your model ONCE the training is completed

- this will help you to understand what your network is doing

- you will learn how to interpret both training and testing curves





- The deep learning jargon...

GradCAM

Batch size

Overfitting >< underfitting

Convolution layers

Pooling

Vanishing gradients

Loss function

Activation function

Transfer learning

RNN

Learning rate

Normalization

Optimizer

Transposed convolution

Backpropagation

Gradient descent

Fully-connected

ReLU

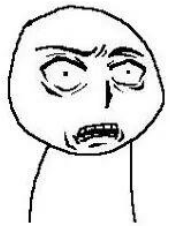
YOLO

Initialization

Course 1 : The end

Ah! With my programming skills,
I will always have a job!

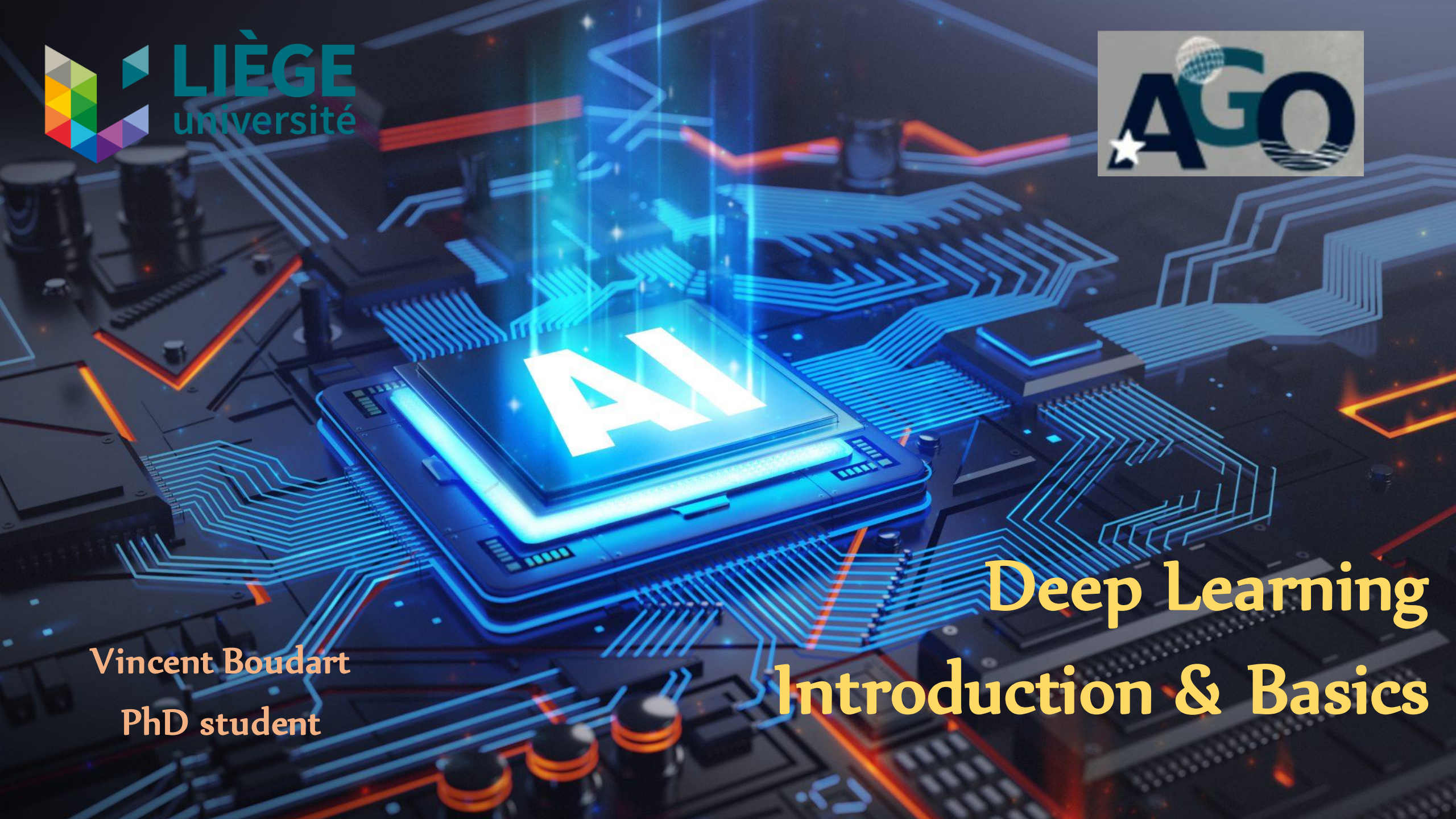
Breaking News: Machine Learning researchers managed to get an AI to write code



...

Vincent Boudart, PhD student

vboudart@uliege.be



LIÈGE université

AGO

Deep Learning
Introduction & Basics

Vincent Boudart
PhD student

Course 2 : Table of contents

1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent

- Let us consider a loss function $\mathcal{L}(\theta)$ defined over model parameters θ

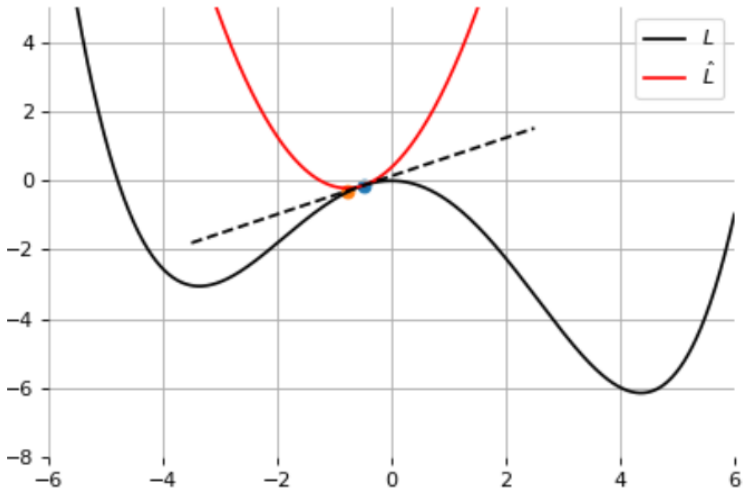
- We will use the local information to iteratively find the minimum

- Let us define a starting point θ_0

- For a small perturbation ϵ of this starting point, the loss can be written :

$$\hat{\mathcal{L}}(\epsilon;\theta_0) = \mathcal{L}(\theta_0) + \epsilon^T \nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{2\gamma}||\epsilon||^2$$

where γ is a constant that has been added



Taken from Louppe, G.
Deep Learning

1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent

- To minimize the loss approximation, one has to solve : $\nabla_\epsilon \hat{\mathcal{L}}(\epsilon; \theta_0) = 0$

$$\nabla_\theta \mathcal{L}(\theta_0) + \frac{1}{\gamma}\epsilon = 0$$

which happens when $\epsilon = -\gamma \nabla_\theta \mathcal{L}(\theta_0)$

- By repeating this "small step" process, the update rule for the model parameters is thus :

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

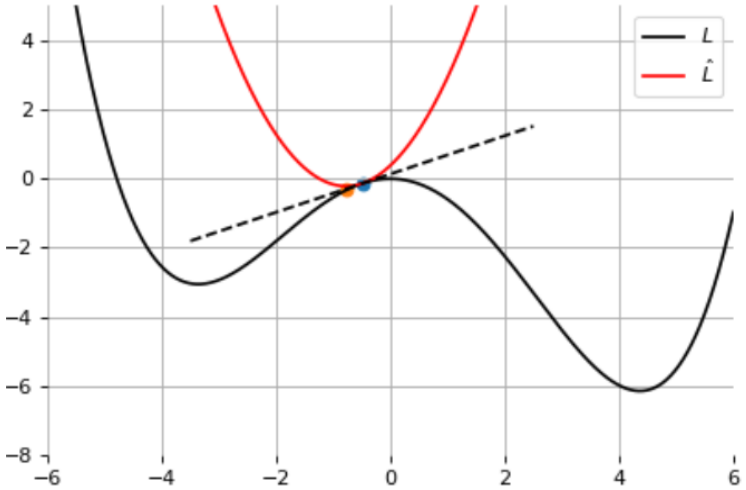
1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent

- We know how to update model parameters iteratively : $\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$

- θ_0 are the initial parameters of the model
- γ is called the learning rate

- Both parameters are critical for the convergence of the update rule !

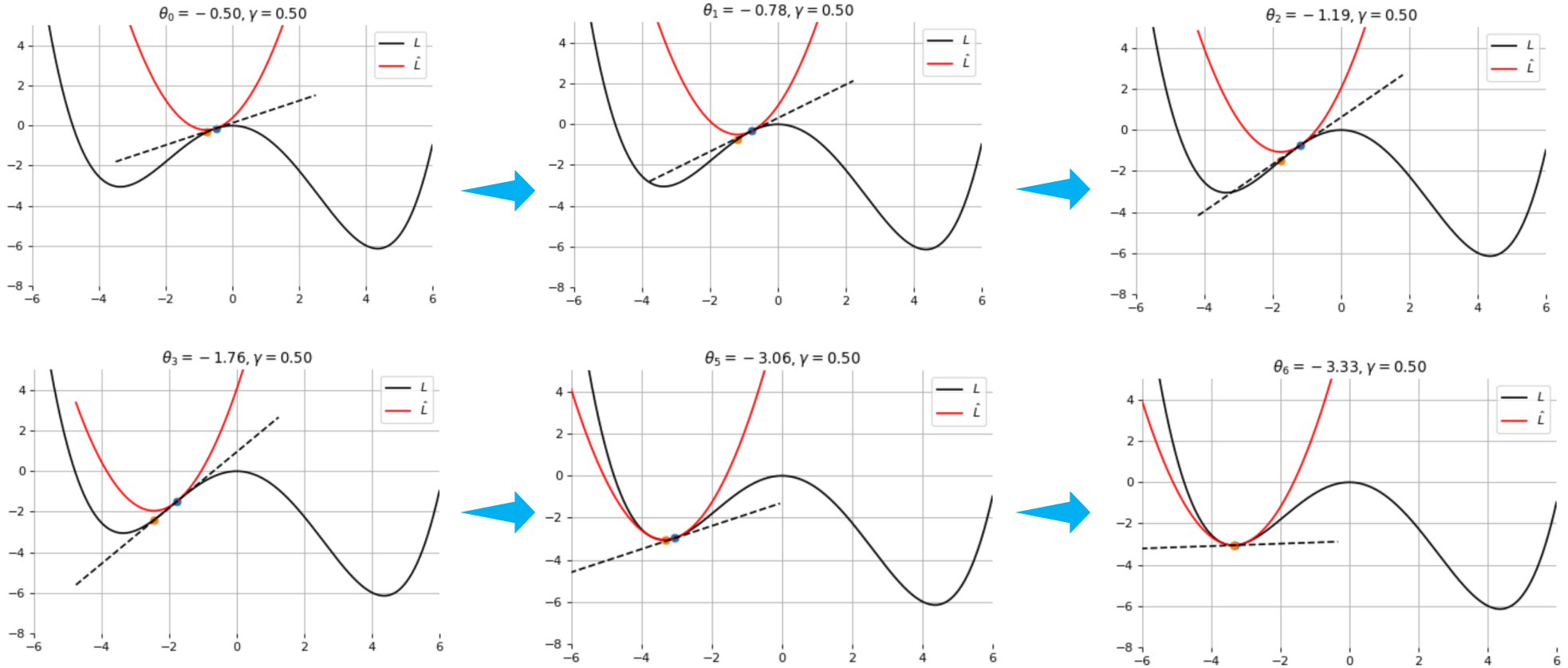


Taken from Louppe, G.
Deep Learning

1) How neural networks learn ?

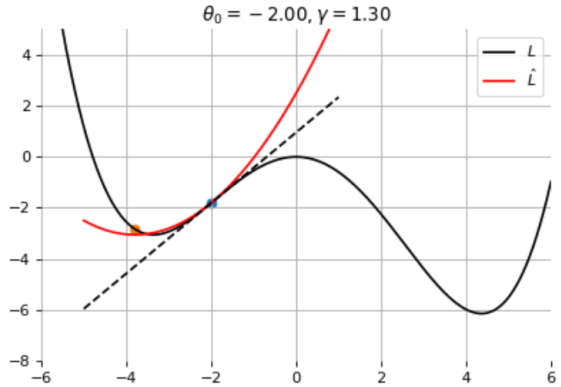
- The learning phase of a neural network is an iterative process based on gradient descent
- When the learning rate has the right value :

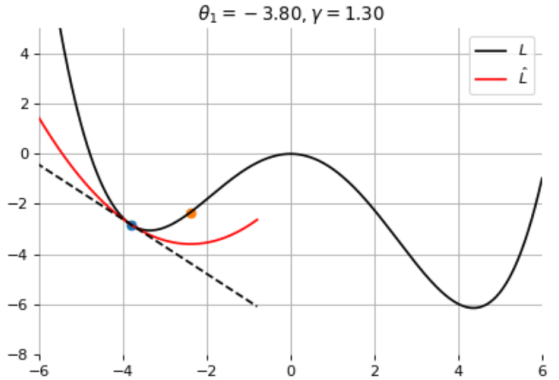
Taken from Louppe, G. *Deep Learning*

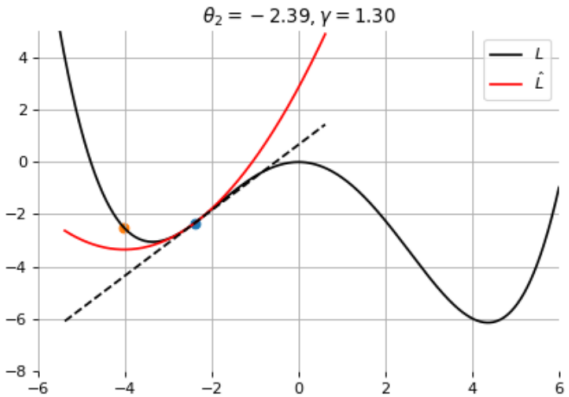


1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent
- When the learning rate is too high :







Taken from Louppe, G.
Deep Learning

1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent

- How do we know if the feedback from the gradients is accurate ? $\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$

- training a NN means finding the minimum of your loss function **over your data**

➡ Ideal solution : compute the loss **over all the samples** in the training set

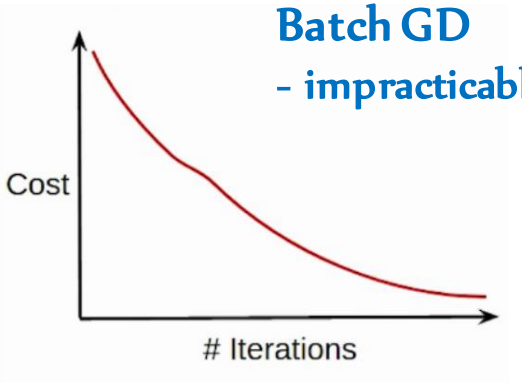
⚠ In practice you have thousands/millions of samples, leading to memory overload !

- Solution 1 : evaluate the loss over every single sample : stochastic gradient descent

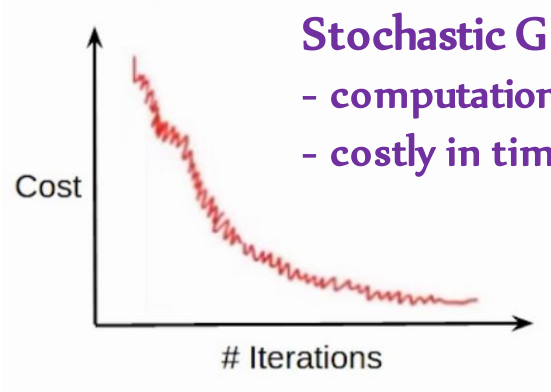
- Solution 2 : evaluate the loss over a small subset : mini-batch gradient descent

1) How neural networks learn ?

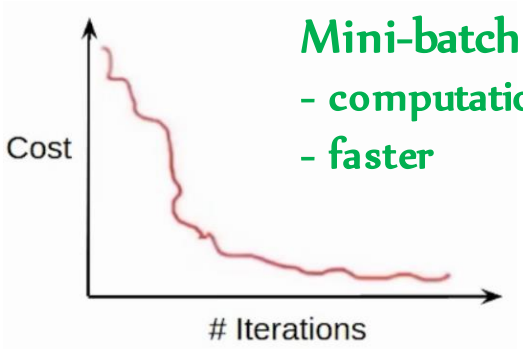
- The learning phase of a neural network is an iterative process based on gradient descent
- Comparison between stochastic and mini-batch gradient descent



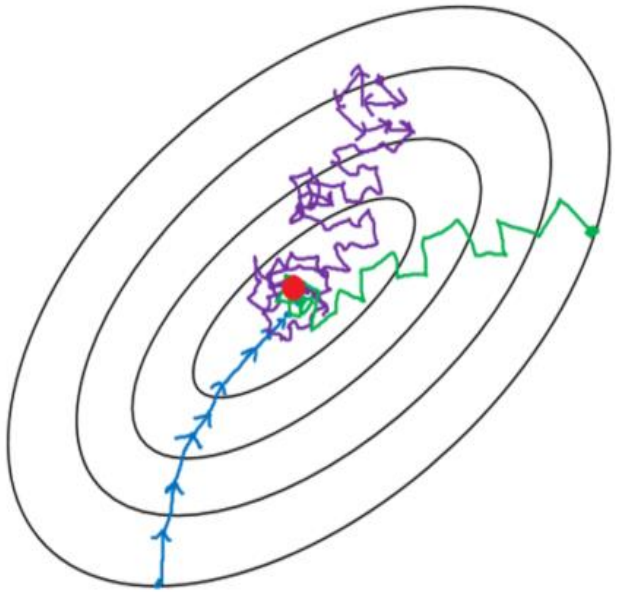
Batch GD
- impracticable



Stochastic GD
- computationally cheap
- costly in time



Mini-batch GD
- computationally heavier
- faster



1) How neural networks learn ?

- The learning phase of a neural network is an iterative process based on gradient descent

➥ We will use mini-batch gradient descent as a good compromise

- At this point, we know how to update the model parameters $\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$

- How do we evaluate the gradients ?

- gradient for one parameter : $\nabla \mathcal{L} = \frac{d\mathcal{L}}{d\theta}$

- gradient for multiple parameters : $\nabla \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial \theta_0}, \frac{\partial \mathcal{L}}{\partial \theta_1}, ..., \frac{\partial \mathcal{L}}{\partial \theta_{K-1}} \right]$

1) How neural networks learn ?

- The procedure to evaluate partial derivatives in a NN is called backpropagation
- To introduce this concept, let us define a very simple NN with 2 hidden layers

- Let us define the forward pass

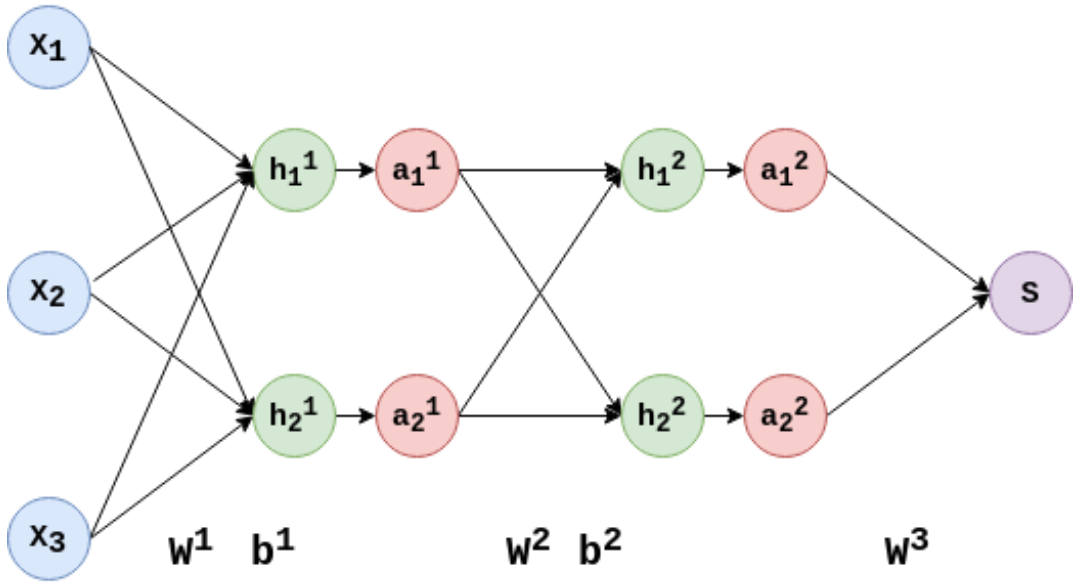
- First layer : $h^1 = W^1 x + b^1$

$a^1 = f(h^1)$

- Second and third layer : $h^2 = W^2 a^1 + b^2$

$a^2 = f(h^2)$

$S = W^3 a^2$



$\mathcal{L} = loss(y, S)$

1) How neural networks learn ?

- The procedure to evaluate partial derivatives in a NN is called backpropagation
- The forward pass expressions and matrices :

$$h^1 = W^1 x + b^1 \qquad h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} \qquad W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix} \qquad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}$$

$$a^1 = f(h^1) \qquad a^1 = \begin{bmatrix} a_1^1 \\ a_2^1 \end{bmatrix}$$

$$h^2 = W^2 a^1 + b^2 \qquad h^2 = \begin{bmatrix} h_1^2 \\ h_2^2 \end{bmatrix} \qquad W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{bmatrix} \qquad b^2 = \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix}$$

$$a^2 = f(h^2) \qquad a^2 = \begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}$$

$$S = W^3 a^2 \qquad W^3 = \begin{bmatrix} w_1^3 & w_2^3 \end{bmatrix}$$

Total : 16 parameters

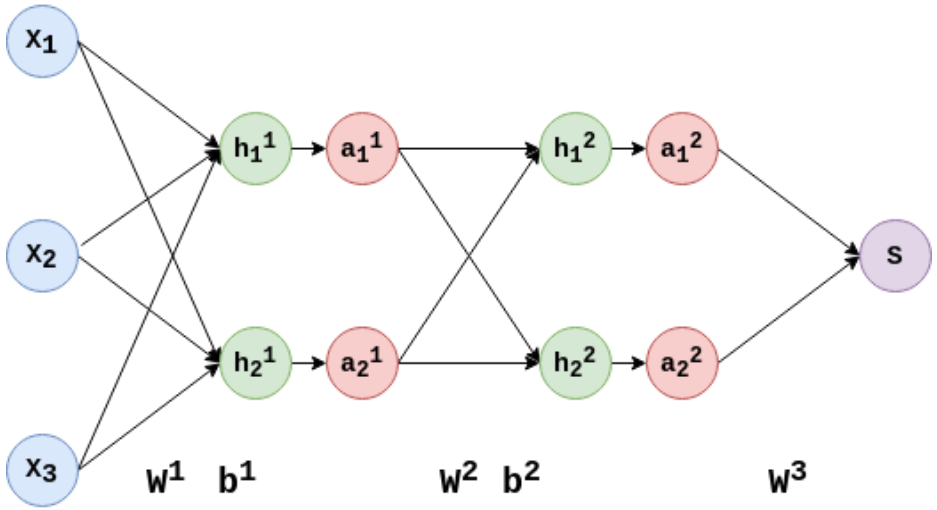
1) How neural networks learn ?

- The procedure to evaluate partial derivatives in a NN is called backpropagation
- Let us try to evaluate the partial derivative of the loss with respect to one parameter

- The chain rule of partial derivatives

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_0} = \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}} \underbrace{\frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_0}}_{\text{recursive case}}$$

$$= \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_{t-1}} \frac{\partial \mathbf{x}_{t-1}}{\partial \mathbf{x}_{t-2}} \cdots \frac{\partial \mathbf{x}_2}{\partial \mathbf{x}_1} \frac{\partial \mathbf{x}_1}{\partial \mathbf{x}_0}$$



- A neural network is a composition of very simple functions !

$$S = W^3 \left(f\left(W^2 \, f(W^1 \, x + b^1) + b^2 \right) \right)$$

1) How neural networks learn ?

- The procedure to evaluate partial derivatives in a NN is called backpropagation
- Let us try to evaluate $\frac{\partial \mathcal{L}}{\partial w_{22}^2}$

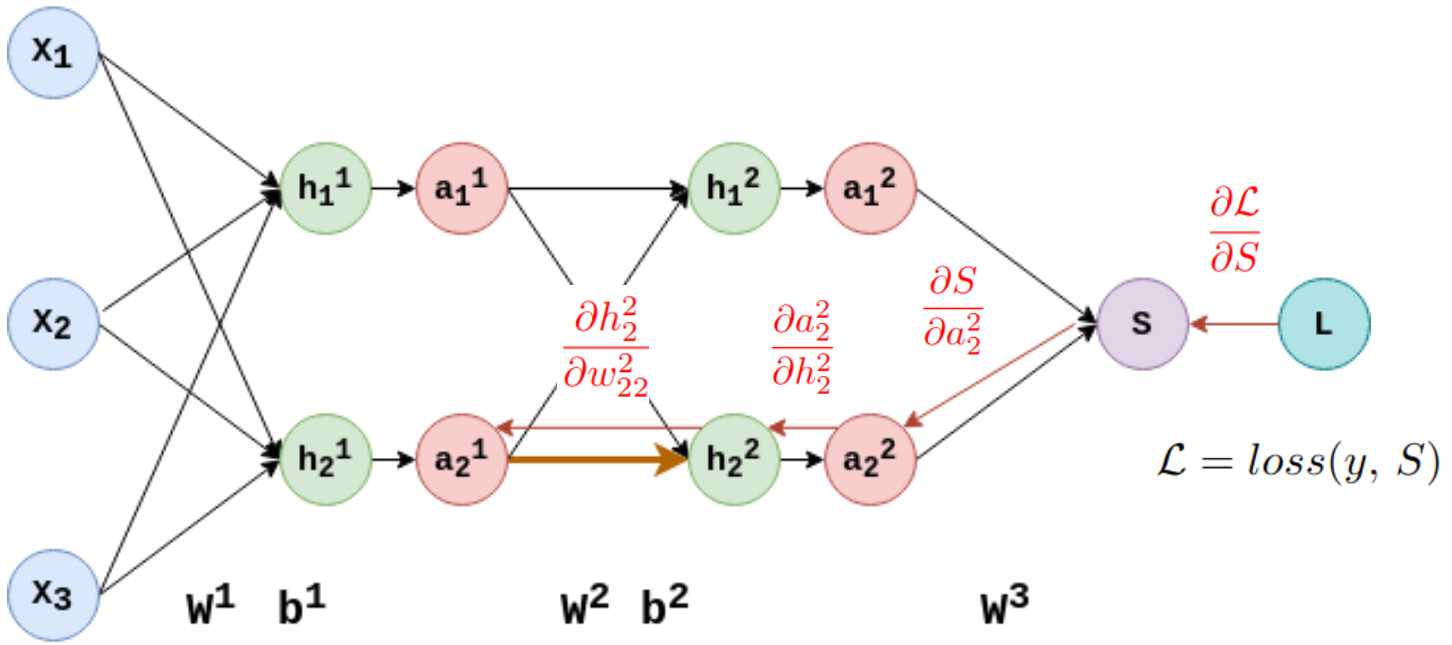
$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial h_2^2} \frac{\partial h_2^2}{\partial w_{22}^2}$$

$$= \frac{\partial \mathcal{L}}{\partial a_2^2} \frac{\partial a_2^2}{\partial h_2^2} \frac{\partial h_2^2}{\partial w_{22}^2}$$

$$= \frac{\partial \mathcal{L}}{\partial S} \frac{\partial S}{\partial a_2^2} \frac{\partial a_2^2}{\partial h_2^2} \frac{\partial h_2^2}{\partial w_{22}^2}$$

$$= \frac{\partial \mathcal{L}}{\partial S} \; w_2^3 \; f'(h_2^2) \; a_2^1$$

- $\frac{\partial \mathcal{L}}{\partial S}$: depends on the loss function
- w_2^3 : known
- $f'(h_2^2)$: easy to compute
- a_2^1 : known



1) How neural networks learn ?

- Gradient descent and backpropagation are the core of deep learning algorithms

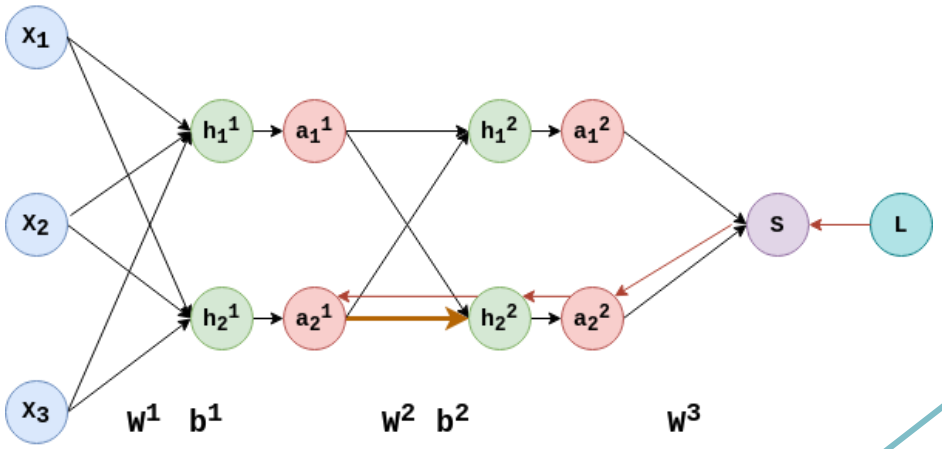
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t) \qquad \frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \quad w_2^3 \quad f'(h_2^2) \quad a_2^1$$

- What have we learned ?

- Backpropagation is cheap since we know some of the terms thanks to the forward pass !
- The other terms are easy to compute if we choose activation and loss functions adequately
- The weights **cannot** be initialized to zero

- What is left ?

- How to choose the loss function ?
- How to initialize the weights ?



1) How neural networks learn ?

- Let us have a look at the values of the different terms

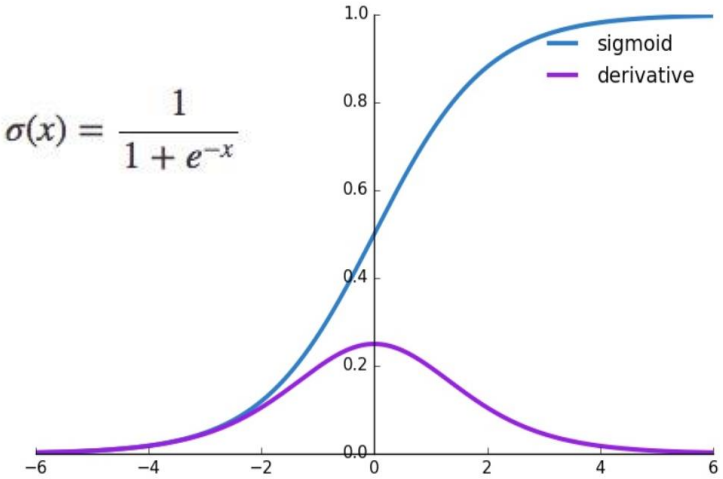
- sigmoid as the activation function

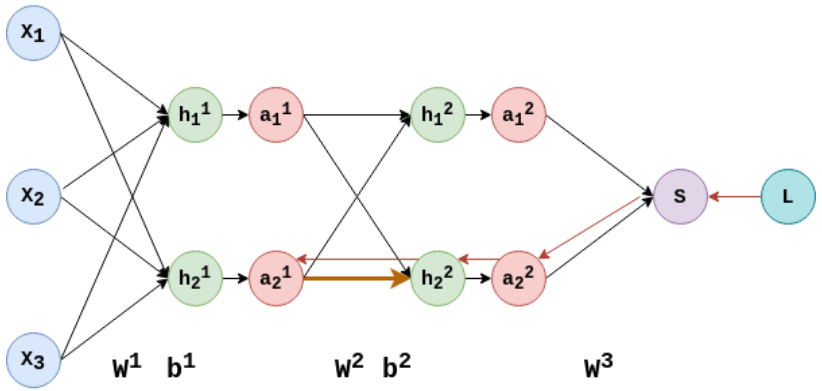
- weights are initialized randomly from a Gaussian $\mathcal{N}(0,1)$

$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \quad w_2^3 \quad f'(h_2^2) \quad a_2^1$$

$-1 \leq w \leq 1 \qquad \leq 0.25 \qquad \leq 1$

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





- Gradients shrink to zero as the number of layers grows !

➥ vanishing gradient problem

2) A word about activation functions

- Activation function brings non-linearities to neural networks

- they help constraining the output of neurons

- they influence the network's capacity to learn and converge

- To be a good activation function (AF), you need to :

- be efficient : an AF should reduce the computation time

- be differentiable (almost everywhere)

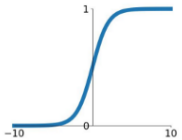
➥ Consequence of backpropagation

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

- This is ok ! In practice, the values never reach exactly zero !

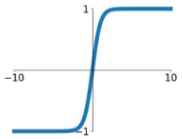
Sigmoid

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



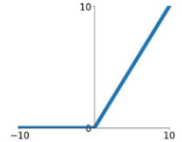
tanh

$\tanh(x)$



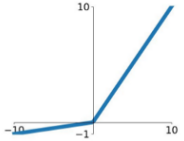
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

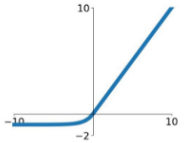


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



2) A word about activation functions

- Bounded activation functions are prone to vanishing gradient

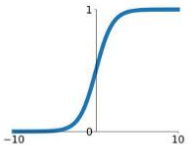
- That is why the ReLU has been introduced !

$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \quad w_2^3 \quad f'(h_2^2) \quad a_2^1$$

$-1 \leq w \leq 1$ $\quad = 1$ $\quad = 1$

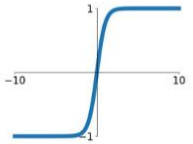
Sigmoid

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



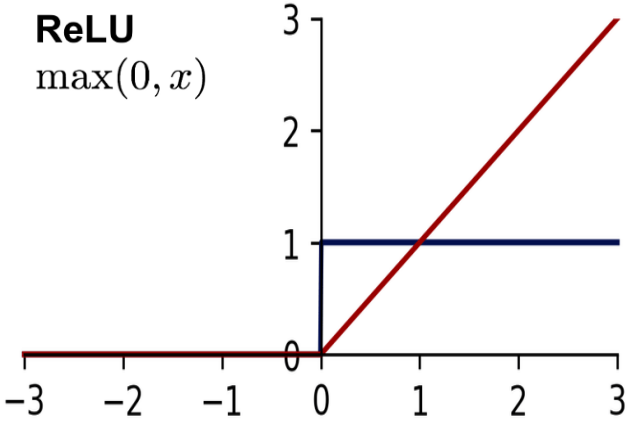
tanh

$\tanh(x)$



ReLU

$\max(0, x)$

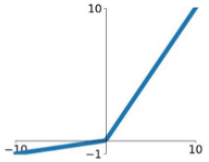


- This is a useful property to induce sparsity

- Other activation functions have been proposed to solve sparsity

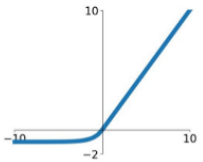
Leaky ReLU

$\max(0.1x, x)$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



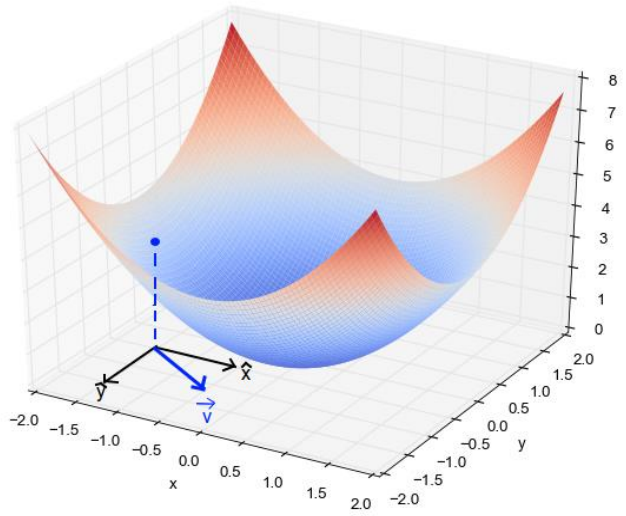
3) How to train neural networks ?

- Training a NN consists in applying a strategy to update the weights of your NN

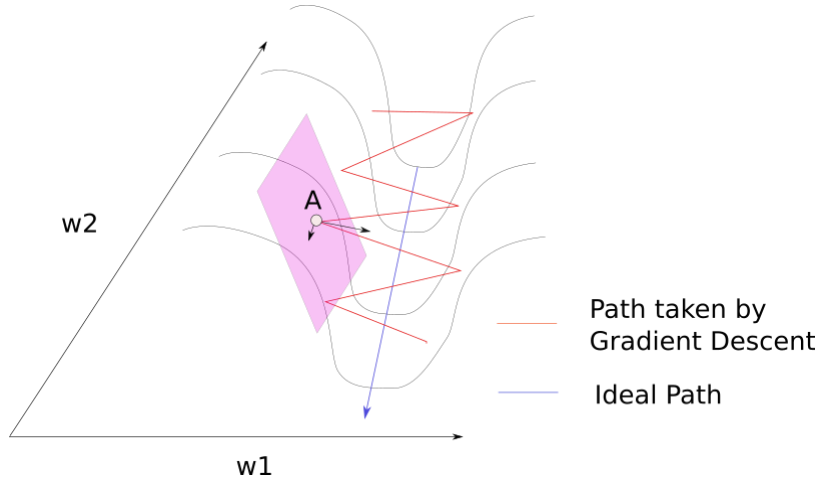
$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t) \qquad \frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \quad w_2^3 \quad f'(h_2^2) \quad a_2^1$$

- Why do we need a strategy ?

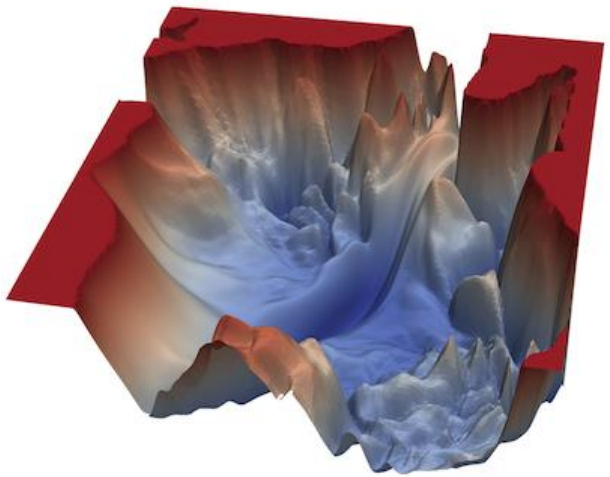
Ideal surface



Surface valley



Real surface



3) How to train neural networks ?

- The learning rate chosen in the beginning of the training might not be adequate later on !

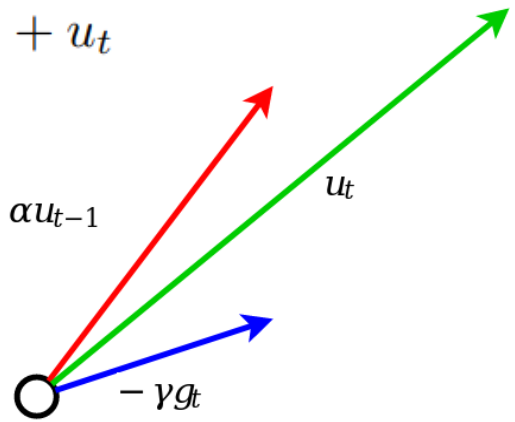
- One trick to solve this problem is called momentum

- Momentum adds inertia in the choice of the step direction
- The new variable is called the velocity u_t

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

$$u_t = \alpha\, u_{t-1} - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

$$\theta_{t+1} = \theta_t + u_t$$



- Properties :

- It can go through barrier walls
- It accelerates when the gradient does not change much
- It dampens oscillations in narrow valleys

3) How to train neural networks ?

- Particular case of momentum : Nesterov momentum

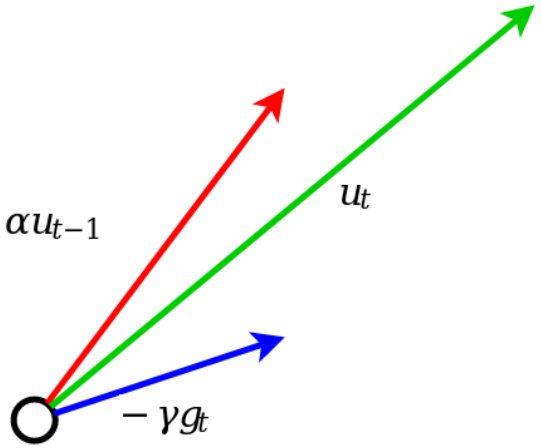
- simulate a step in the direction of the velocity, then calculate the gradient and make a correction

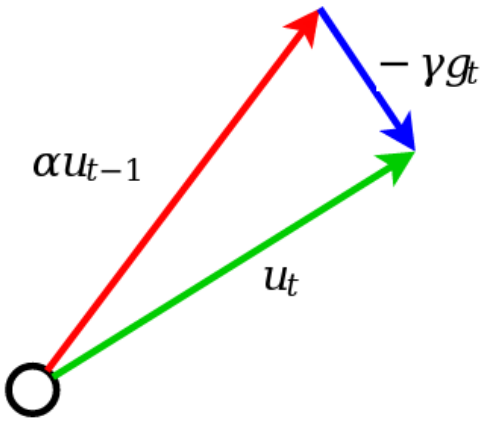
$$u_t = \alpha\, u_{t-1} - \gamma\, \nabla_\theta \mathcal{L}(\theta_t)$$

$$\theta_{t+1} = \theta_t + u_t$$

$$u_t = \alpha\, u_{t-1} - \gamma\, \nabla_\theta \mathcal{L}(\theta_t + \alpha\, u_{t-1})$$

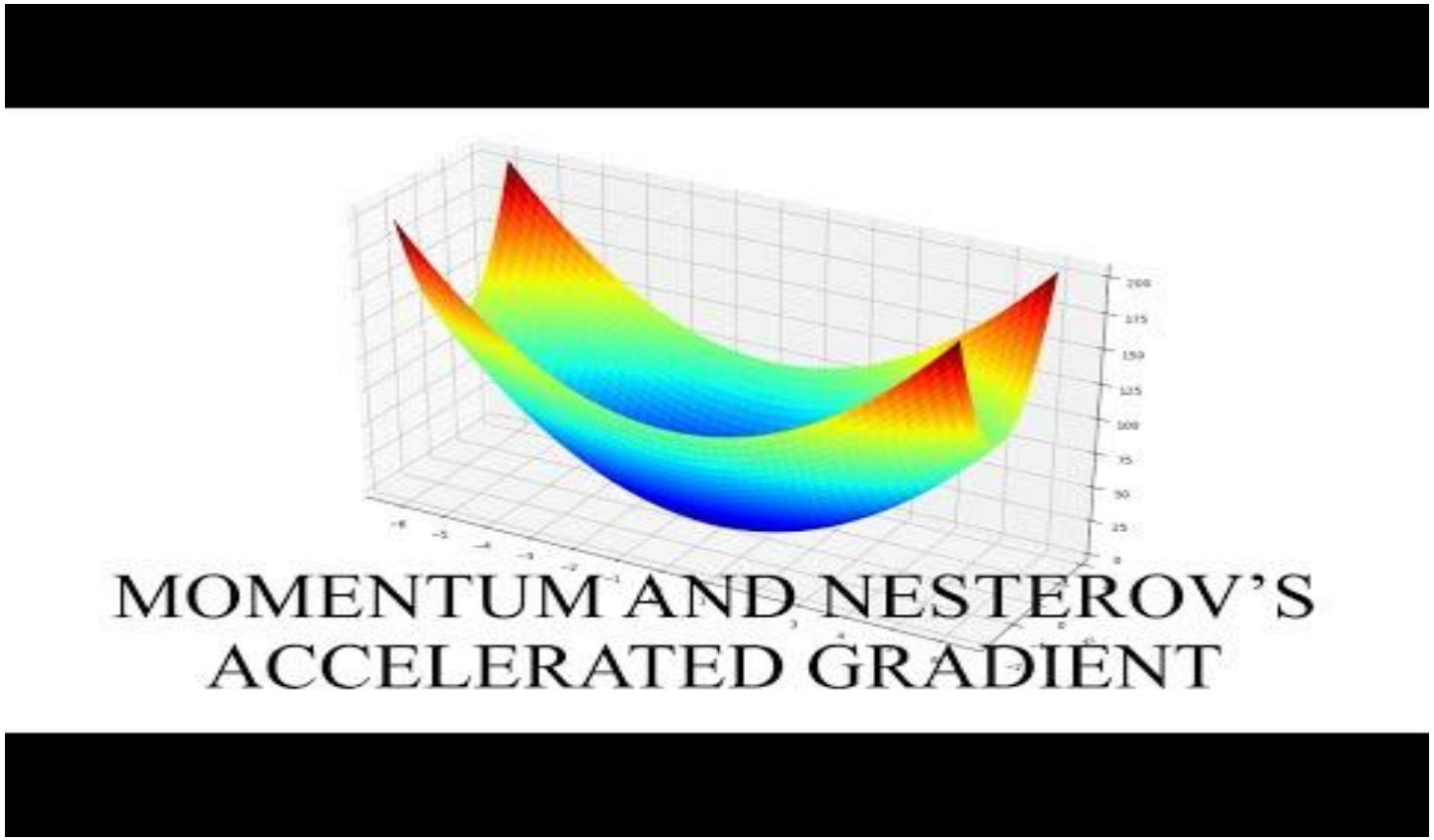
$$\theta_{t+1} = \theta_t + u_t$$





3) How to train neural networks ?

- Particular case of momentum : Nesterov momentum



3) How to train neural networks ?

- The learning rate chosen in the beginning of the training might not be adequate later on !
- Other algorithms implementing momentum-like methods :

Adam

$$
\begin{aligned}
s_t &= \rho_1 s_{t-1} + (1-\rho_1) g_t \\
\hat{s}_t &= \frac{s_t}{1-\rho_1^t} \\
r_t &= \rho_2 r_{t-1} + (1-\rho_2) g_t \odot g_t \\
\hat{r}_t &= \frac{r_t}{1-\rho_2^t} \\
\theta_{t+1} &= \theta_t - \gamma \frac{\hat{s}_t}{\delta + \sqrt{\hat{r}_t}}
\end{aligned}
$$

- works well with $\rho_1 = 0.9 \quad \rho_2 = 0.999$
- one of the default optimizers in deep learning

RMSProp

$$
\begin{aligned}
r_t &= \rho r_{t-1} + (1-\rho) g_t \odot g_t \\
\theta_{t+1} &= \theta_t - \frac{\gamma}{\delta + \sqrt{r_t}} \odot g_t .
\end{aligned}
$$

- performs better in non-convex problems
- does not grow unboundedly

3) How to train neural networks ?

- Training a NN consists in applying a strategy to update the weights of your NN

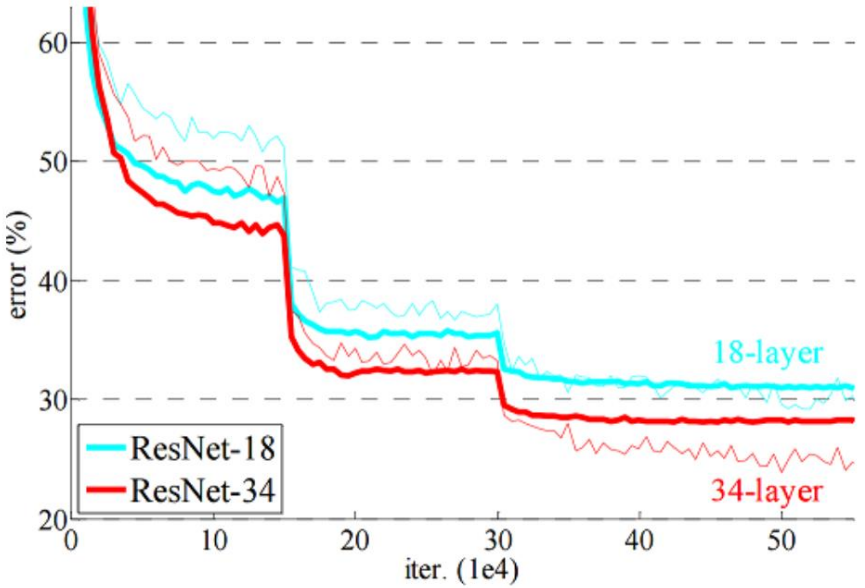
➥ this means adapting the learning rate and implementing momentum

- The algorithms that compute the gradients, implement the backpropagation, deal with the learning rate and the momentum are called optimizers

- Other methods exist to tune the learning rate,

such as the scheduling :

- consists in reducing the learning rate over time
- can be combined with Adam, RMSProp, ...



3) How to train neural networks ?

- So far, we have learned how to train a neural network

- gradient descent and backpropagation allows to update the weights
- batch size and learning rate are very important
- choosing the right activation and loss functions is critical

$$\theta_{t+1} = \theta_t - \gamma \nabla_\theta \mathcal{L}(\theta_t)$$

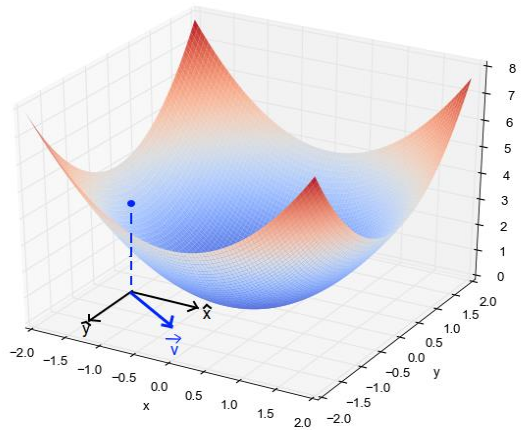
$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \quad w_2^3 \quad f'(h_2^2) \quad a_2^1$$

weight's value

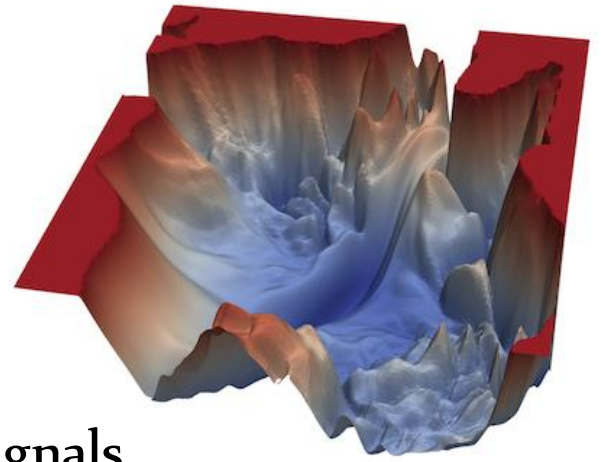
- What about the initial values of the weights ?

- we know they cannot be zero
- are there some preferred initialization schemes ?

➥ for convex problems, providing a good learning rate, convergence is achieved regardless of the initial parameter values



3) How to train neural networks ?



- In practice, most of the problems are non-convex

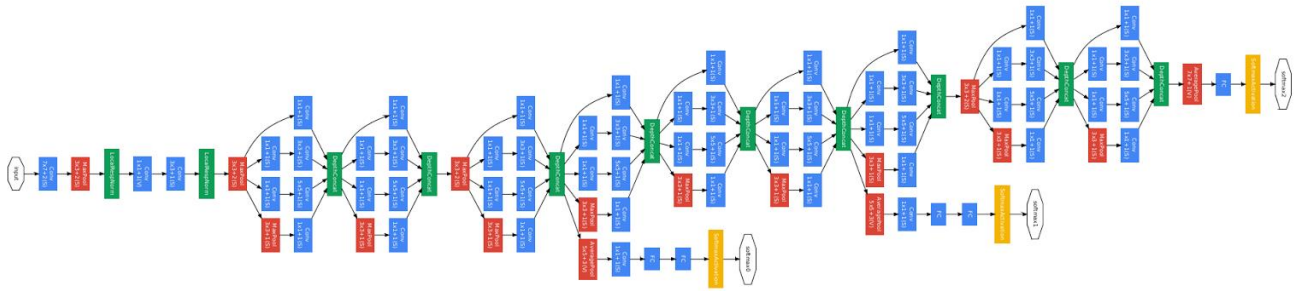
➥ initial values are important

- First strategy : you want the information to flow in your network without reducing or magnifying the amplitude of the signals

- deeper layers should receive the information

- a way of stating that consists in preserving the same variance across layers !

$$S = W^3\left(f\left(W^2\, f(W^1 x + b^1) + b^2\right)\right)$$



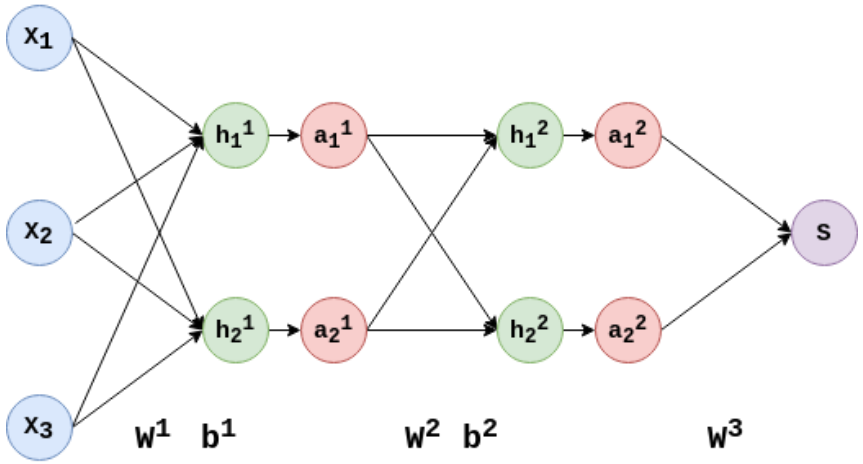
3) How to train neural networks ?

- Mathematically, this can be expressed as :

$$\mathbb{V}\left[h_i^l\right] = \mathbb{V}\left[\sum_{j=0}^{q_{l-1}-1} w_{ij}^l h_j^{l-1}\right]$$

$$= \sum_{j=0}^{q_{l-1}-1} \mathbb{V}\left[w_{ij}^l\right] \mathbb{V}\left[h_j^{l-1}\right]$$

$$\mathbb{V}\left[h^l\right] = q_{l-1}\mathbb{V}\left[w^l\right] \mathbb{V}\left[h^{l-1}\right]$$



where q_l is the width of layer l.

The only way to ensure the same variance across layers is : $\mathbb{V}\left[w^l\right] = \frac{1}{q_{l-1}} \quad \forall l$

Taken from Louppe, G.
Deep Learning

This is enforced in LeCun's uniform initialization : $w_{ij}^l \sim \mathcal{U}\left[-\sqrt{\frac{3}{q_{l-1}}}, \sqrt{\frac{3}{q_{l-1}}}\right]$

3) How to train neural networks ?

- Second strategy : you want the gradients to flow in the backward pass without vanishing

- this can be enforced by maintaining the variance of the gradient fixed across layers

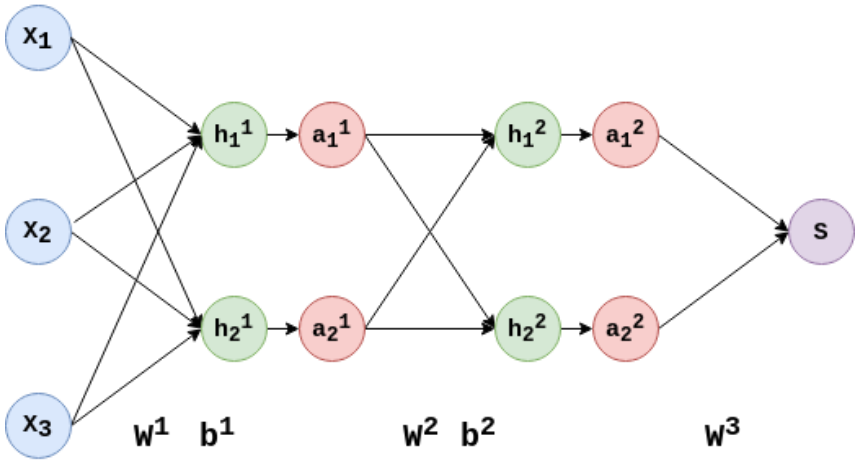
- Mathematically,

$$\mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h_i^l}\right] = \mathbb{V}\left[\sum_{j=0}^{q_{l+1}-1} \frac{\mathrm{d}\hat{y}}{\mathrm{d}h_j^{l+1}} \frac{\partial h_j^{l+1}}{\partial h_i^l}\right]$$

$$= \mathbb{V}\left[\sum_{j=0}^{q_{l+1}-1} \frac{\mathrm{d}\hat{y}}{\mathrm{d}h_j^{l+1}} w_{j,i}^{l+1}\right]$$

$$= \sum_{j=0}^{q_{l+1}-1} \mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h_j^{l+1}}\right] \mathbb{V}\left[w_{ji}^{l+1}\right]$$

$$\mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h^l}\right] = q_{l+1}\mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h^{l+1}}\right] \mathbb{V}\left[w^{l+1}\right]$$



Taken from Louppe, G.
Deep Learning

3) How to train neural networks ?

- The variance of the gradients with respect to the activations is preserved across layers if :

$$\mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h^l}\right] = q_{l+1}\mathbb{V}\left[\frac{\mathrm{d}\hat{y}}{\mathrm{d}h^{l+1}}\right]\mathbb{V}\left[w^{l+1}\right]$$

$$\rightarrow \quad \mathbb{V}\left[w^l\right] = \frac{1}{q_l} \quad \forall l$$

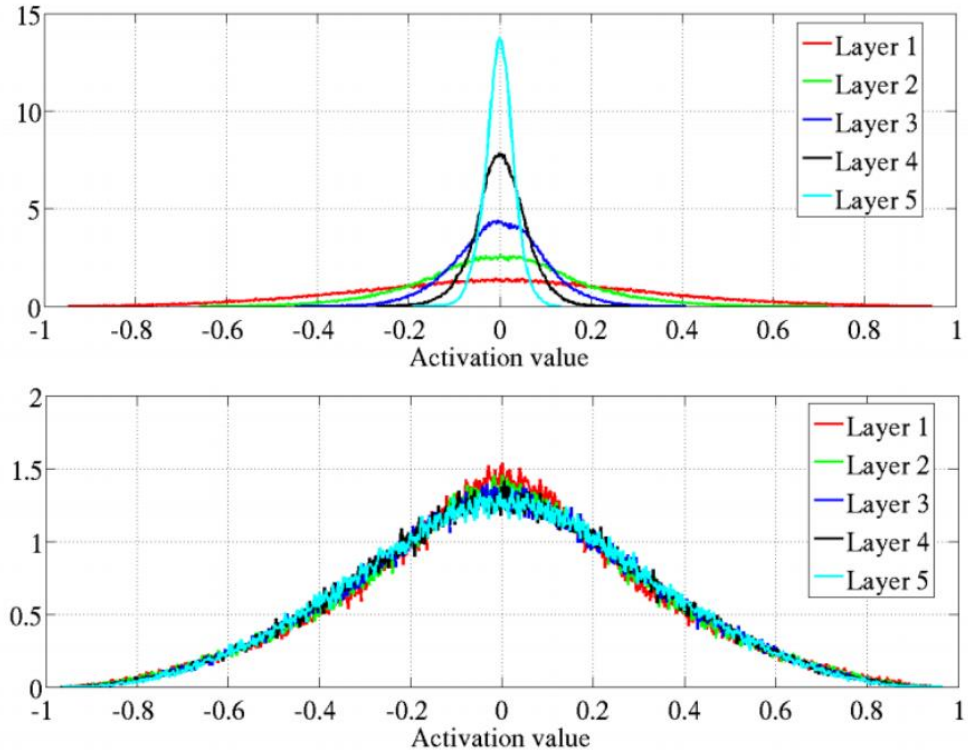
- A compromise on the conditions for the forward and backward pass is found in Xavier's initialization, which initializes the weights with a variance :

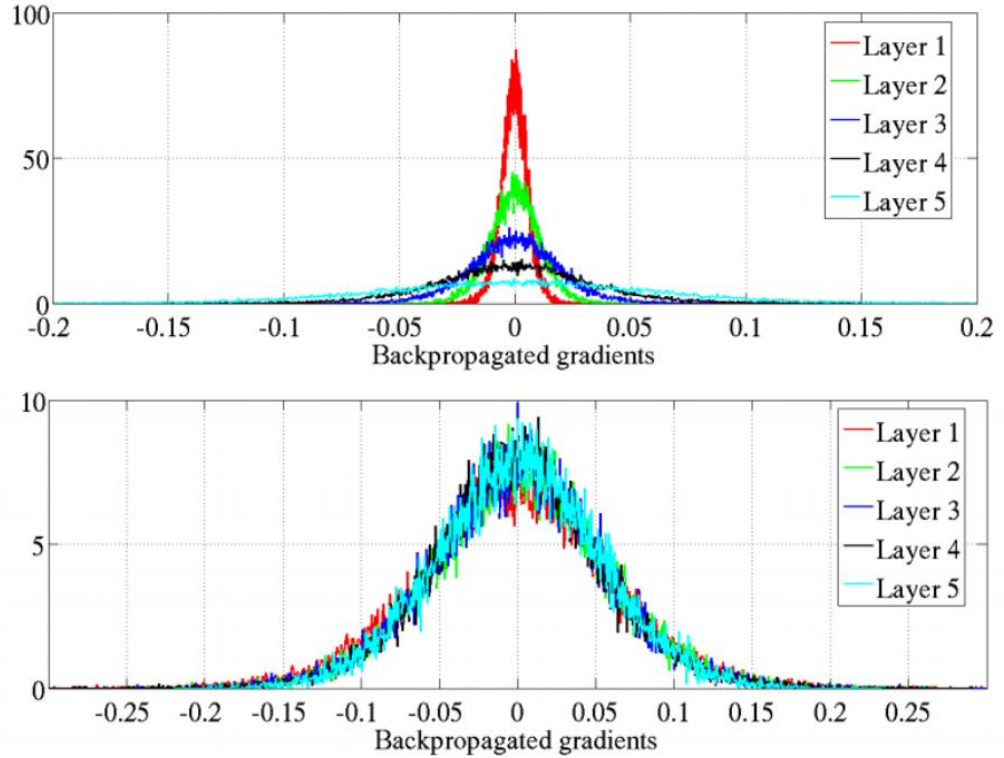
$$\mathbb{V}\left[w^l\right] = \frac{1}{\frac{q_{l-1}+q_l}{2}} = \frac{2}{q_{l-1}+q_l}$$

- The normalized Xavier's initialization : $w_{ij}^l \sim \mathcal{U}\left[-\sqrt{\frac{6}{q_{l-1}+q_l}}, \sqrt{\frac{6}{q_{l-1}+q_l}}\right]$

3) How to train neural networks ?

- How effective is the Xavier's initialization ?





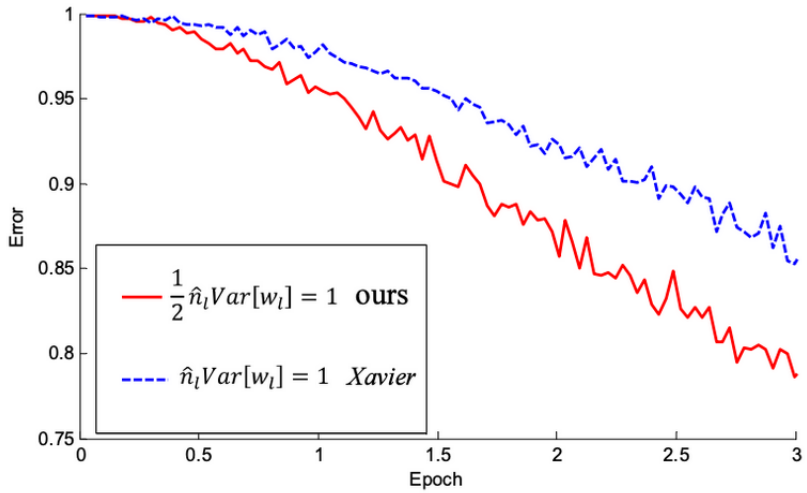
Credits: Glorot and Bengio, Understanding the difficulty of training deep feedforward neural networks, 2010

3) How to train neural networks ?

- He initialization (He et al., 2015) vs Xavier's initialization

$$\mathbb{V}\left[w^l\right] = \frac{2}{q_{l-1}} \quad \times \quad \mathbb{V}\left[w^l\right] = \frac{1}{\frac{q_{l-1}+q_l}{2}} = \frac{2}{q_{l-1}+q_l}$$

- Performs better than Xavier's initialization in some cases



3) How to train neural networks ?

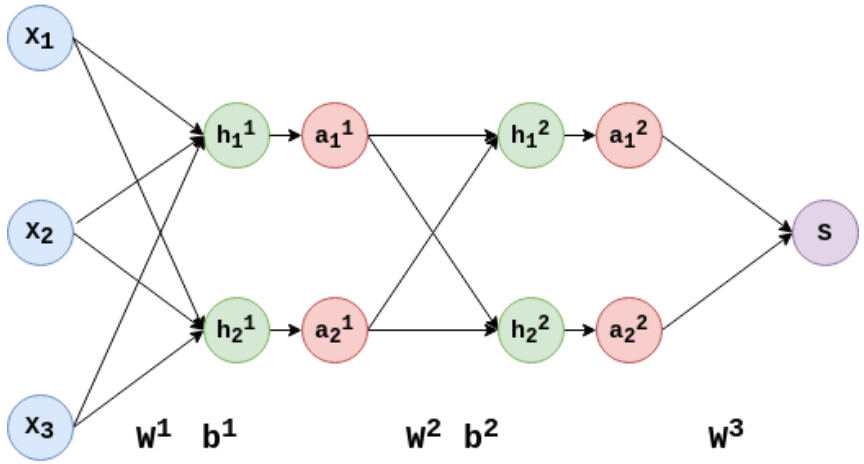
- Previous initialization strategies rely on preserving the variance across layers

- what about the first layer ?

$$\mathbb{V}\left[h_i^l\right] = \mathbb{V}\left[\sum_{j=0}^{q_{l-1}-1} w_{ij}^l h_j^{l-1}\right]$$

$$= \sum_{j=0}^{q_{l-1}-1} \mathbb{V}\left[w_{ij}^l\right] \mathbb{V}\left[h_j^{l-1}\right]$$

$$\mathbb{V}\left[h^l\right] = q_{l-1} \mathbb{V}\left[w^l\right] \mathbb{V}\left[h^{l-1}\right]$$



- For the first layer, we have assumed that the variances of the input features are the same, that is

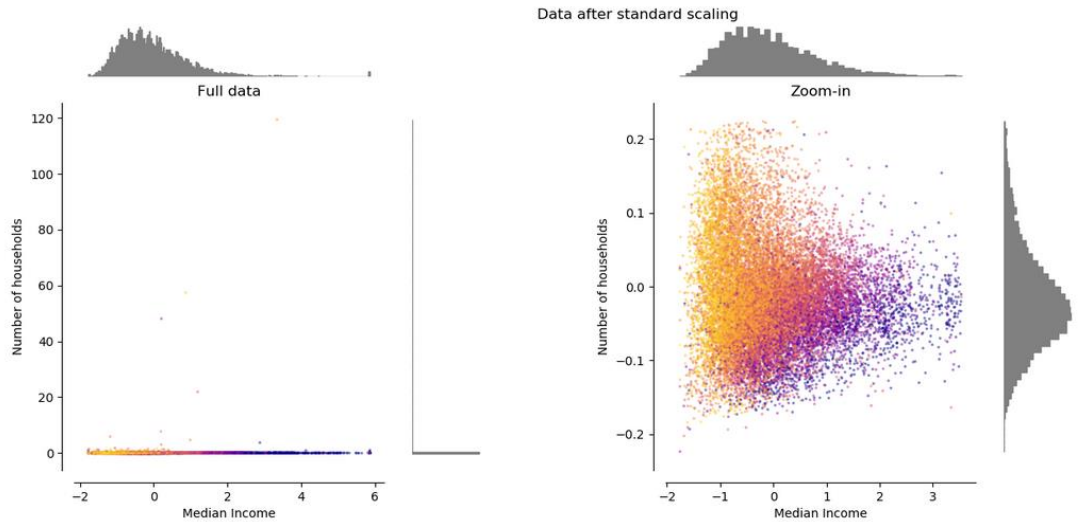
$$\mathbb{V}\left[x_i\right] = \mathbb{V}\left[x_j\right] \triangleq \mathbb{V}\left[x\right]$$

3) How to train neural networks ?

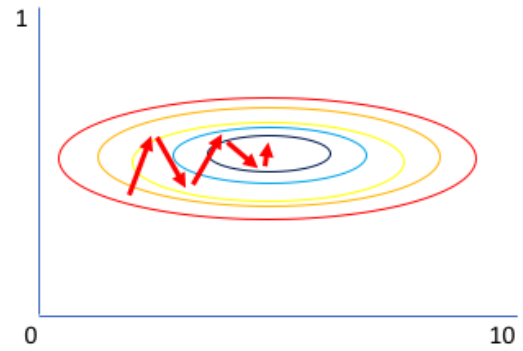
- This constraint is not satisfied in general but can be enforced by normalizing or standardizing the inputs through

$$\mathbf{x}' = (\mathbf{x} - \hat{\mu}) \odot \frac{1}{\hat{\sigma}}$$

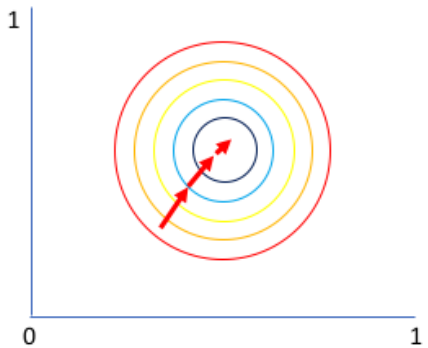
with $\hat{\mu} = \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{d}} \mathbf{x}$ $\hat{\sigma}^2 = \frac{1}{N} \sum_{\mathbf{x} \in \mathbf{d}} (\mathbf{x} - \hat{\mu})^2$



Before normalization



After normalization

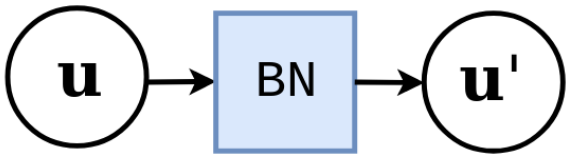


3) How to train neural networks ?

- Another method to normalize your data is called batch normalization

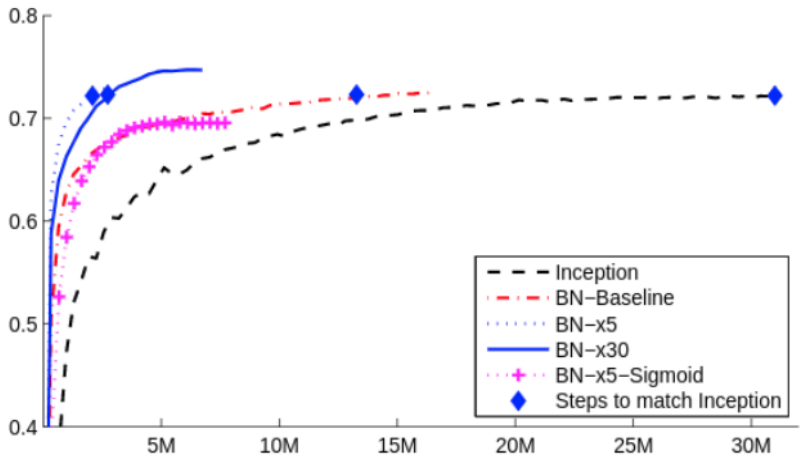
- let us consider a mini-batch of data, where B is the number of data

- a mean and a variance are estimated across the batch



$$\hat{\mu}_{\text{batch}} = \frac{1}{B}\sum_{b=1}^{B} \mathbf{u}_b \qquad \hat{\sigma}^2_{\text{batch}} = \frac{1}{B}\sum_{b=1}^{B} (\mathbf{u}_b - \hat{\mu}_{\text{batch}})^2$$

- These estimates will be used to normalize the output : $\mathbf{u}'_b = \gamma \odot (\mathbf{u}_b - \hat{\mu}_{\text{batch}}) \odot \frac{1}{\hat{\sigma}_{\text{batch}} + \epsilon} + \beta$

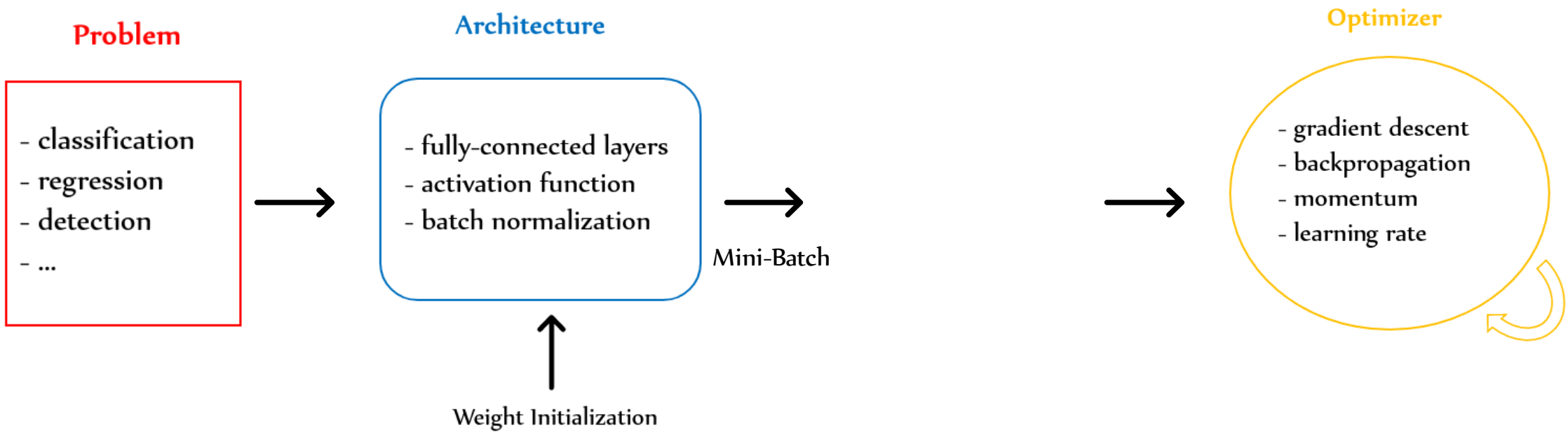


Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Credits: Ioffe and Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.

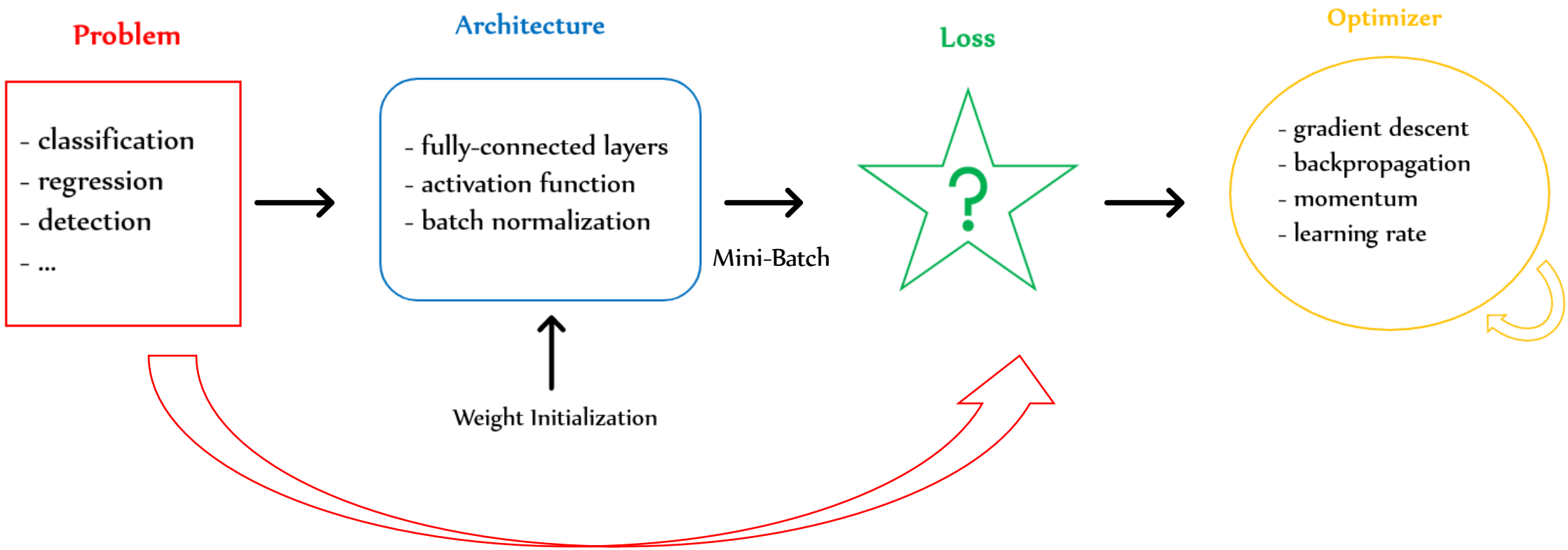
3) How to train neural networks ?

- Where are we so far ?



3) How to train neural networks ?

- Where are we so far ?



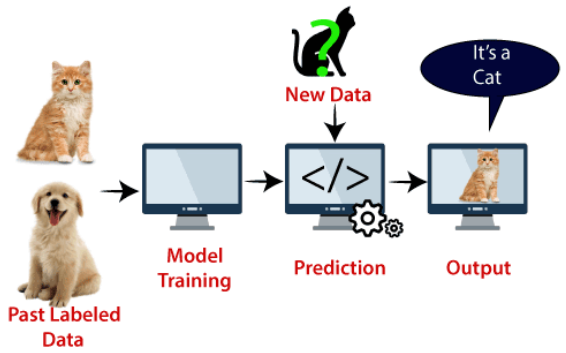
4) How to choose the loss function ?

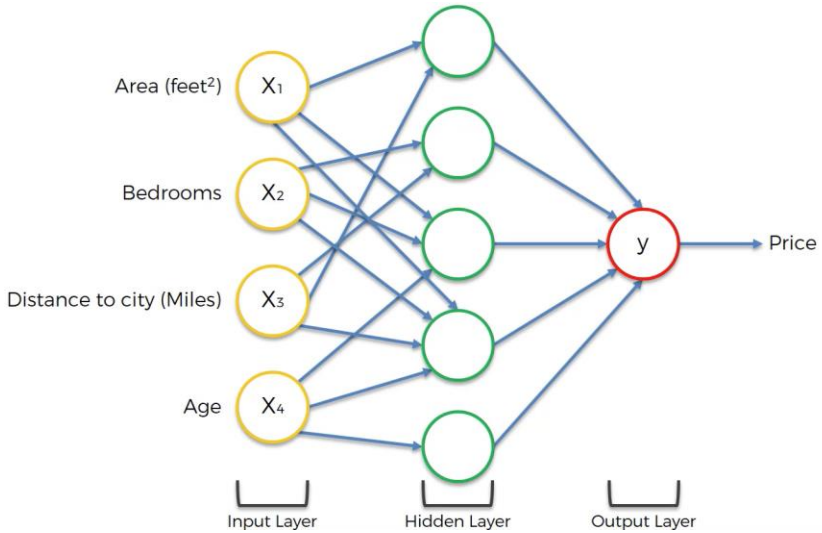
- The choice of the loss function is critical

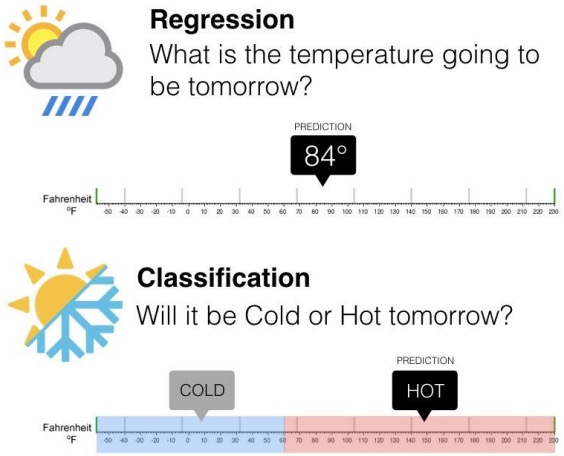
- the best architecture + the best optimizer + the wrong loss = FAIL

- a simple architecture + a random optimizer + the good loss = SUCCESS

- Most of the problems can be translated into regression OR classification







4) How to choose the loss function ?

- Regression : involves predicting a value that is continuous in nature (temperature, price, ..)

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{i,true} - y_{i,predicted}\right)^2$$

Advantages :
- no local minima
- penalizes large errors

Drawbacks :
- outliers are not handled properly

Mean Absolute Error (L1 loss)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_{i,true} - y_{i,predicted}\right|$$

Advantages :
- outliers are handled better than MSE

Drawbacks :
- computationally expensive
- there may be local minima

Mean Squared Logarithmic Error

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}\left(log(y_{i,true}) - log(y_{i,predicted})\right)^2$$

Advantages :
- relax the penalties on huge errors

Drawbacks :
- more expensive than MSE
- penalizes more underestimates than overestimates

Huber loss

$$L_\delta = \begin{cases} \frac{1}{2}\left(y_{true} - y_{predicted}\right)^2, & \text{if} |y_{true} - y_{predicted}| \leq \delta \\ \delta\left|y_{true} - y_{predicted}\right| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases}$$

Advantages :
- outliers handled properly
- no local minima

Drawbacks :
- parameter to tune
- complex

4) How to choose the loss function ?

- Classification : involves predicting a discrete class output

Binary Cross Entropy Loss

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\left(y_{i,true}\,log(y_{i,predicted}) + (1-y_{i,true})\,log(1-y_{i,predicted})\right)$$

Notes :

- requires the use of Sigmoid function as the output of your NN

BCE with Logits Loss

$$BCEL = -\frac{1}{n}\sum_{i=1}^{n}\left(y_{i,true}\,log(\sigma(y_{i,predicted})) + (1-y_{i,true})\,log(1-\sigma(y_{i,predicted}))\right)$$

Notes :

- Sigmoid included in the loss (more numerically stable than BCE)

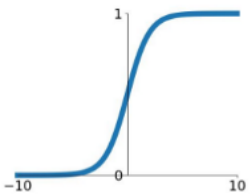
Cross Entropy Loss

$$CE = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C}\left(y_{ij,true}\,log(y_{ij,predicted})\right)$$

Notes :

- requires one-hot encoding of labels
- requires the use of Softmax function as the output of your NN

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



Pet	Cat	Dog	Turtle	Fish
Cat	1	0	0	0
Dog	0	1	0	0
Turtle	0	0	1	0
Fish	0	0	0	1
Cat	1	0	0	0

$$\text{Softmax } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

4) How to choose the loss function ?

- Training a NN is long, complex and sometimes confusing

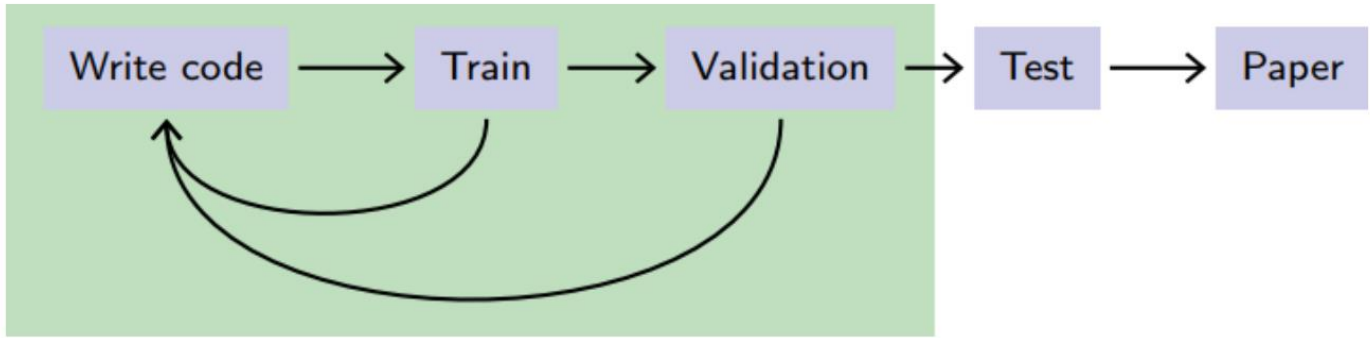
- You don't want to use brute force and redo the training 20 times

- Need visualization tools meanwhile the NN is being trained !

➥ Plot your loss curves !

- Once your model is trained, check the loss curves before testing your network !!!

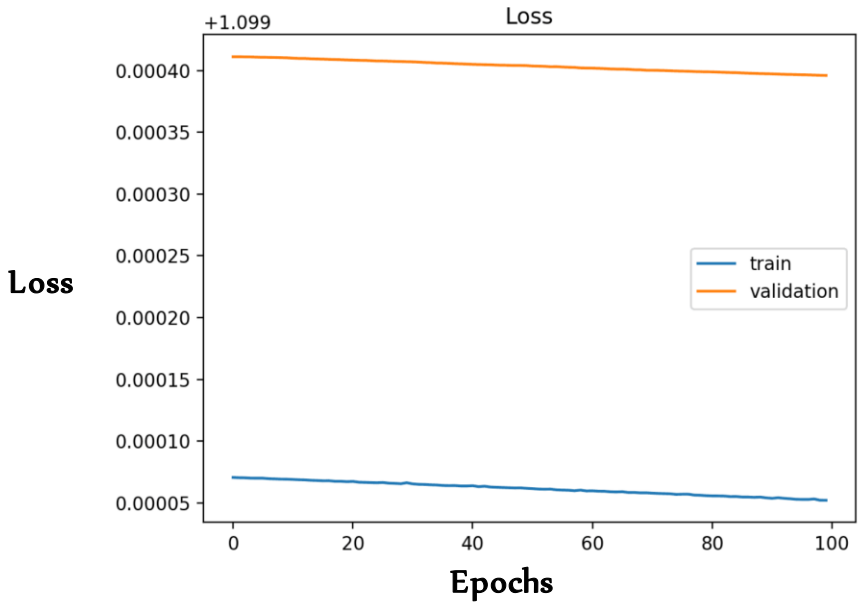
- Workflow :



4) How to choose the loss function ?

- First problem : underfitting/overfitting (Jason Brownlee, 2019)

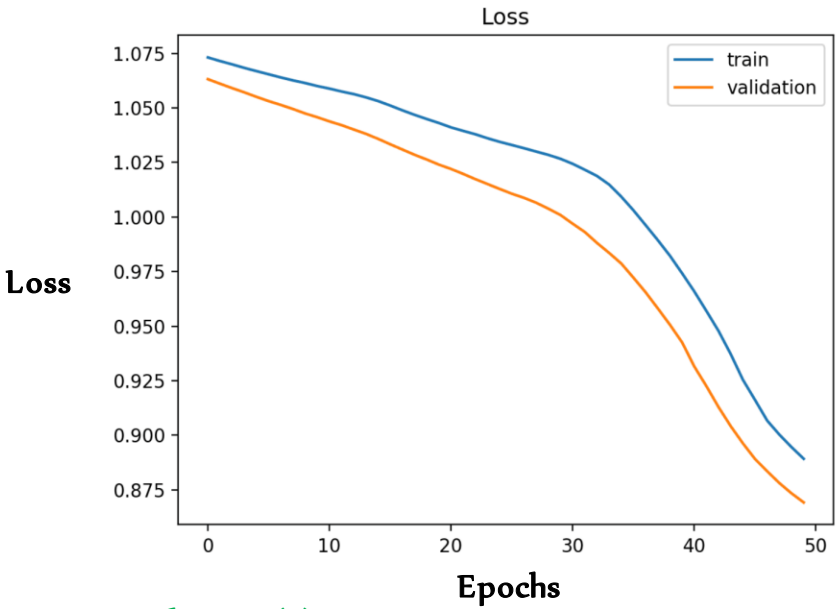
Problem : flat curve or noisy behavior of with high values



Solution(s) :

- add more data
- increase model capacity (more layers, etc..)

Problem : loss curves can go lower but halted prematurely



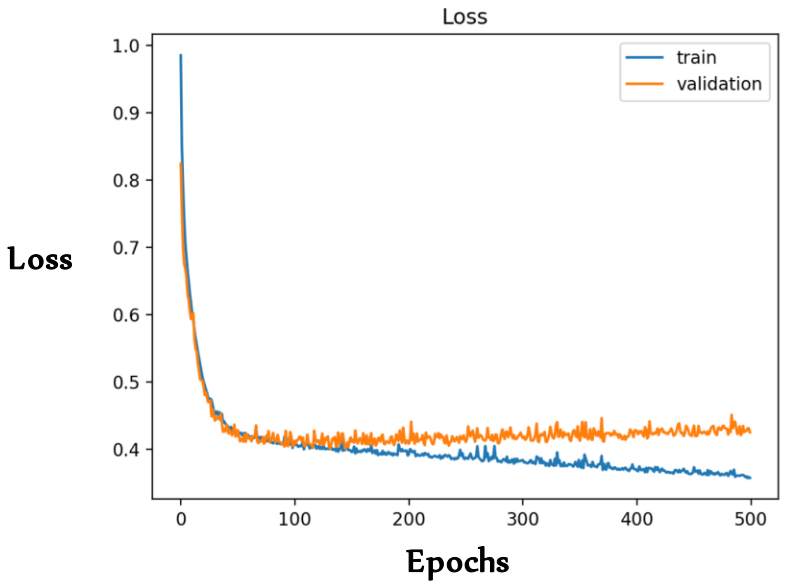
Solution(s) :

- add more training steps
- increase the learning rate if the training is too long

4) How to choose the loss function ?

- First problem : underfitting/overfitting (Jason Brownlee, 2019)

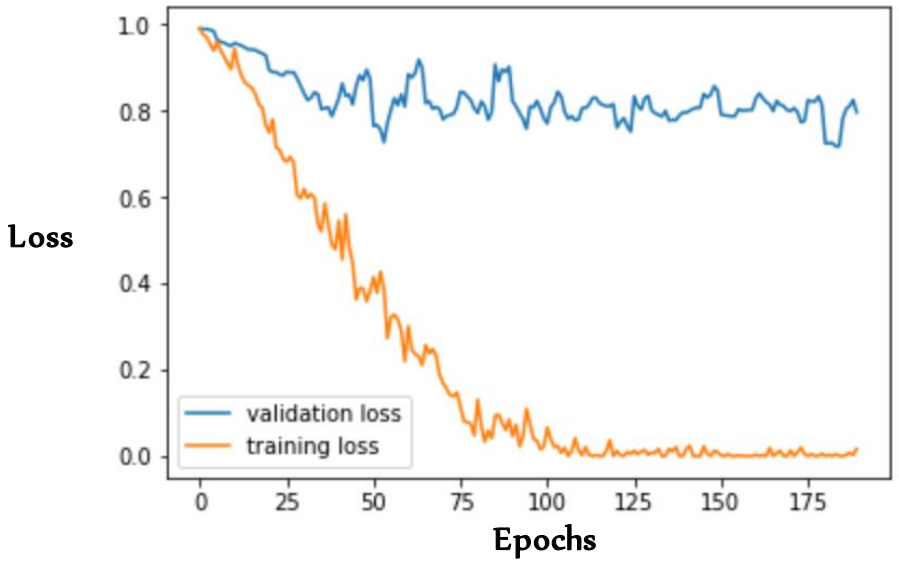
Problem : validation curve increases while training curve decreases



Possible solution(s) :

- reduce model capacity (less layers, etc..)
- reduce the learning rate if NN learns too fast

Problem : validation curve levels off while training curve decreases



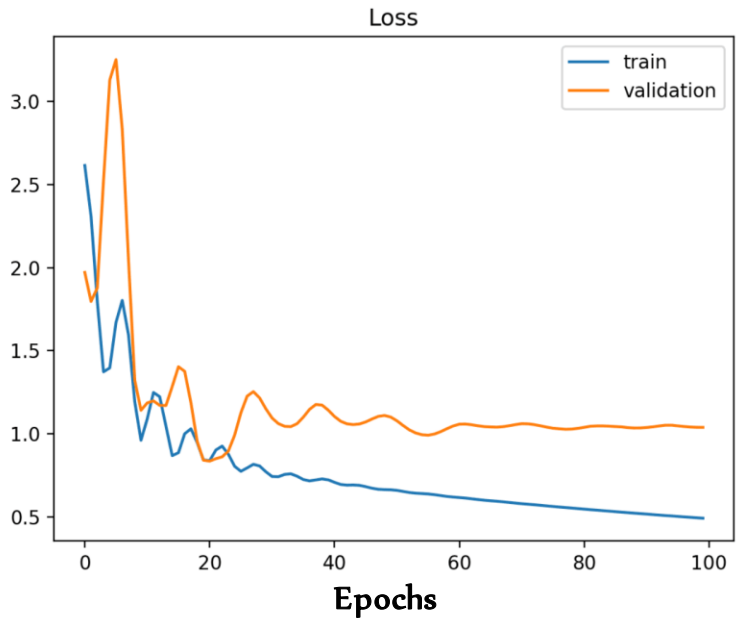
Possible solution(s) :

- collect more data (in case of memorization)
- change NN architecture

4) How to choose the loss function ?

- Second problem : unrepresentative training/validation dataset (Jason Brownlee, 2019)

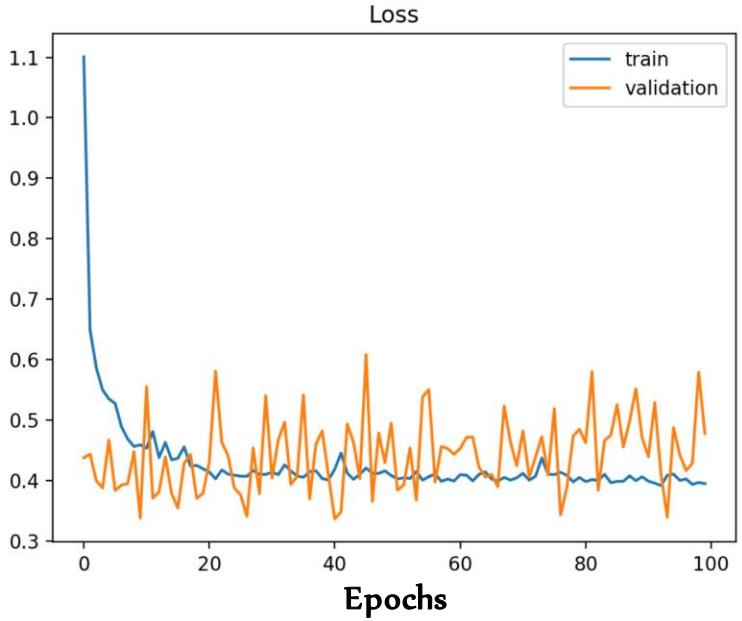
Problem : both losses show improvement but a large gap remains



Solution(s) :

- add more samples in the training set

Problem : training loss smoothly decreases while the validation loss oscillates "randomly"



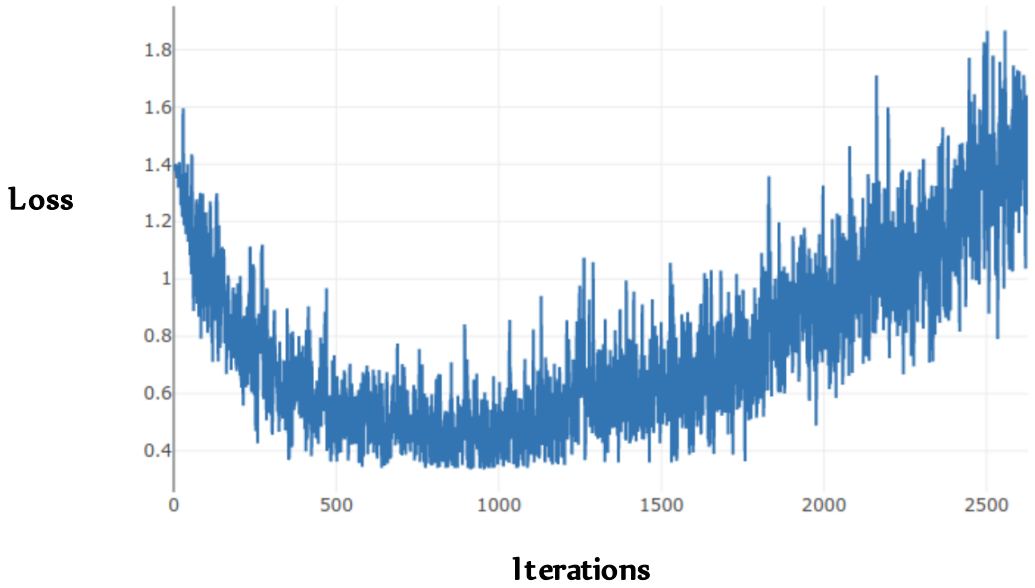
Solution(s) :

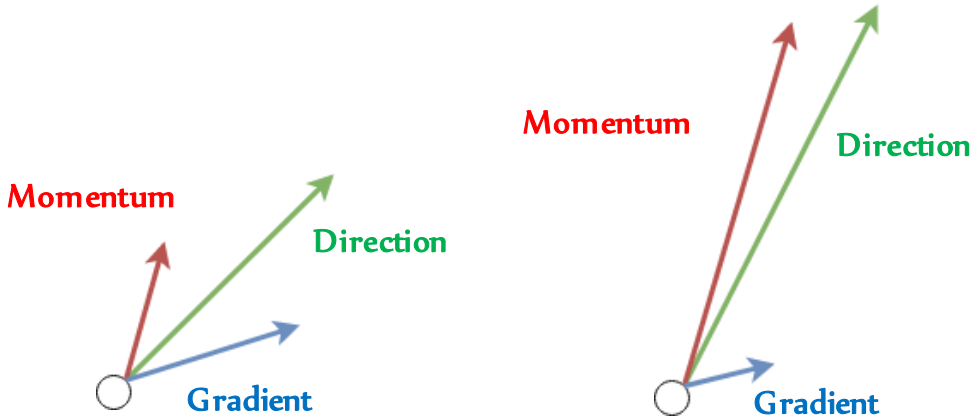
- add more samples in the validation set

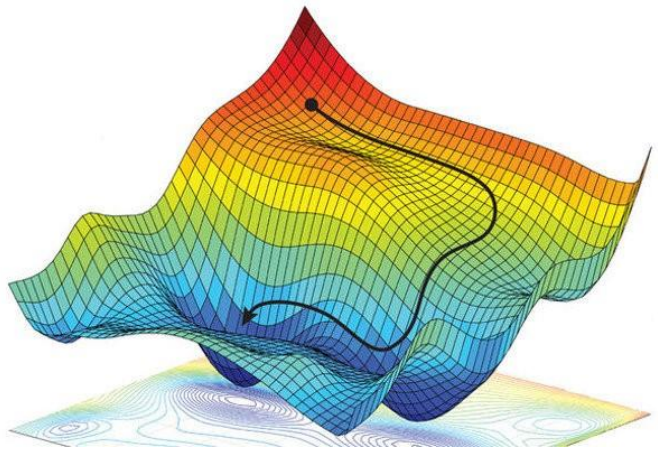
4) How to choose the loss function ?

- Third problem : too large momentum

Problem : Training loss is increasing after some iterations!







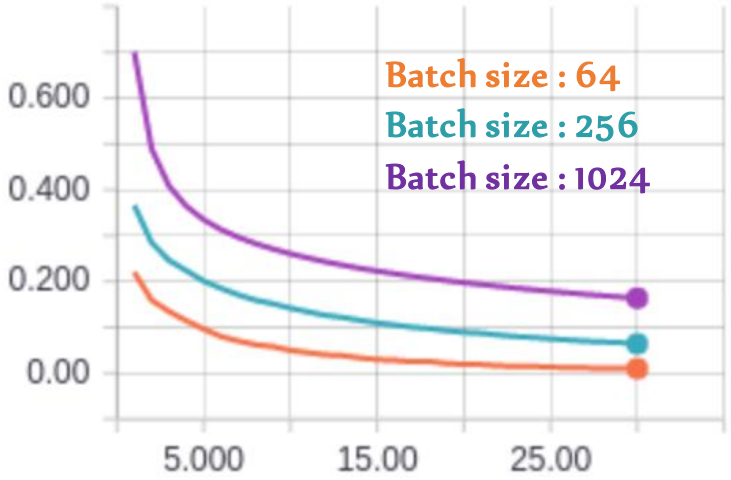
Possible solution(s) :

- use scheduling to reduce the learning rate or change the optimizer
- use another loss function (if possible)

4) How to choose the loss function ?

- Comment : the (mini-)batch size is closely related to the learning rate

Identical learning rate, different batch sizes

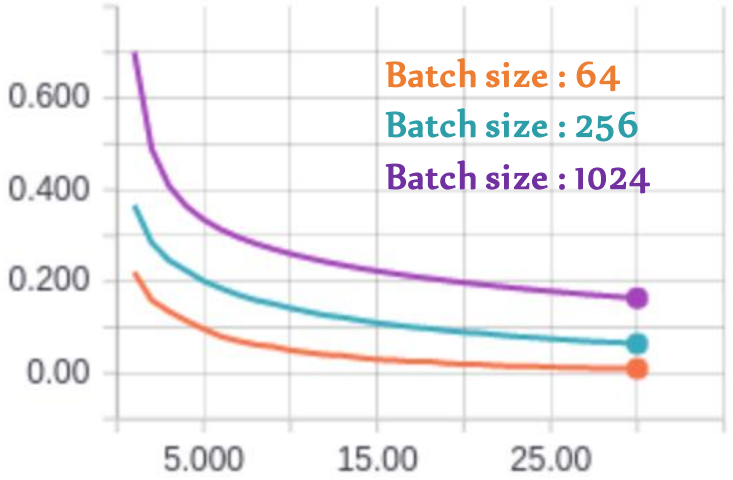


Naïve conclusion : increasing the batch size leads to worst results

4) How to choose the loss function ?

- Comment : the (mini-)batch size is closely related to the learning rate

Identical learning rate, different batch sizes



Naïve conclusion : increasing the batch size leads to worst results

Different batch sizes AND learning rates

Correct conclusion : batch size and learning rate have to be adapted together !

4) How to choose the loss function ?

- Most of the problems can be solved by simply looking at the loss curves !

- the examples shown should serve as a general rule

- depending on your architecture, lots of other things can go wrong

- You will not get it immediately

- understanding loss curves take time

- ... and failures !

- From "Amateurs" to "Advanced" :

- interpreting loss curves + adapting your NN training

- be comfortable with other architectures

- The deep learning jargon...

GradCAM

Batch size

Overfitting >< underfitting

Convolution layers

Pooling

Vanishing gradients

Loss function

Activation function

Transfer learning

RNN

Learning rate

Normalization

Optimizer

Transposed convolution

Backpropagation

Gradient descent

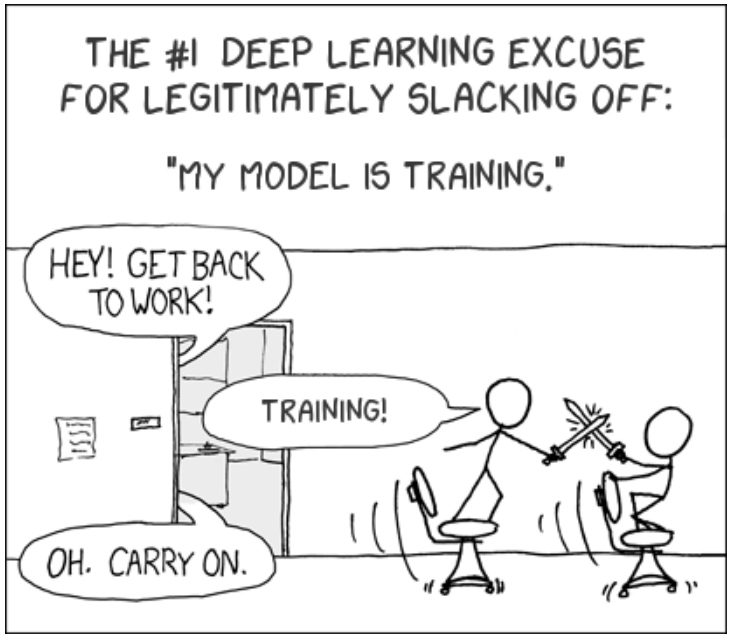
Fully-connected

ReLU

YOLO

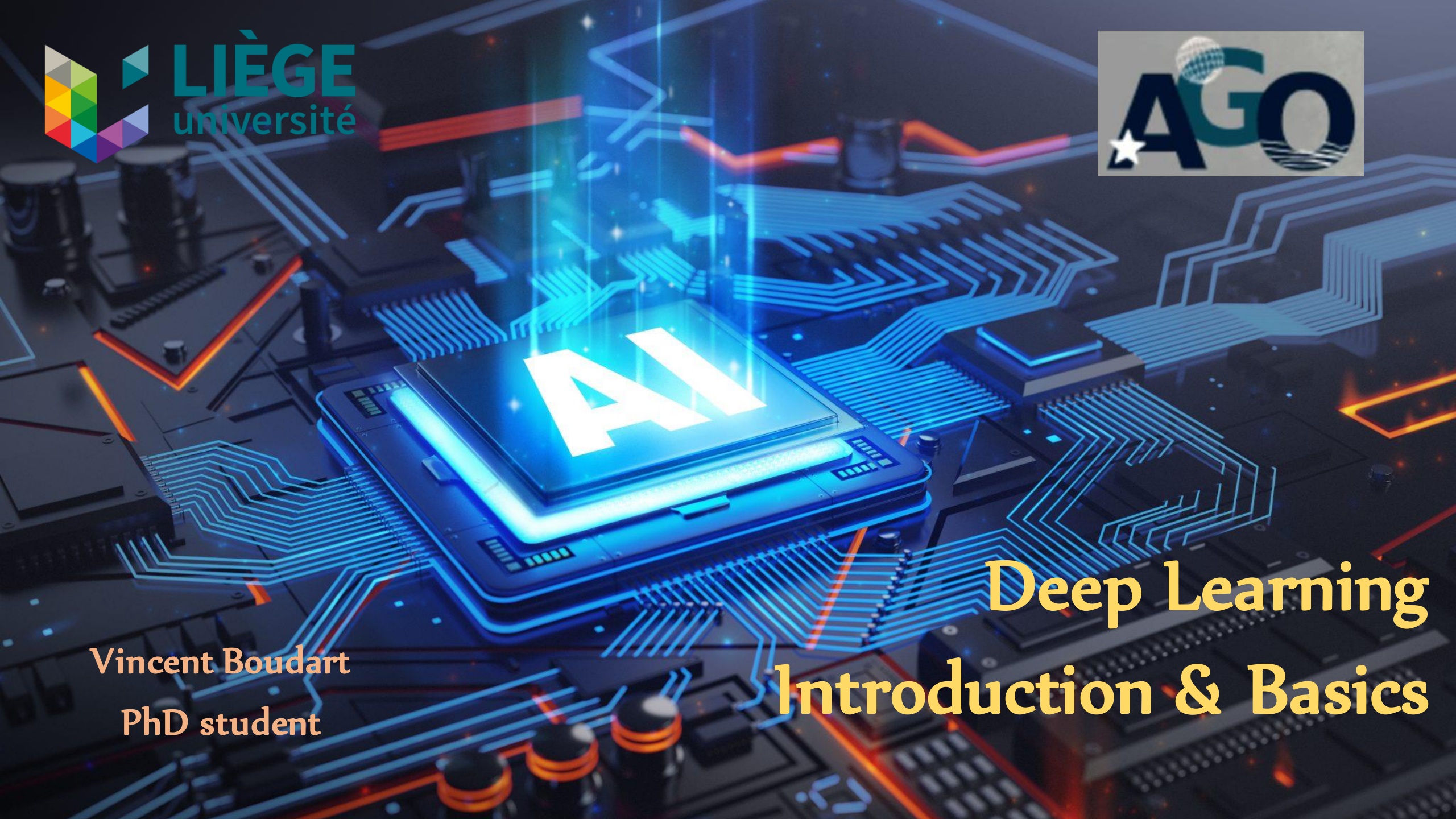
Initialization

Course 2 : The end



Vincent Boudart, PhD student

vboudart@uliege.be



LIÈGE université

AGO

Deep Learning
Introduction & Basics

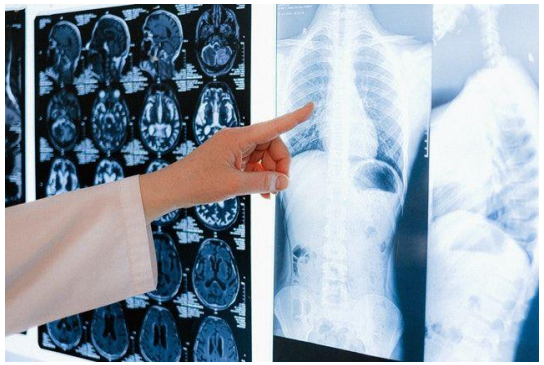
Vincent Boudart
PhD student

Course 3 : Table of contents

1) Why convolutional neural networks ?

- For the moment, we know how to process tabular data with the multi-layer perceptron
- In lots of domains, data are much different than values in a table

Telescopes



Medical diagnosis

Security Camera

1) Why convolutional neural networks ?

- Can you process images with the multi-layer perceptron ?

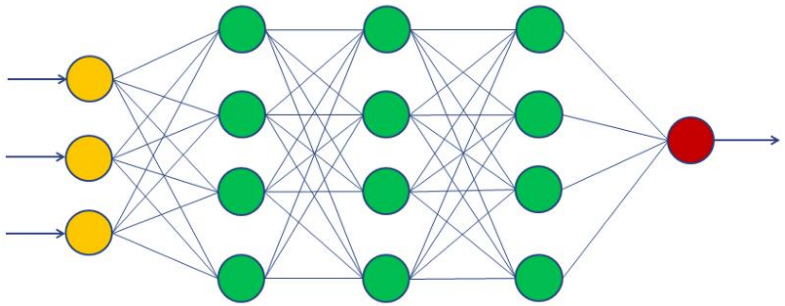
- Let us consider Hubble images captured via the Advanced Camera for Surveys (ACS)

- High Resolution Channel : 1000 pixels square images

- Wide Field Channel : 4000 pixels square image

- You can flatten the images to get 1 dimensional data

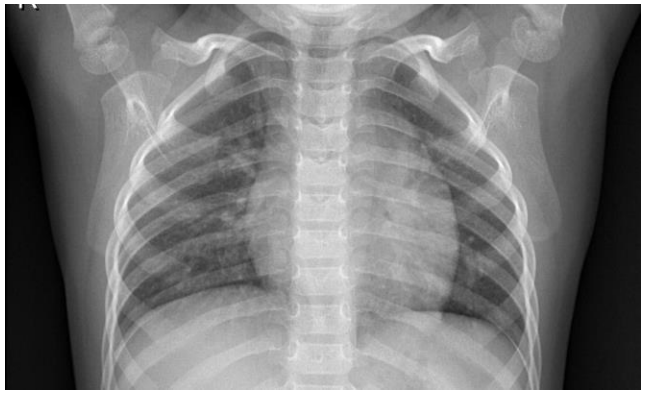
➡ millions of pixels leads to billions of parameters to train



- (Might be ok for small images)

1) Why convolutional neural networks ?

- Why MLP is not a good idea to process images ?

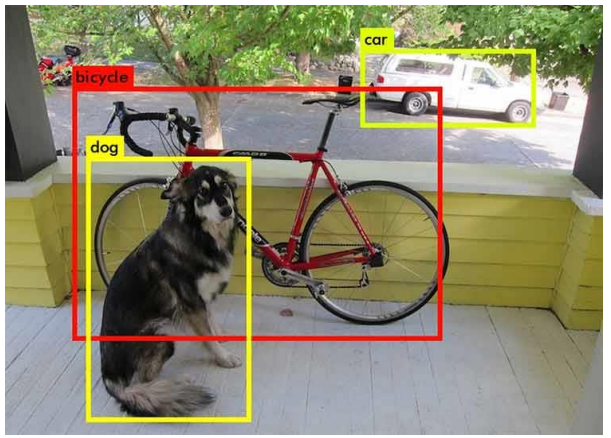


- Do we really need to connect all the pixels together ?

- No, resource consuming

- How can you be sure your MLP detects what you want ?

- No visual information, even in the intermediate layers



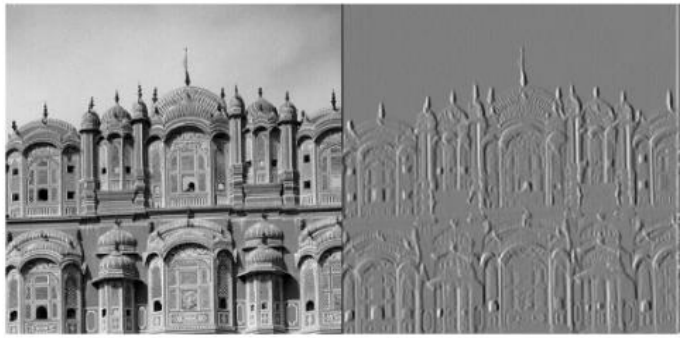
1) Why convolutional neural networks ?

- What conditions should we fulfill ?

- **locality** : only look in a small region around the pixel of interest
- **invariance to translation** : should recognize objects everywhere in the image
- **feature hierarchy** : should be composed of layers learning features at different scales

- What can we use to satisfy these conditions ?

- In computer vision, lots of tools make use of **kernels**
- edge detection, blurring and sharpening operators, template matching, ...



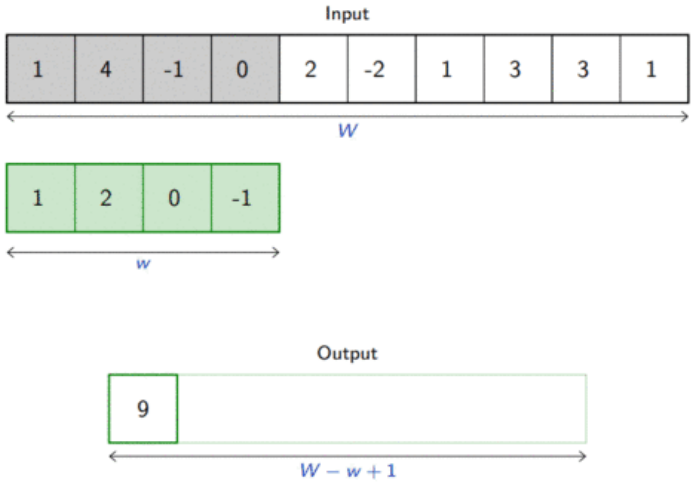
- Kernels are linear operators convolved with the input image

➡ Convolution

$$[h_x] = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

1) Why convolutional neural networks ?

- A convolution layer applies the same linear transformation locally everywhere



Taken from : Francois Fleuret, EE559 Deep Learning, EPFL

- Convolutions generalize easily to N dimensions through multi-dimensions tensors

- for a 3D kernel ***u***, it slides across the input image along its height ***h*** and width ***w***

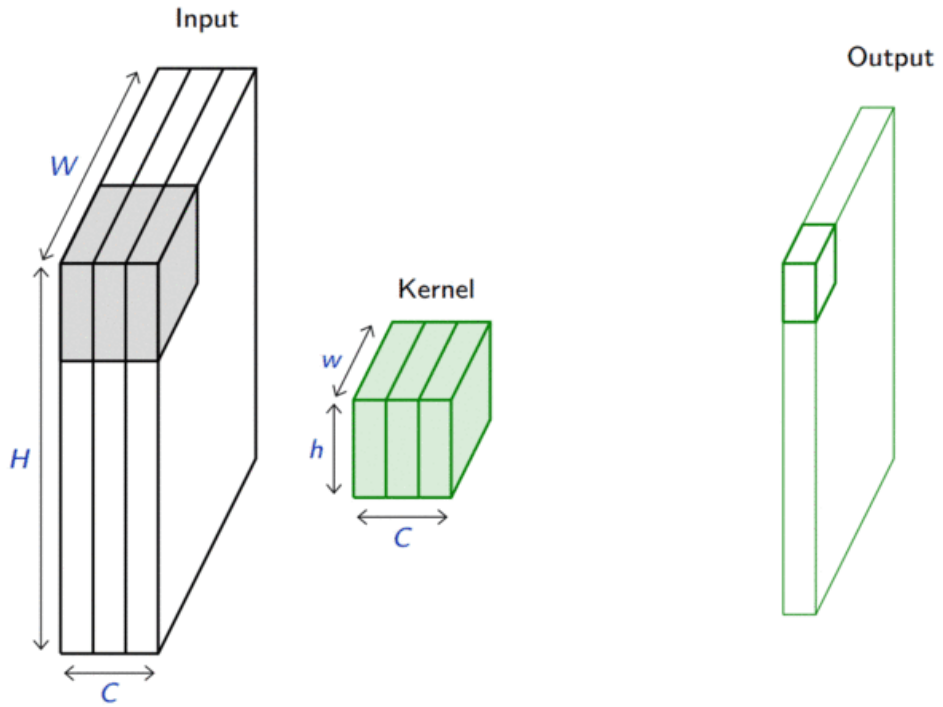
- the size ***h*** x ***w*** is the size of the receptive field

1) Why convolutional neural networks ?

- Convolutions generalize easily to N dimensions through multi-dimensions tensors

- for a 3D kernel ***u***, it slides across the input image along its height ***h*** and width ***w***

- the size ***h*** x ***w*** is the size of the receptive field



Taken from : Francois Fleuret, EE559 Deep Learning, EPFL

1) Why convolutional neural networks ?

- Convolutions are great BUT do gradient descent and backpropagation still work ?

- Example of a very simple convolution : $x \circledast u = \begin{pmatrix} 4 & 6 & 3 \\ 8 & 5 & 1 \\ 2 & 4 & 6 \end{pmatrix} \circledast \begin{pmatrix} 3 & 2 \\ 5 & 1 \end{pmatrix} = \begin{pmatrix} 69 & 50 \\ 48 & 43 \end{pmatrix} = h$

- Let us write the convolution operation as a single matrix multiplication :

- For this, we can flatten the input : $v(x) = \begin{pmatrix} 4 & 6 & 3 & 8 & 5 & 1 & 2 & 4 & 6 \end{pmatrix}^T$

- The kernel ***u*** can be expressed as : $U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$

1) Why convolutional neural networks ?

- Convolutions are great BUT do gradient descent and backpropagation still work ?

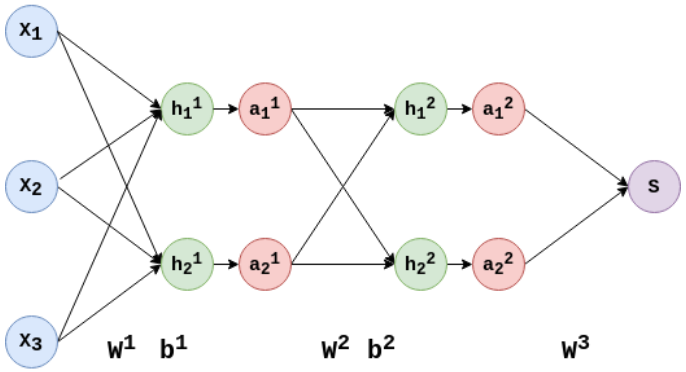
$$v(x) = \begin{pmatrix} 4 & 6 & 3 & 8 & 5 & 1 & 2 & 4 & 6 \end{pmatrix}^T \qquad U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$$

- The matrix multiplication gives :

$$U\,v(x) = \begin{pmatrix} 69 & 50 & 48 & 43 \end{pmatrix}^T = v(h)$$

- A convolution layer is just a special case of a fully connected layer :

$$v(h) = U\,v(x) \quad \Rightarrow \quad h^1 = W^1\,x + b^1$$



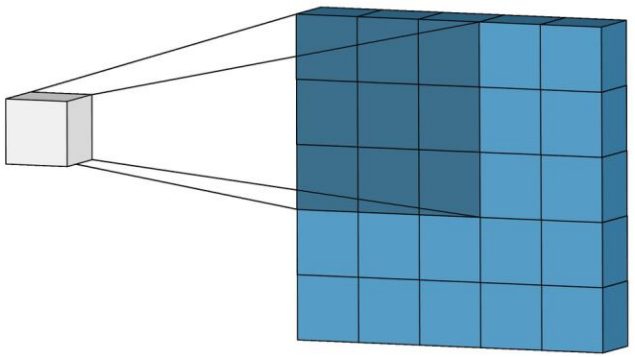
2) How to build a CNN ?

- Convolutions allow to process images with less parameters than fully-connected layers
- Gradient descent and backpropagation still work

➥ We can use convolutions in a neural network !

$$U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$$

- Can we adjust the convolution operation ?
- What are the ingredients needed to build a CNN ?
- How do the layers interact with each other ?



2) How to build a CNN ?

- Can we adjust the convolution operation ?

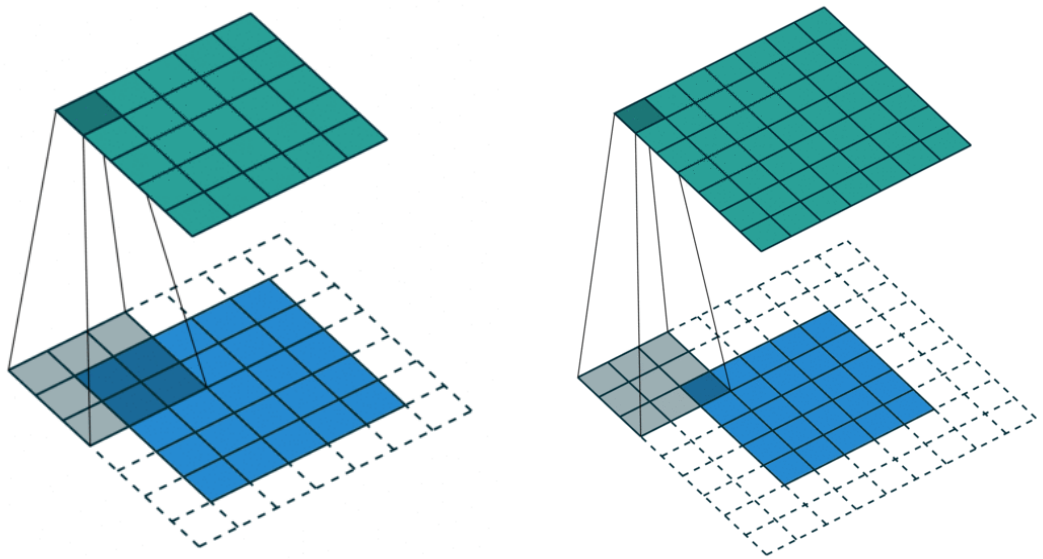
- convolution is a linear combination of values from the input

- the kernel size is not the only parameter we can tune

$$U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$$

- The padding specifies the size of a zeroed frame added around the input

- Padding is useful to control the spatial dimension of the output map, for example to keep it constant across layers



Taken from : Dumoulin and Visin, 2016.

2) How to build a CNN ?

- Can we adjust the convolution operation ?

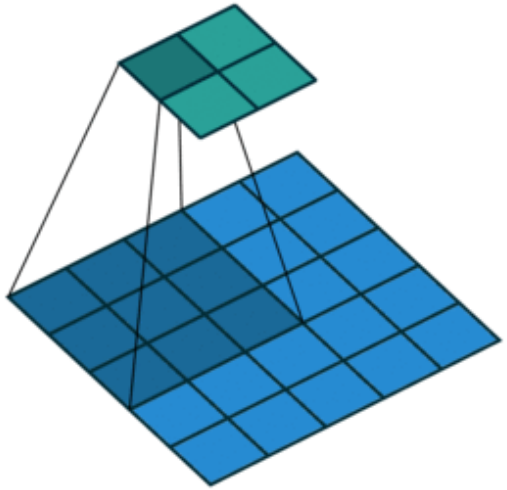
- convolution is a linear combination of values from the input

- the kernel size is not the only parameter we can tune

$$U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$$

- The stride specifies a step size when moving the kernel across the signal

- Stride is useful to reduce the spatial dimension of the feature map by a constant factor



Taken from : Dumoulin and Visin, 2016.

2) How to build a CNN ?

- Can we adjust the convolution operation ?

- convolution is a linear combination of values from the input

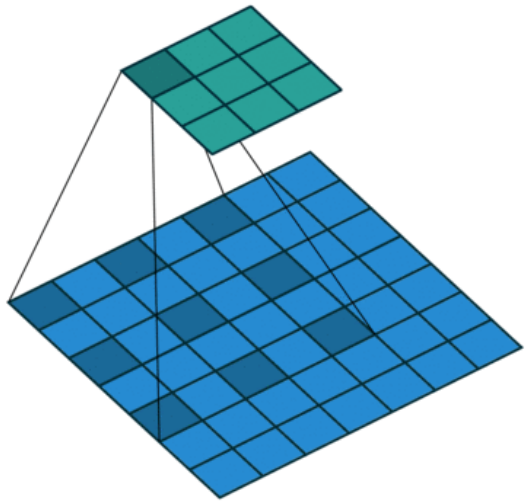
- the kernel size is not the only parameter we can tune

$$U = \begin{pmatrix} 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 2 & 0 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 0 & 5 & 1 \end{pmatrix}$$

- The dilation modulates the expansion of the filter without adding weights

- The dilation modulates the expansion of the kernel support by adding rows and columns of zeros between coefficients

- Having a dilation coefficient greater than one increases the units receptive field size without increasing the number of parameters



Taken from : Dumoulin and Visin, 2016.

2) How to build a CNN ?

- Convolution are defined through 4 parameters : kernel size, padding, stride and dilation

- The output of a convolution layer can be computed via :

$$W_{out} = \left\lfloor \frac{W_{in} - 1 + 2\,padding - dilation\,(kernel - 1)}{stride} + 1 \right\rfloor$$

- example : input image of size 200x200, kernel 3x3, stride 2x2, dilation 1x1, padding 0x0

- output : image of size 99x99

- The stride is the main parameter to reduce the size of the data !

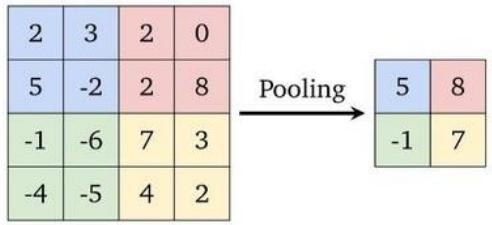
➔ Can we reduce the size of the image in another way ?

2) How to build a CNN ?

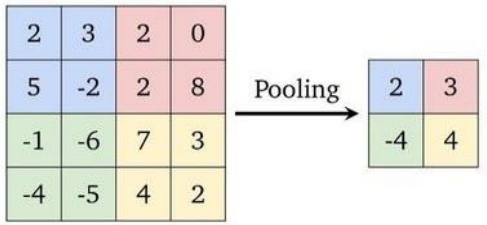
- Pooling layers

- When the input image is large, pooling layers can be very useful
- the goal of pooling is to reduce the dimensions of the image but retaining useful features
- pooling layers **do not have trainable parameters**

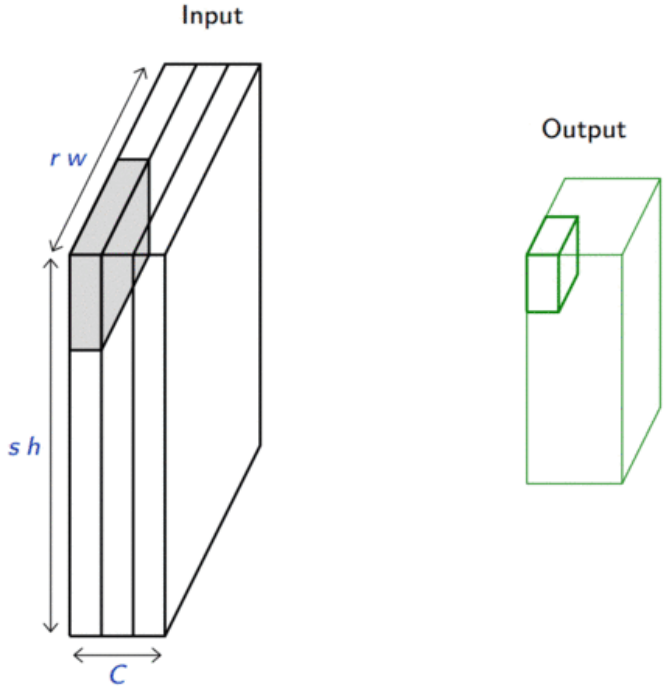
- Most used pooling operations :



Maximum Pooling



Average Pooling



Taken from : Francois Fleuret, EE559 Deep Learning, EPFL

2) How to build a CNN ?

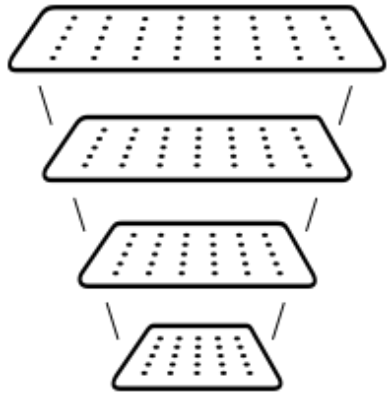
- Let say we want to classify images that are 200x200 into two categories : cats and dogs

- Let us apply a first convolution with k = 3x3, s = 2x2, d = 1x1, p = 0x0

- output : 99x99

- Applying again a convolution with the same parameters

- output : 49x49



- What are the problems of this procedure ?

Cat !

2) How to build a CNN ?

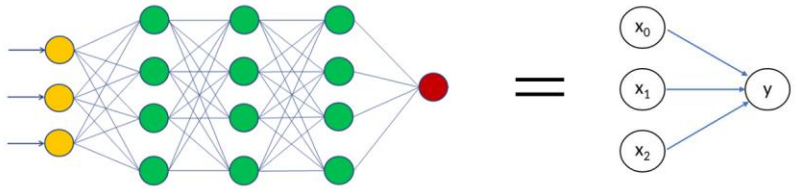
- 1) Convolutions are linear operators

- stacking them do not lead to a more complex relationship

- need non-linearities : activation functions !

• First guess : add more layers ?

➡ composition of linear functions = linear function



- 2) You cannot choose the output of a convolution layer

- need to tune the parameters accordingly (kernel size, stride, dilation, padding)

- large images require a lot of convolution layers to be reduced by a sufficient factor

- how do you go from 2D images to 1D outputs (the classes "dogs" and "cats") ?

```
# Calling the network with the right number of inputs and outputs + Put it on the GPU
N_inputs = batch_x.size(1)
N_outputs = batch_label.size(1)

N_hidden_1 = 1000
N_hidden_2 = 100
N_hidden_3 = 100

my_network = MLP(ngpu, N_inputs, N_hidden_1, N_hidden_2, N_hidden_3, N_outputs).to(device)
```

2) How to build a CNN ?

- What are the ingredients needed to build a CNN ?
  - convolutions
  - activation functions (ReLU, Leaky ReLU, ...)
  - fully-connected layers (FC)
  (- pooling layers)

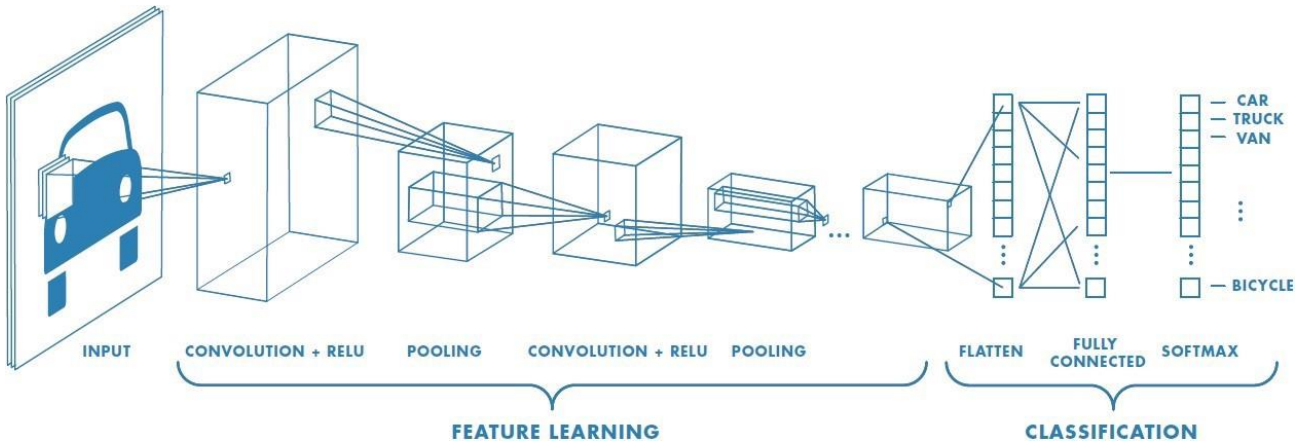
- The most common CNN architectures follow the pattern :

$$\texttt{INPUT} \rightarrow [[\texttt{CONV} \rightarrow \texttt{RELU}]*N \rightarrow \texttt{POOL?}]*M \rightarrow [\texttt{FC} \rightarrow \texttt{RELU}]*K \rightarrow \texttt{FC}$$

- usually $N \leq 3$ (for basic CNN), $M \geq 0$ and $K \leq 3$
- YOLO : N = 13, GoogLeNet : N = 22, ResNet50 : N = 48

2) How to build a CNN ?

- A typical CNN architecture :



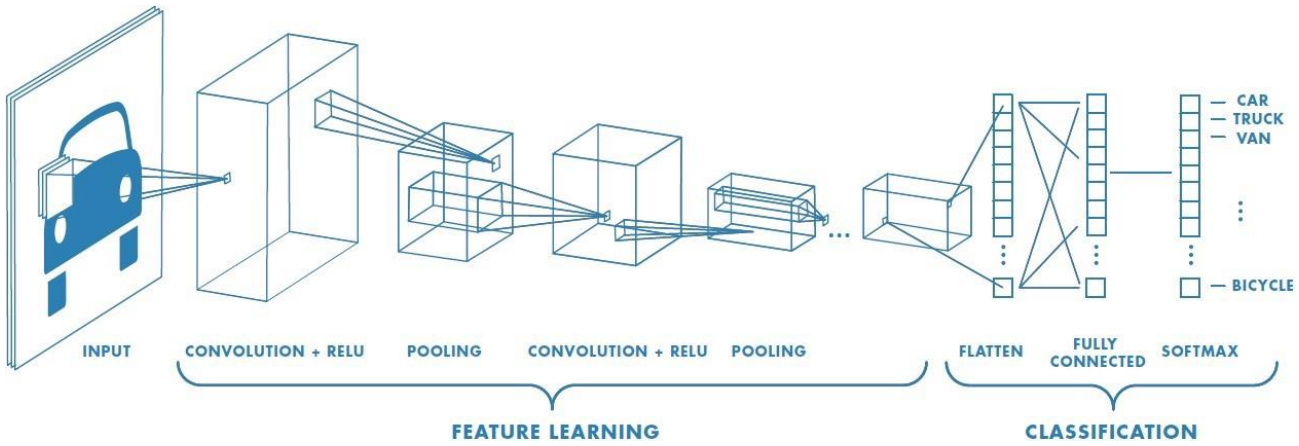
- convolution layers are used to learn some features in the input image

- fully-connected layers gather all these features to produce an output

NB : some CNN do not contain fully-connected layers, it is usually the case when the goal is not classification

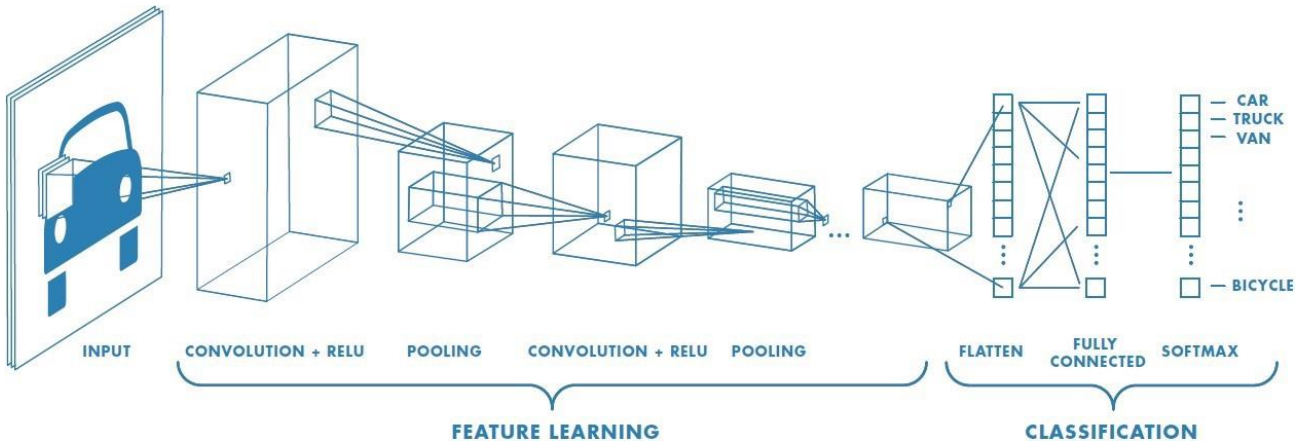
2) How to build a CNN ?

- What does the depth correspond to ?



2) How to build a CNN ?

- What does the depth correspond to ?



- The depth of a layer corresponds to the number of different kernels used in that layer

- the more kernel, the more features can be learned

- but the more kernel, the more parameters to train

2) How to build a CNN ?

- To distinguish and classify objects, animals or scenes lots of aspects are useful

- shape, color, texture, orientation, companion object, ...

- these aspects are often complex and require to be learned in a couple of steps

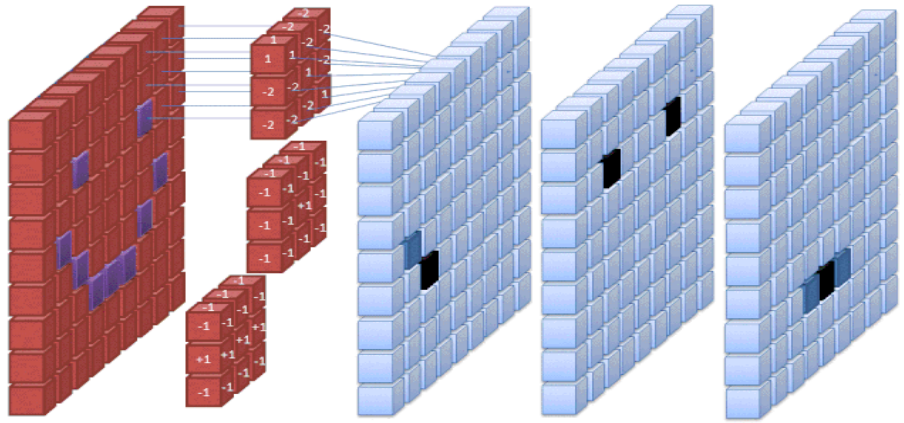
➥ That is why multiple kernels are useful !

- Let us consider applying a convolution on a RGB image

- a 2D kernel slides over each channel (R, G and B)

- the resulting images are summed into a **feature map**

- this is repeated N times with N different kernels, giving N feature maps for this layer (= depth of the layer)



2) How to build a CNN ?

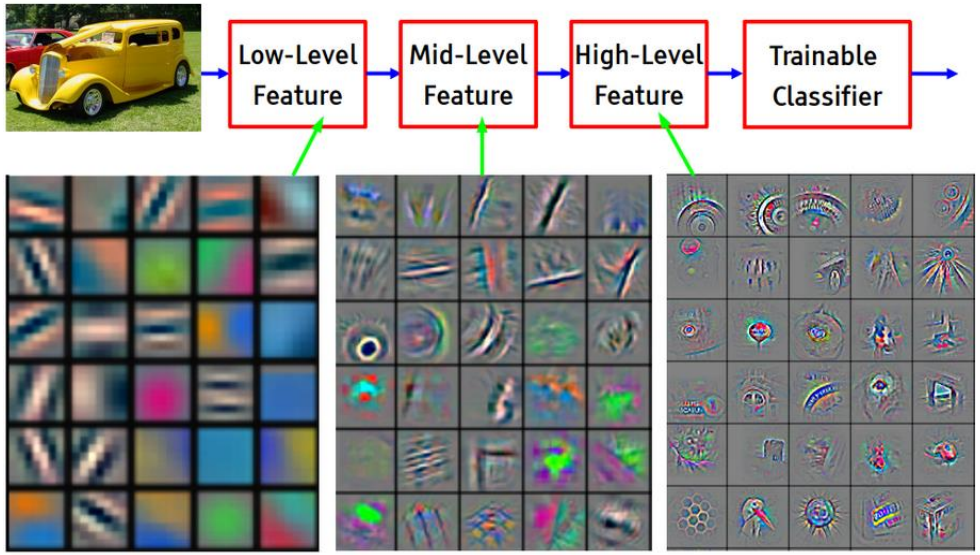
- A composition of multiple convolutions allows the network to learn a **hierarchical composition of patterns**

- Example with the ImageNet feature maps

- first layers appear to encode direction and color

- the direction and color filters get combined into grid and spot textures

- these textures gradually get combined into increasingly complex patterns



Taken from Zeiler & Fergus, 2013

3) Going further with convolutions

- So far, we have seen the basic ingredients that compose convolutional networks

- you can now build simple networks for classification tasks

- if you want to build large network, some additional ingredients will be needed

- Large networks, even with ReLU, suffer from vanishing gradients

$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \; w_2^3 \; f'(h_2^2) \; a_2^1$$

$-1 \leq w \leq 1 \qquad \leq 0.25 \qquad \leq 1$

With Sigmoid

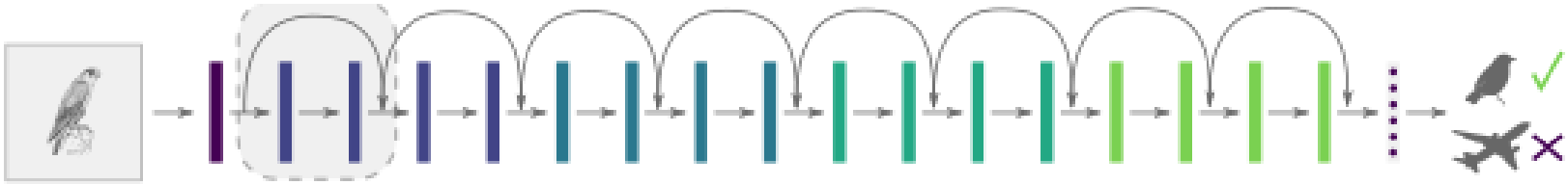
$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \; w_2^3 \; f'(h_2^2) \; a_2^1$$

$-1 \leq w \leq 1 \qquad = 1 \qquad = 1$

With ReLU

3) Going further with convolutions

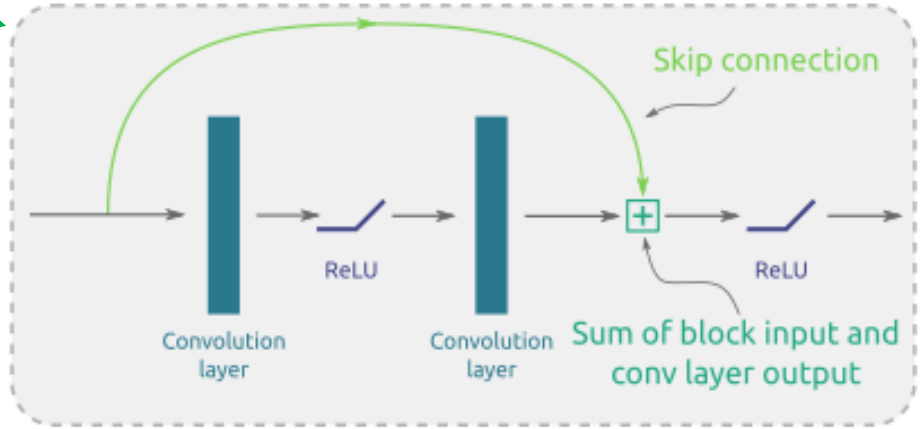
- Training large neural networks is made possible thanks to **skip connections**



- They allow the gradients to shortcut the layers and pass through without vanishing

$$\frac{\partial \mathcal{L}}{\partial w_{22}^2} = \frac{\partial \mathcal{L}}{\partial S} \; w_2^3 \; f'(h_2^2) \; a_2^1$$

$-1 \leq w \leq 1 \qquad = 1 \qquad = 1$



3) Going further with convolutions

- For some applications, you do not want to reduce the size of the image, rather you want to have images as outputs

Image segmentation

Super-resolution

- Convolutions are used to downscale the useful information

➥ Can we reverse the convolution operation ?

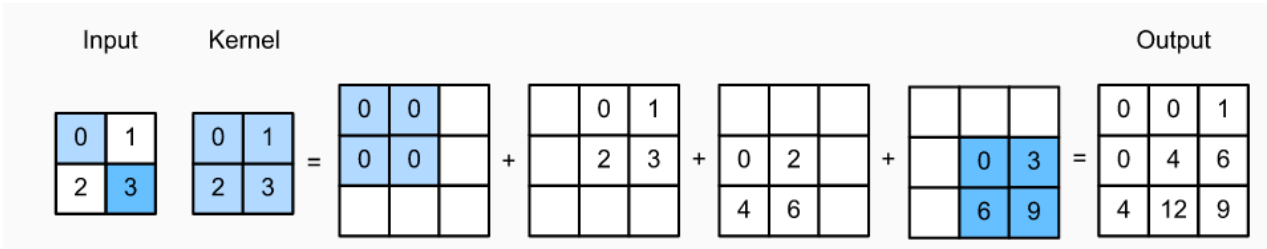
3) Going further with convolutions

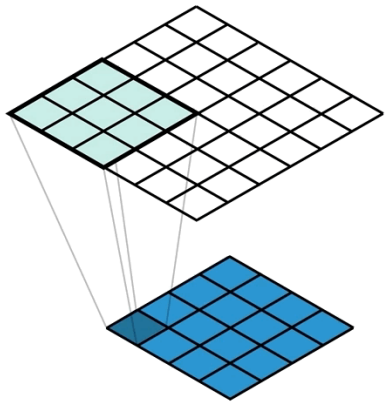
- Transposed convolutions

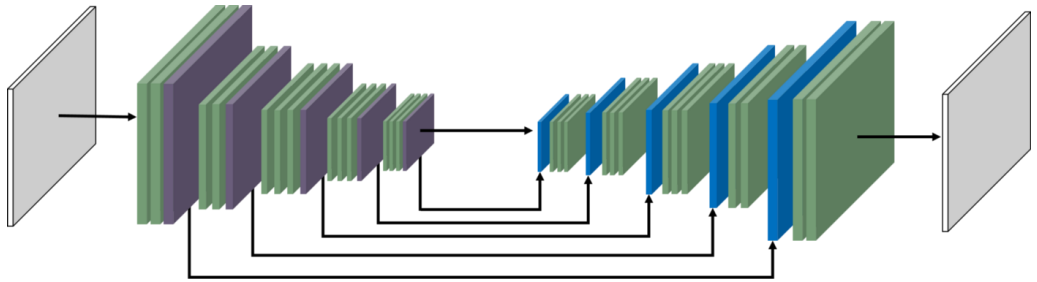
- used to increase the size of the data

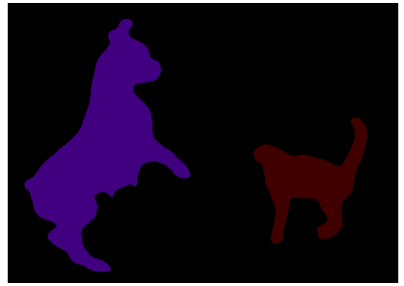
- different from deconvolution (that is the exact inverse convolution)

- generally used together with convolutions









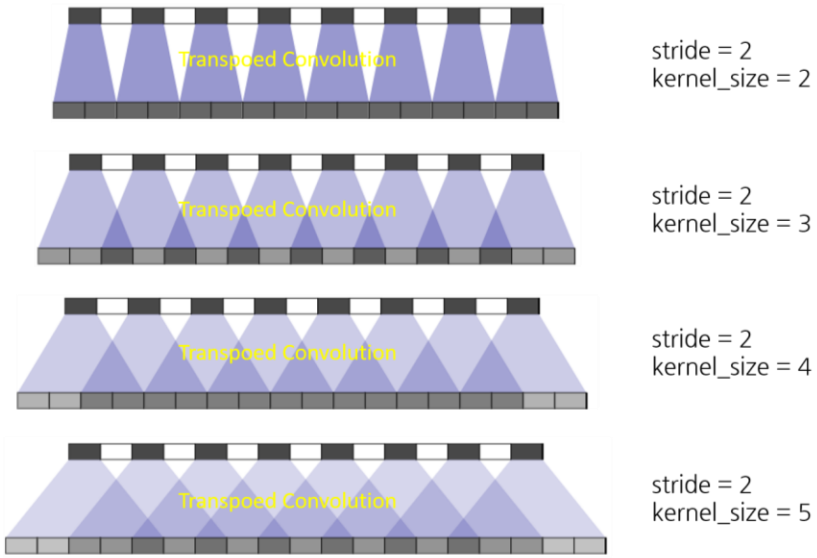
3) Going further with convolutions

- Transposed convolutions

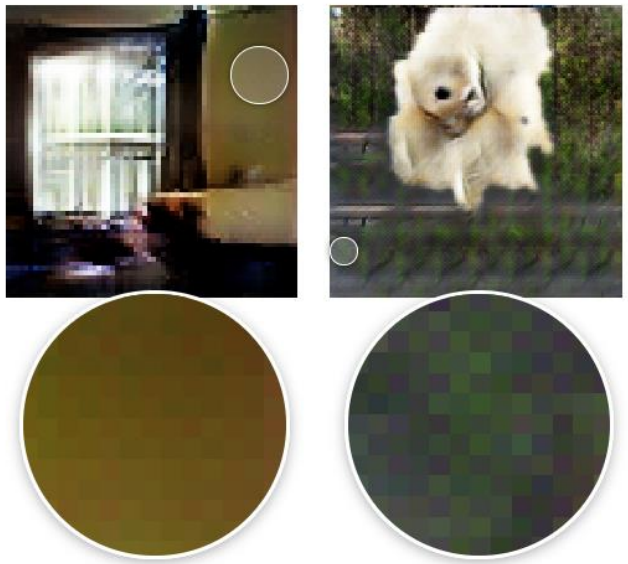
- suffer from checkerboard effects (can be reduced by using a kernel size divisible by the stride)

- can prevent the network to converge (leading to funny images...)

- sometimes other upscaling strategies are preferred



Taken from Jinsol Kim, 2022



Taken from Odena, et al., 2016

3) Going further with convolutions

- Transposed convolutions are not always the best solution to upscale the information

- suffer from checkerboard effects

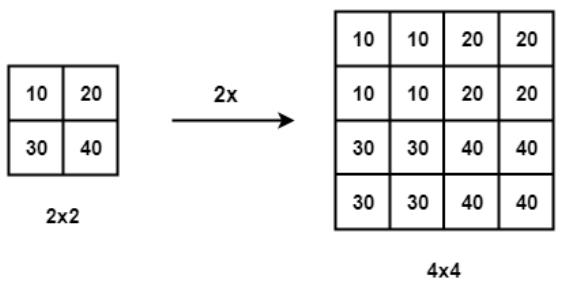
- involve trainable parameters

- Upsampling methods can be used to this purpose !

- only one parameter to tune : upsampling factor

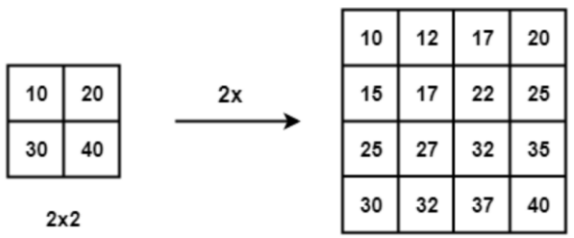
- no trainable weights

Nearest Neighbors



2x2 → 4x4

Bilinear Interpolation



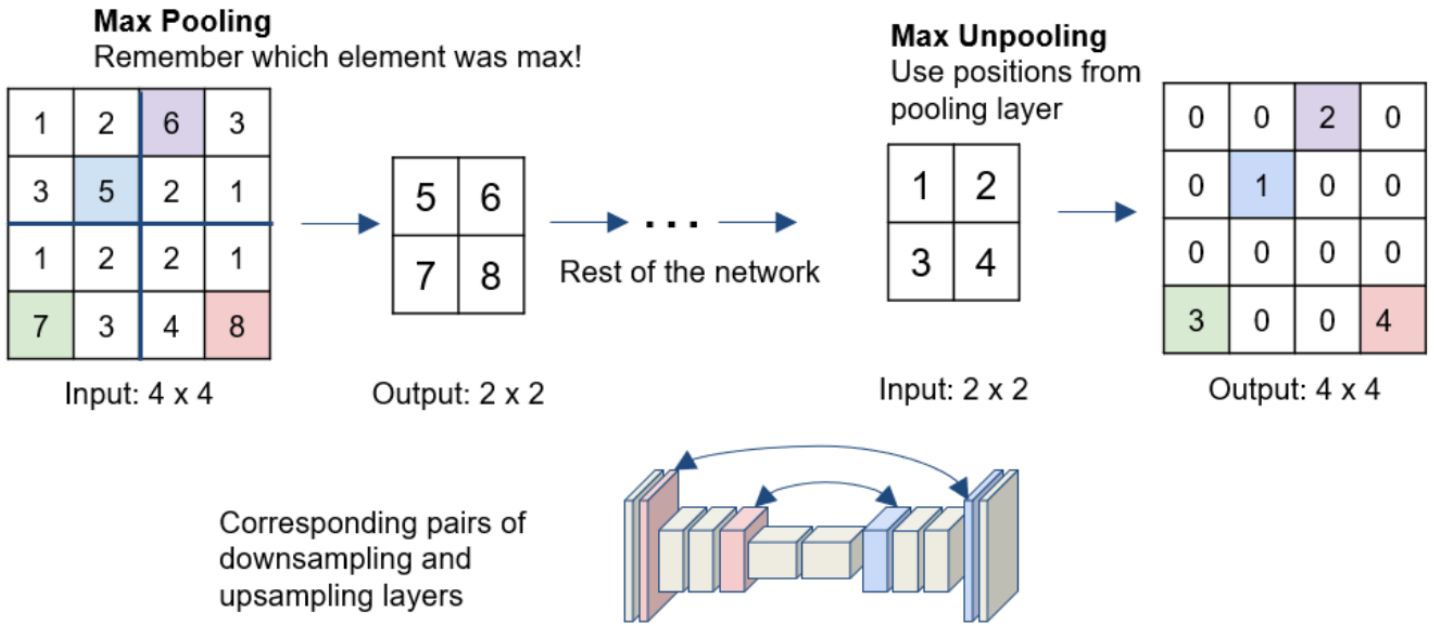
2x2 → 4x4

3) Going further with convolutions

- Another upsampling method : unpooling

- exact inverse of pooling

- used together with pooling layers in two-sided network

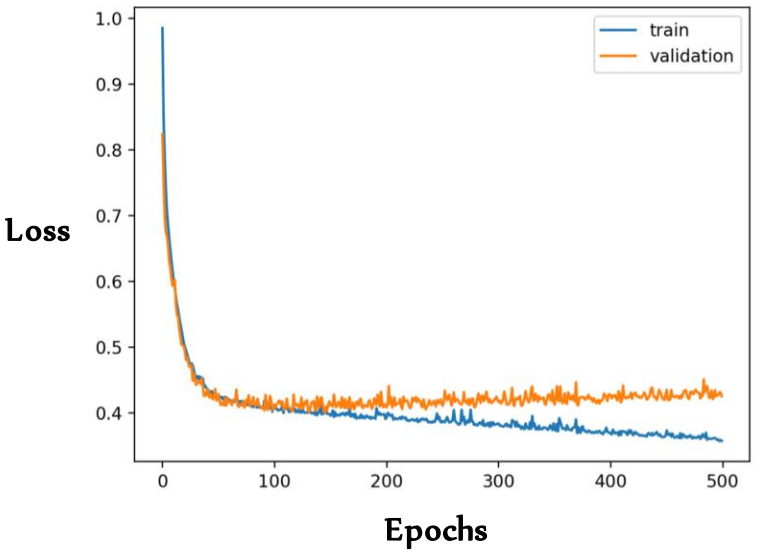


Corresponding pairs of downsampling and upsampling layers

3) Going further with convolutions

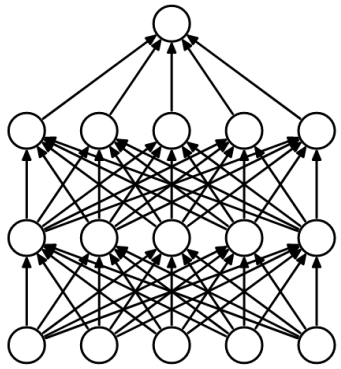
- Increasing the depth of your CNN can lead to overfitting

- because you increase the capacity of your network
- for some tasks, it can learn the statistical noise in your data
- this improves the training curves but fails on the validation curves

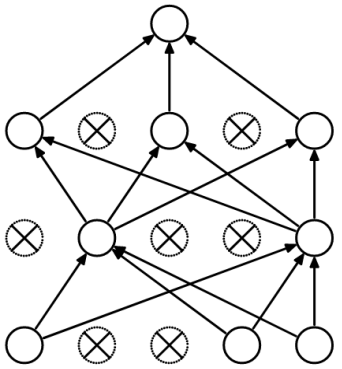


- Dropout has been introduced to solve this problem

- dropout consists in "dropping" out some nodes
- each node has a probability p to be dropped
- for a layer of 1000 nodes, if p=0.5, 500 nodes will be dropped
- dropout is activated only during the training phase



(a) Standard Neural Net



(b) After applying dropout.

Taken from :
Srivastava, N. 2014

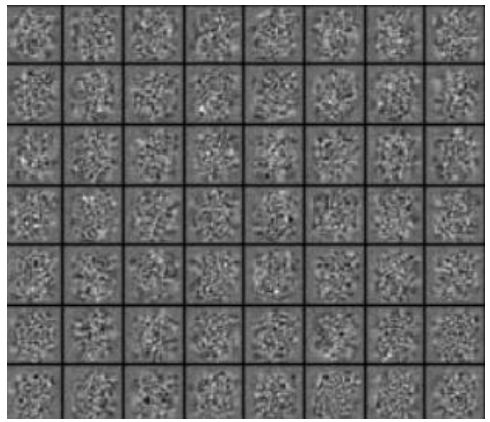
3) Going further with convolutions

- Dropout is part of the regularization methods

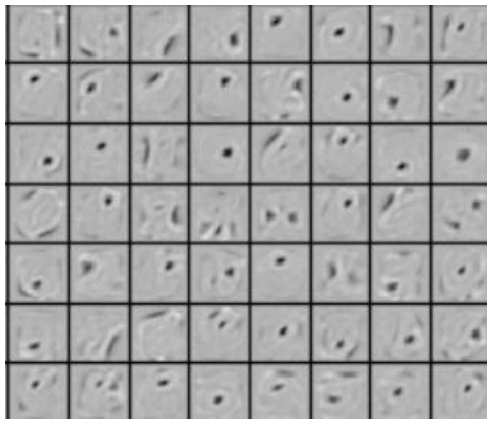
- regularization consists in discouraging the learning of a more complex model to prevent overfitting

- bring sparsity in the network layers

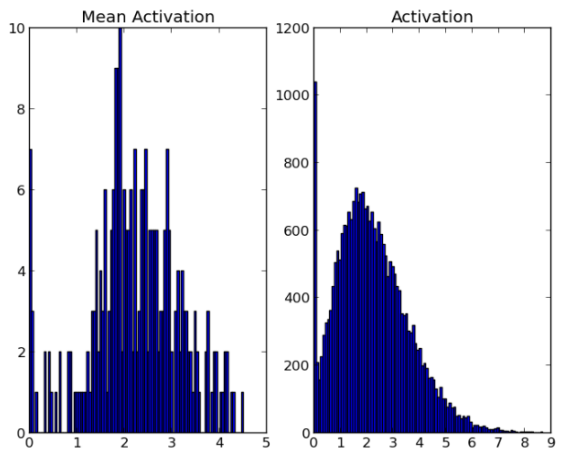
- the nodes focus more on learning the generalized features



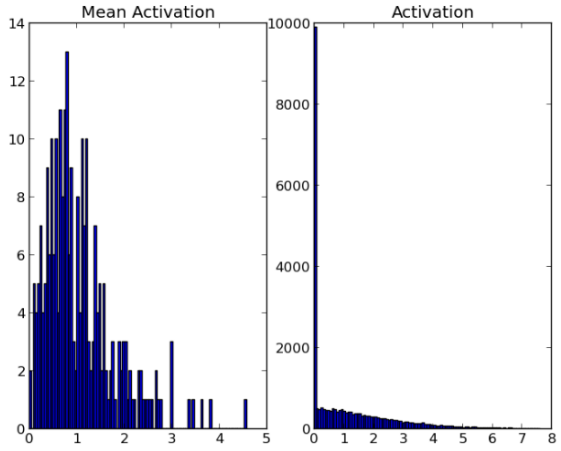
Without dropout



With dropout



Without dropout



With dropout

Taken from : Srivastava, N. 2014

4) Tips and tricks to train NN better

- The lack of data is the biggest limit to the performance of deep learning models

- collecting more data is usually expensive and laborious

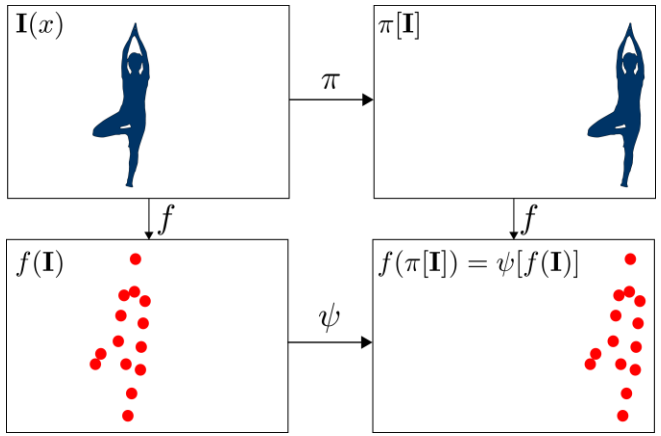
- synthesizing data is sometimes complicated and may not represent the true distribution

- How can we create larger datasets in a simple and efficient way ?

- convolutions are translation invariant, they are translation equivariant

- but they are not equivariant to other transformations

- Data augmentation consists in applying transformations to your data to create an artificially larger dataset



Taken from Divyanshu M. 2020

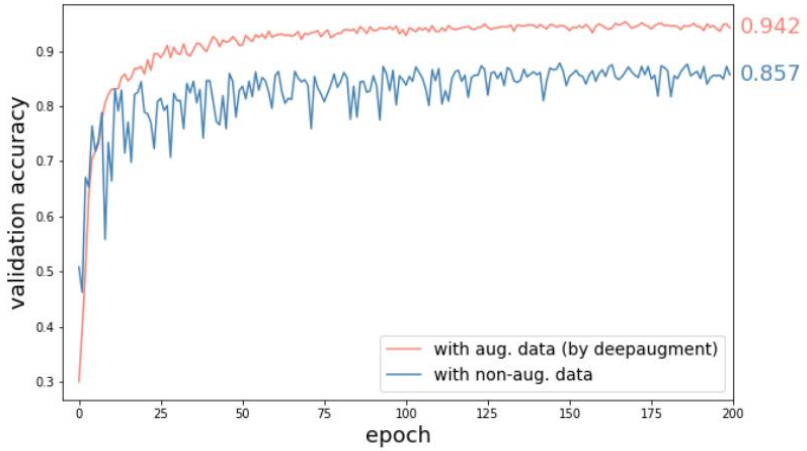
4) Tips and tricks to train NN better

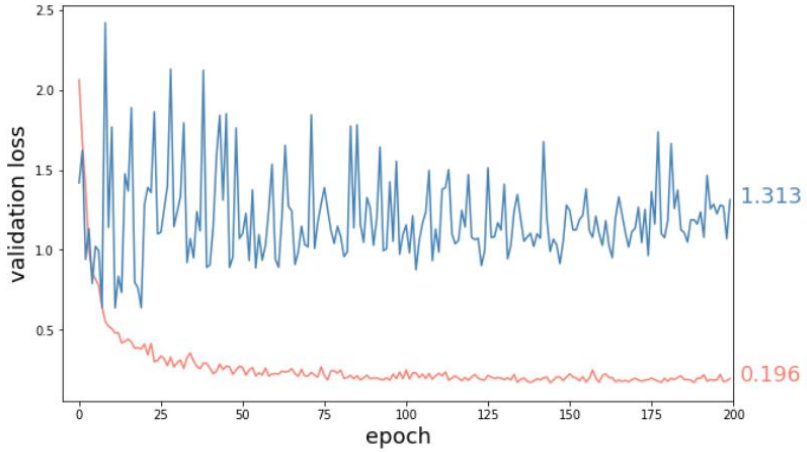
4.1) Data augmentation
4.2) Transfer learning

- Some of the transformations used

- rotation, flipping and cropping are the most used

- noise can also be added to the weights, gradients or activations





Taken from [DeepAugment, 2020.]

4) Tips and tricks to train NN better

- Training deep models from scratch on millions of images can take days or weeks

- sometimes you do not have the resources required to train large models for a long period

- or you lack experience to build deep networks

- Why can't you take advantage of the resources of others ?

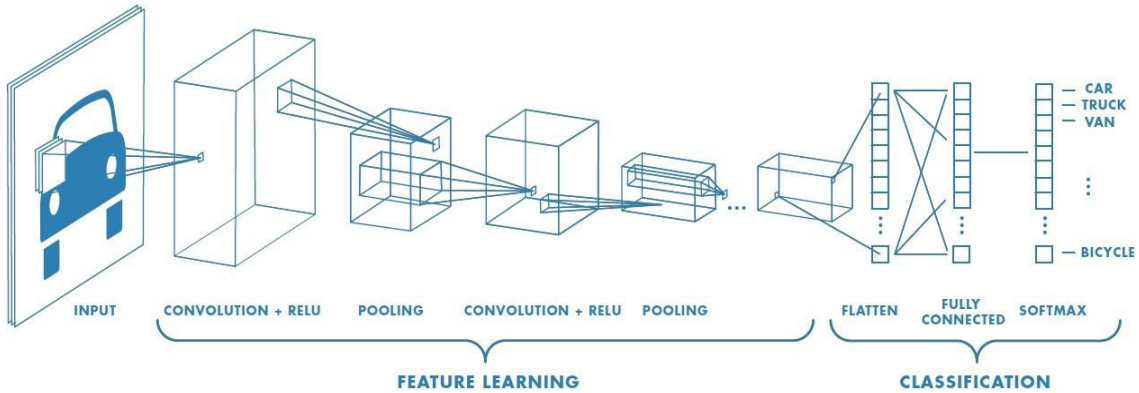
- many trained models are publicly available

- they can be used as features extractors for your own problem !

- There exists two ways :

- transfer learning

- fine-tuning



4) Tips and tricks to train NN better

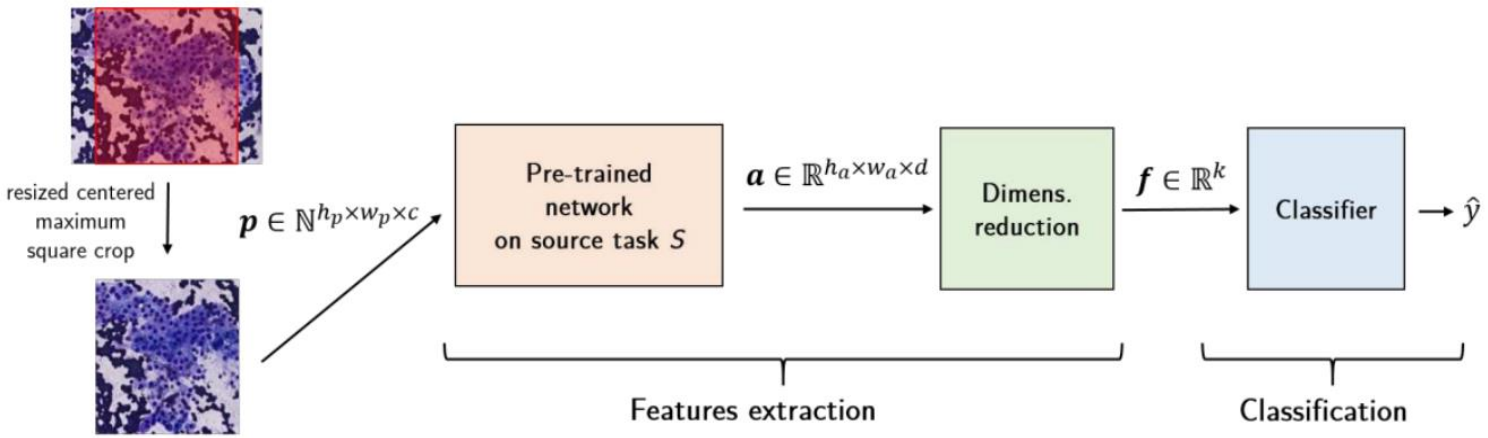
- Transfer learning

- consists in taking a pre-trained network, remove the last layer(s) and take the rest of the network as a fixed feature extractor

- the only trainable weights are in your own layers at the end

- generally better than training from your own data only

- requires sometimes to adjust your data (resizing, resampling, ...)



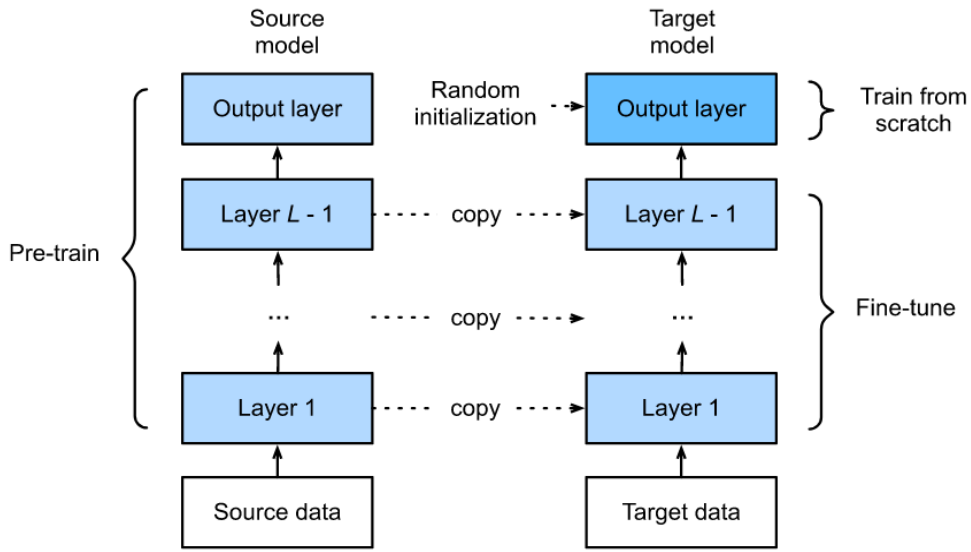
Taken from Mormont et al, 2018.

4) Tips and tricks to train NN better:

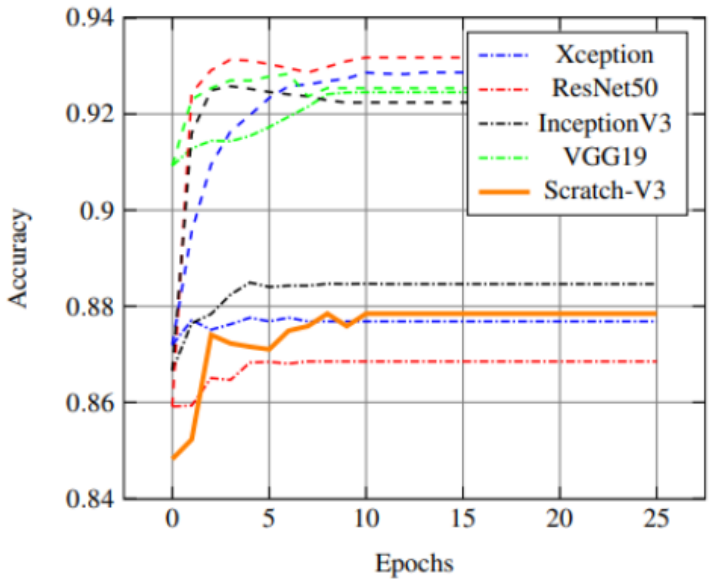
- Fine-tuning

- Nearly identical to transfer learning but the weights from the pre-trained network are also fine-tuned

- all or only some of the layers can be tuned



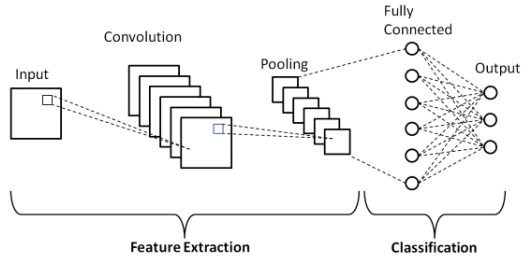
Taken from Dive Into Deep Learning, 2020.



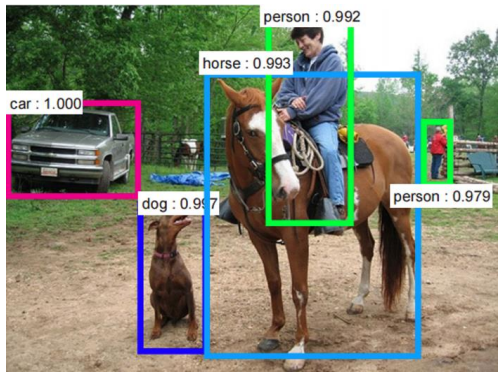
Taken from Matthia Sabatelli et al, 2018.

5) More advanced networks

- Let us consider a problem where you want to classify images
  - we know how to do it if there is only one object in the image
  - in the case of multiple objects, our approach does not work anymore
  - the ideal goal would be to separate the objects in the image



- To achieve that goal, we can use bounding boxes
  - this implies finding the location AND the class of an object
  - these two problems are not easy two solve together !



- .... or segmentation !
  - it consists in finding a label for every pixel in the image
  - can transposed convolutions be useful ?

5) More advanced networks

- Bounding boxes : YOLO (You Only Look Once – Redmon et al, 2015.)

- YOLO is one of the best object detection algorithm

- it considers object detection as a regression problem

- from a pre-identified set of bounding boxes and confidence values, it select the best ones

- Many engineering choices :

- the network is pre-trained on the ImageNet dataset

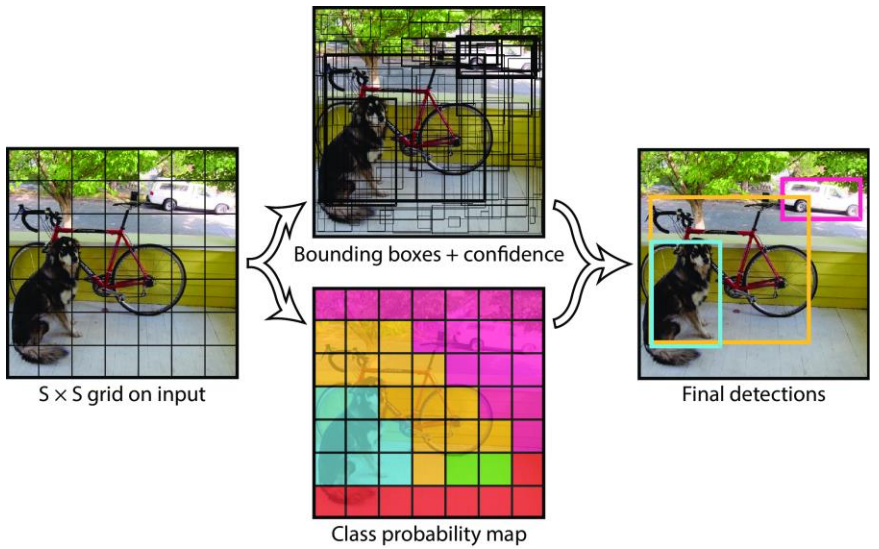
- use Leaky ReLUs for all layers

- data augmentation with scaling and color transformation

- normalize the bounding box parameters in [0,1]

- dropout after the first convolutional layer

- reduce the weight of large bounding boxes by using the square roots of the size in the loss



5) More advanced networks

- Bounding boxes : YOLO (You Only Look Once – Redmon et al, 2015)

- the input image is divided into a grid (S x S) and for each grid cell, the network predicts B bounding boxes, a confidence value for each box and C class probabilities

- the loss designed to train YOLO is quite complex and contains many terms achieving different purposes

Regression loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

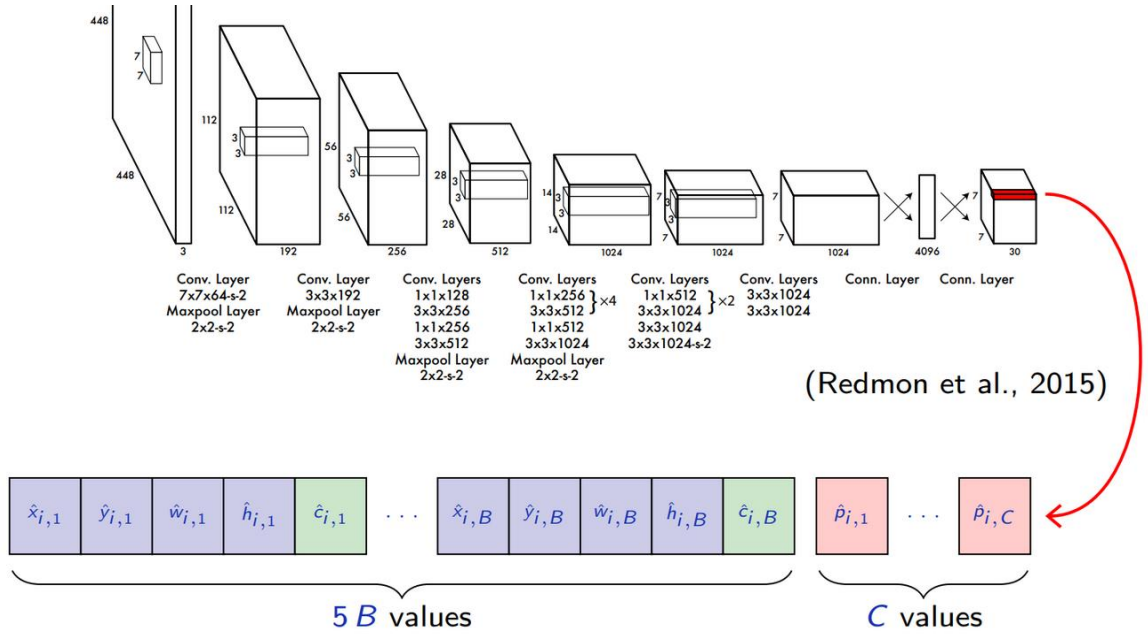
Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$



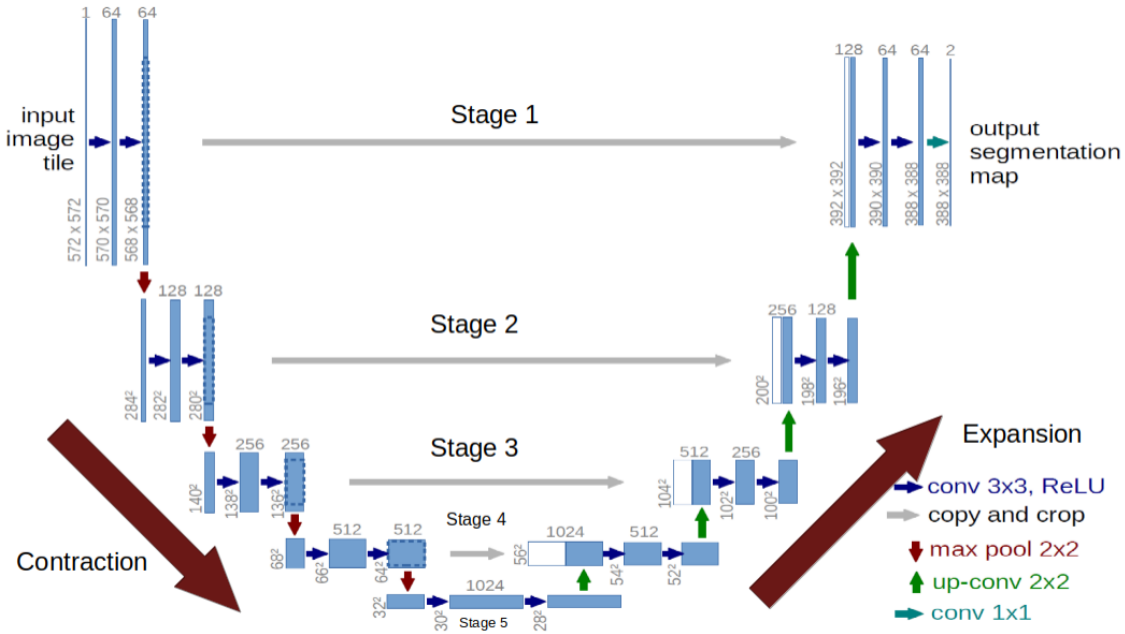
Taken from : Francois Fleuret, EE559 Deep Learning, EPFL

5) More advanced networks

- Segmentation : UNet (Ronneberger et al, 2015)

- built as a fully convolutional network

- consists in symmetric contraction and expansion paths, with connections from the contracted part to the expanding part



Taken from Ronneberger et al, 2015.

5) More advanced networks

- To go further...

- Object detection :

- Single Shot Multi-box Detector (SSD, Liu et al, 2015) : improves over YOLO by using a fully convolutional network

- Region-based CNN (R-CNN, Girshick et al, 2014) : instead of producing a large set of bounding boxes, region proposals are extracted from the image

- Segmentation :

- Mask R-CNN (He et al, 2017) : extends the R-CNN model for segmentation

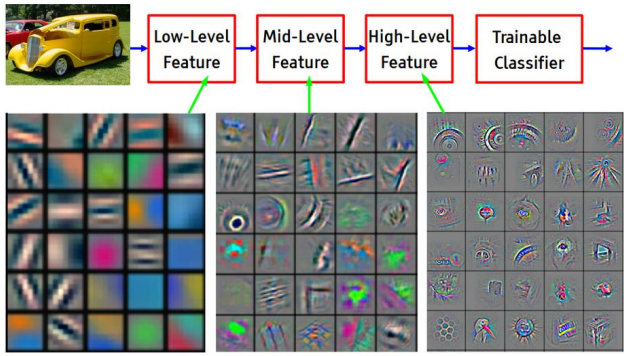
5) More advanced networks

- How can you be sure your network detects what it is supposed to ?

- sometimes CNNs do not work as expected

- a deep network can overfit and learn the statistical noise in the data

- you can look at the feature maps to see how the layers are activated



Taken from Zeiler & Fergus, 2013

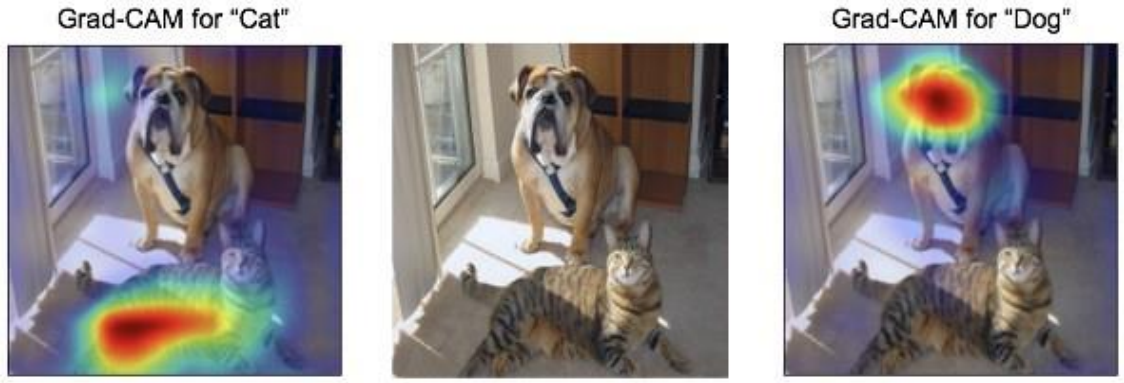
- ...or let a neural network do it for you : Grad-CAM

- Grad-CAM is a neural network producing heatmaps

- check how the gradients flow through a network to trace back the important regions in the input image

- you can trace back the gradients to any feature map

- easily adaptable to any CNN

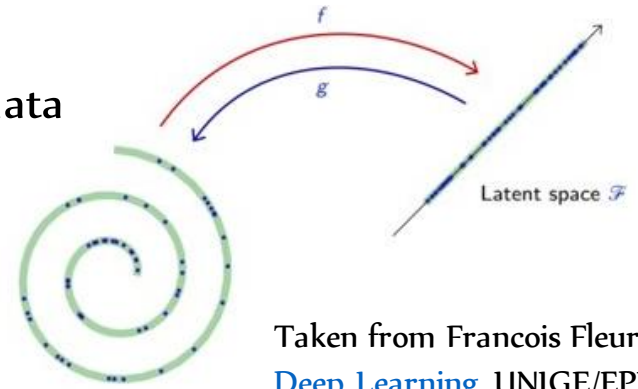


5) More advanced networks

- Generative Adversarial Networks (GANs) and Autoencoders

- the goal of these networks is both to learn the underlying distribution in the data AND to produce realistic data

- they mainly differ by the way they are trained



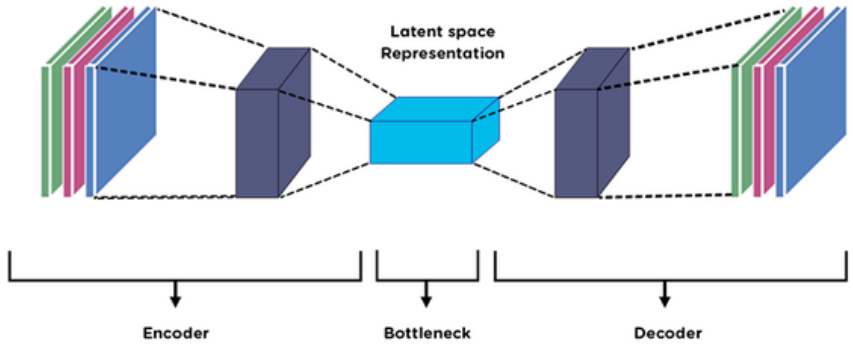
Taken from Francois Fleuret, Deep Learning, UNIGE/EPFL

- Autoencoders

- made of two networks : an encoder and a decoder

- the networks can either be MLPs or CNNs

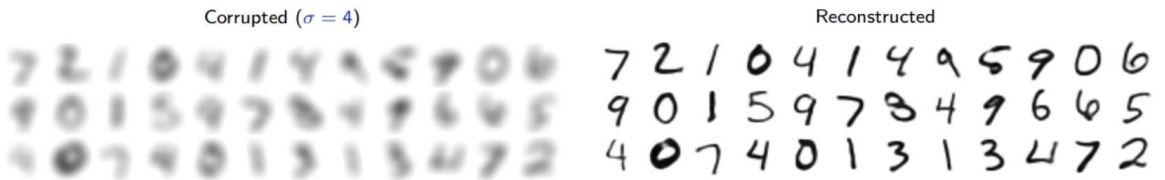
- often used to denoise data



Taken from Deepak Birla, 2019.

- do not really work as it is

➥ Variational Autoencoders

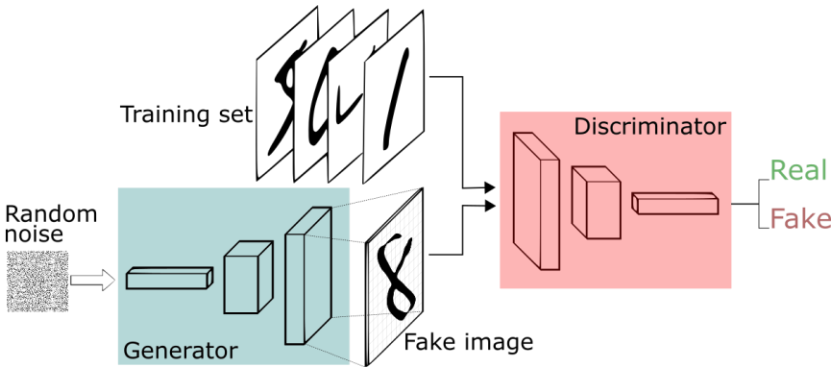


5) More advanced networks

- Generative Adversarial Networks

- the generator is responsible for the generation of new images from a random noise vector (latent space)
- the discriminator tries to discriminate the generated images from the real ones
- the training is a competition between these two networks

- GANs are very powerful but very hard to train



Taken from Thalles Silva, 2018.

Taken from Karras et al, 2018.

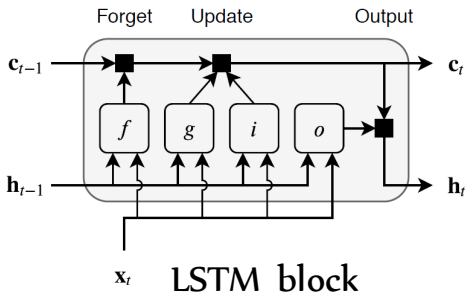
5) More advanced networks

- How to make sense of sequential data ?

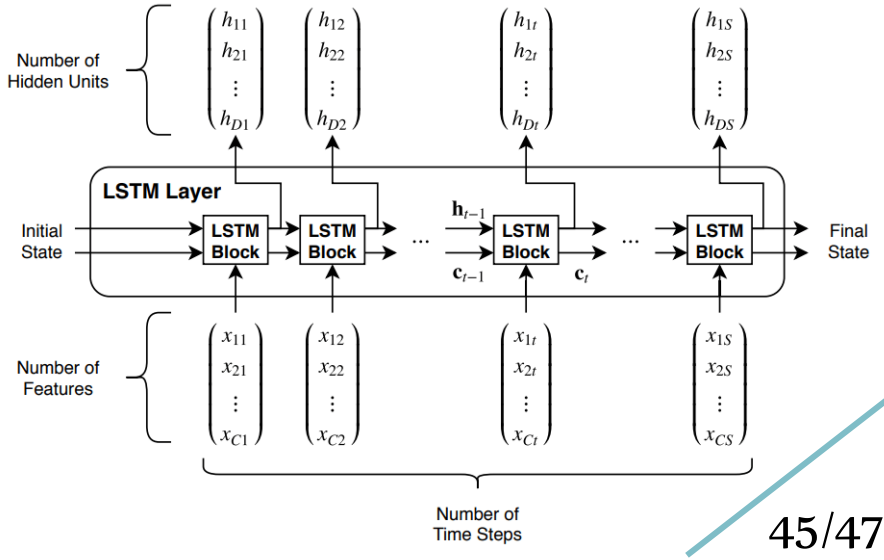
- examples : activity recognition in videos, text translation, speech recognition, ...
- the analysis of some situations can change with time

- Recurrent networks have been designed to address data sequentially

- they maintain a recurrent state, updated at each step
- predictions can be computed at each step
- implement gating, similar to skipped connections
- examples : LSTM and GRU
- might be hard to train for long sequences



LSTM block



5) More advanced networks

- Transformers are very different from other neural networks

- composed of an encoder and a decoder

- they implement the attention mechanism, consisting in transporting information from parts of the input signals to parts of the output specified dynamically

- attention layers produce weights that are functions of the inputs

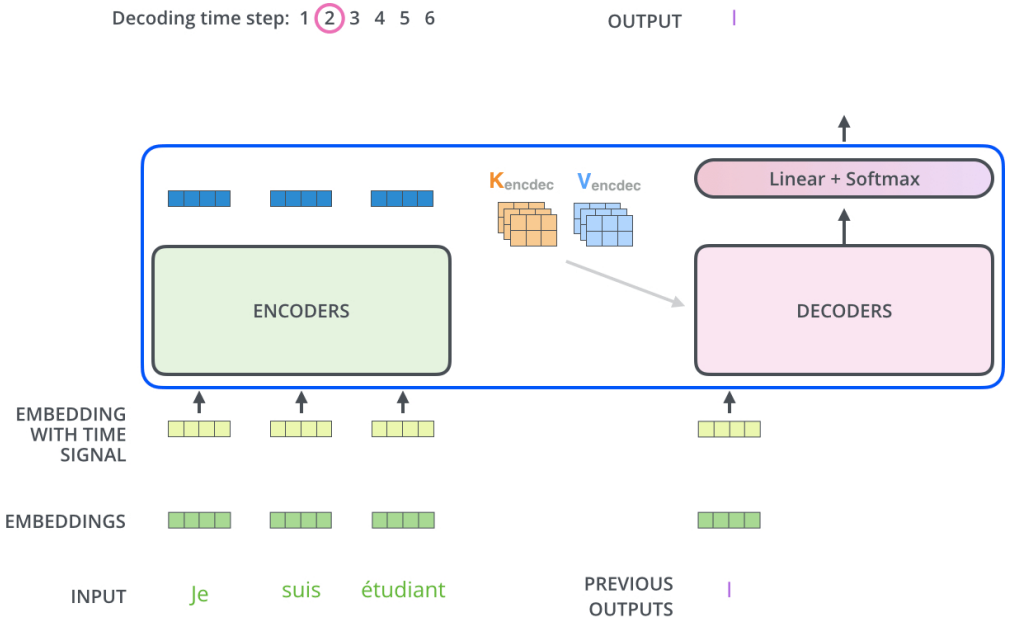
- require input embedding

- Steps :

- The encoders start by processing the input sequence.

- The output of the encoder is then transformed into a set of attention vectors K and V that will help the decoders focus on appropriate places in the input sequence.

- Each step in the decoding phase produces an output token, until a special symbol is reached indicating the transformer decoder has completed its output.



- The deep learning jargon...

GradCAM

Batch size

Overfitting >< underfitting

Convolution layers

Pooling

Vanishing gradients

Loss function

Activation function

Transfer learning

RNN

Learning rate

Normalization

Optimizer

Transposed convolution

Backpropagation

Gradient descent

Fully-connected

ReLU

YOLO

Initialization

Course 3 : The end

Vincent Boudart, PhD student

vboudart@uliege.be