

# Deep Learning

---

Equancy  
November 7, 8, 9 2017

course given by  
Hagop Boghazdekian



equancy

# Nice to meet you!



Hagop Boghazdeklan  
Data Scientist at Equancy  
R&D in Deep Learning/Image Processing  
Double M.Sc in Econometrics & Data Science

# Disclaimer

- First steps in Deep Learning (DL)
- Goals:
  - Learning foundations of DL
  - Understanding/managing neural networks
  - Lead successfully DL projects
- You'll master not only the theory, but also see how to apply it in the industry by learning keras and tensorflow

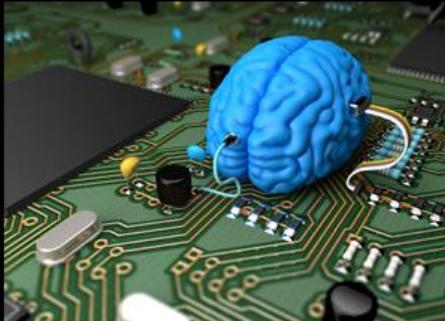
# Disclaimer

- No need to be afraid, it's not that complicated !

## Deep Learning



What society thinks I do



What my friends think I do



What other computer scientists think I do



What mathematicians think I do



What I think I do

```
In [1]:  
import keras  
Using TensorFlow backend.
```

What I actually do

# AGENDA

- ❖ Introduction
- ❖ Challenges Motivating Deep Learning: ML vs DL
- ❖ Theory of Deep Neural Networks
  - Neural Networks
  - Regularization for DL
  - Optimization for Training DL models
  - CNN
  - RNN
  - Transfer Learning, Reinforcement Learning
- ❖ Practicing : train your own networks using keras
  - Image Classification
  - Sentiment Analysis
  - Fisheries kaggle competition
- ❖ Conclusion



1

# Introduction

# What's Deep Learning ?

A subset of Machine Learning

## ARTIFICIAL INTELLIGENCE

Machines that have some of human characteristics (very specific)

## MACHINE LEARNING

Training a model using data so that it learns a specific task.

## DEEP LEARNING

A particular way to train models that recently gave great results

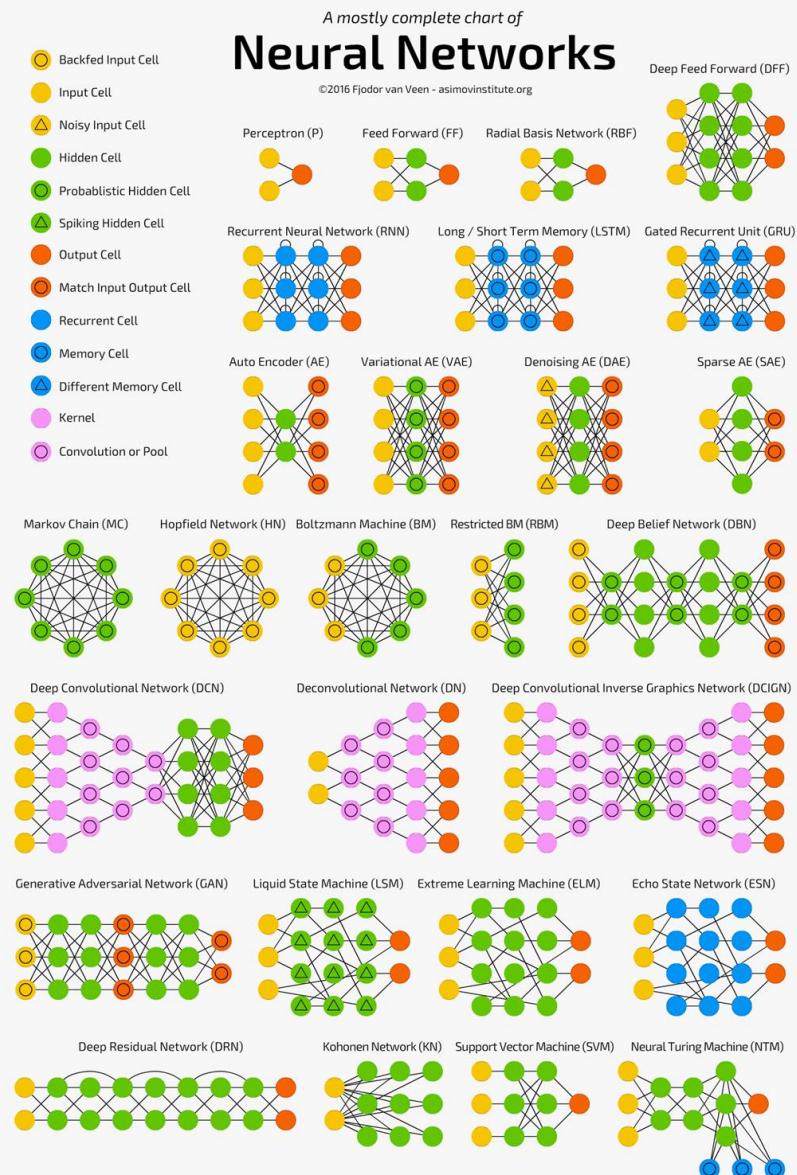


- Used when classical ML algorithms do not succeed
- To solve central problems in AI:
  - Image recognition
  - Natural Language Processing (Part of Speech, Named Entity Recognition, ...)

# Neural Networks

## The core of Deep Learning

- Using Deep Learning often implies training Deep Neural Networks
- Incredibly large number of possible networks ...
- Heavy theory & mathematical foundations
- Highly flexible & sensitive algorithms



# Why Deep ?

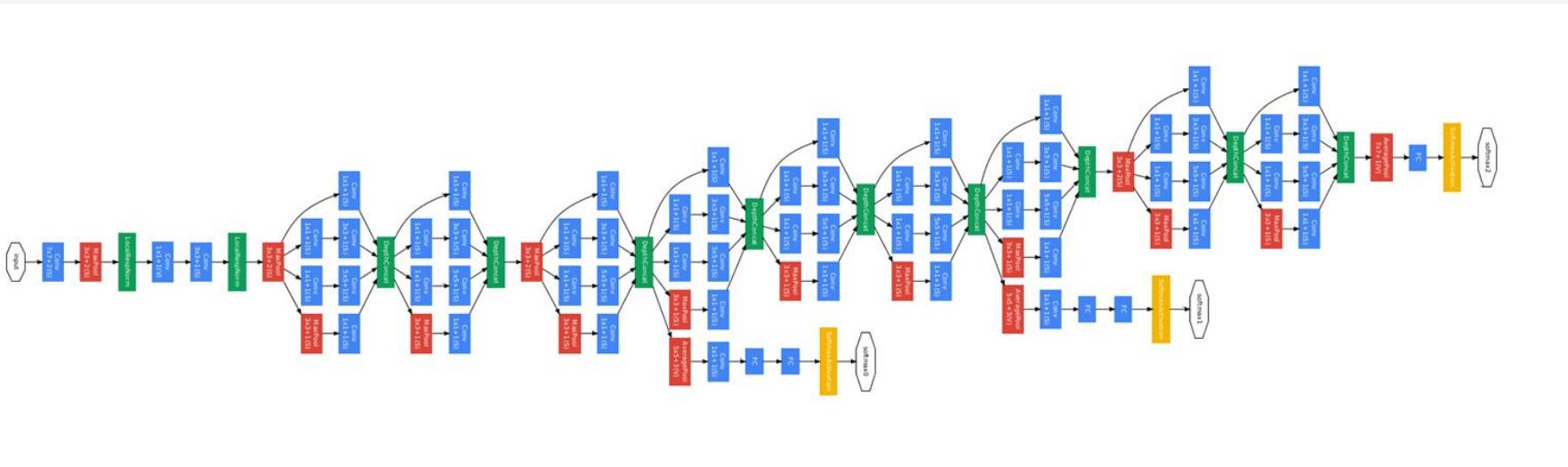
Complex and deep models

- The Deepness refers to the complexity of the Neural Networks
- To simplify! Each time we want to describe a modern deep network

It looks like this ...



GoogleNet - Inception



# Who's using DL ?

All major companies



YAHOO!

Google



IBM



Baidu 百度

# Who's using DL ?

Specialized start-ups ... it's booming!



## 60+ STARTUPS USING DEEP LEARNING

### CORE AI: COMPUTER VISION

clarifai Tractable netra HYPERVERGE  
captricity deepomatic sentient Sighthound  
terraLOUPE imagenil piloPai

### CORE AI: OTHER

TERADEEP affectiva\* Wave Computing  
DigitalGenius LEAPMIND Arya  
heuro twentybn

### BI, SALES & CRM

SKYMIIND TalkIQ intranetum ripjar  
FRACTAL INDUSTRIES MARIANA DEEPGRAM  
True AI KOVID ditto Gridspace

### CORE AI: VOICE INTERFACE

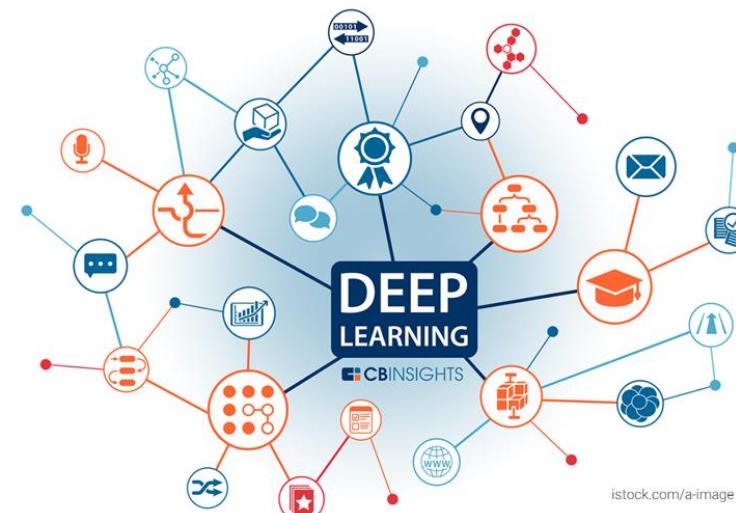
VIV Mobvoi Maluuba  
OPEN

### ROBOTICS & AUTO

netadyne AdasWorks  
drive.ai Rokid 图灵机器人 TURING ROBOT  
comma.ai

### HEALTHCARE

imagiA Mindshare MEDICAL BAYLABS  
BUTTERFLY Network, Inc. deep genomics Atomwise zebra MEDICALVISION  
PATHWAY GENOMICS AVALON Behold.ai MediMatch  
ADRENIO enlitic Lunit SIG TUPLE



### SECURITY

deepinstinct Umbo CV  
signalsense

### OTHER

Alpaca Indico  
Iris Automation

### E-COMMERCE

Reflektion CORTEXICA  
VISENZE STAQU

### ACQUIRED (2014-2016YTD)

Etsy Blackbird Orbeus AlchemyAPI  
intel Movidius nervana SYSTEMS  
Perceptio vocaliq turi  
GOOGL DEEPMIND DNNresearch Dark Blue Labs Vision Factory  
MAGICPONY TECHNOLOGY MADBITS MetaMind

CB INSIGHTS

# What for ?

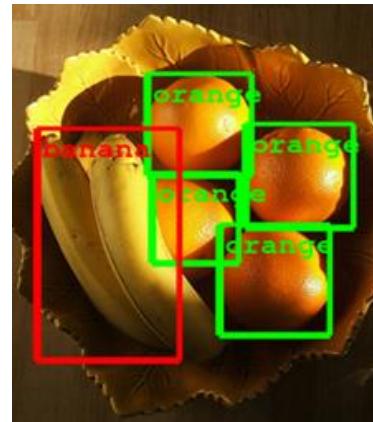
## Image recognition

### Image Classification

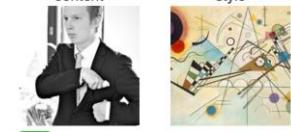


Persian cat : 0.45  
Egyptian cat : 0.22  
Hamster : 0.09  
Lynx : 0.04

### Object detection



### Artist Imitations



### Image Segmentation



### Caption Generation



The man at bat readies to swing at the pitch while the umpire looks on.

### Image Generation



# What for ?

## Natural Language Processing

### Part of Speech Tagging

**Text:**

The grand jury commented on  
a number of other topics

**Legend:**

article, adjective, noun,  
preposition, verb

### Named Entity Recognition

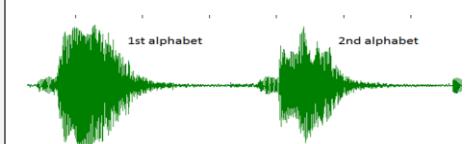
**Text:**

Apple CEO Tim Cook introduces  
the latest iphone version at the  
Cupertino Flint Center event in  
California

**Legend:**

Organization, Location, Person

### Speech Recognition

**Audio:****Text:**

Winter is coming !

### Translation

French

Bonjour à tous !

Chinese



### Sentiment Analysis

**Text 1:**

Today was a great class about  
Deep Learning ! #cepe #dl

**Text 2:**

We've had trouble understanding  
what the guy was talking about  
#equancy #cepe. It was bad.



# Is it new ?

---

Not really ... but we can finally exploit it!

- Neural Networks - studied since the end of the 1950's
- Computationally heavy
- A reborn in the second part of 2000's:
  - Significant computational power
  - Explosion of Data
  - Highly implicated community

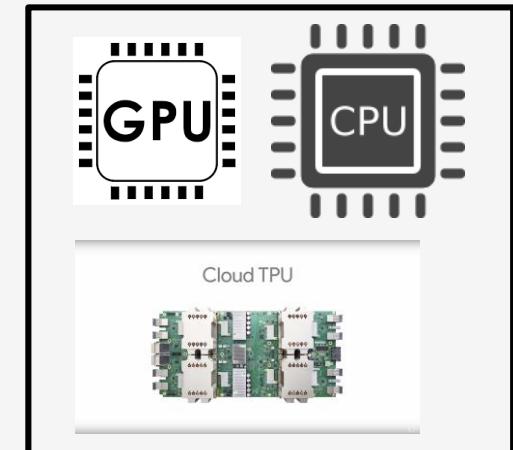
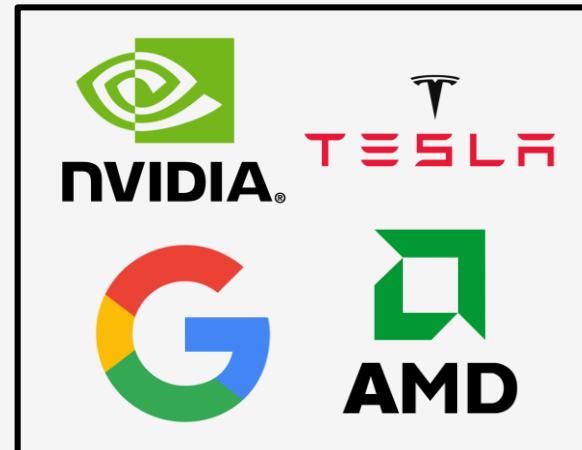
# Is it new ?

Not really ... but we can finally exploit it!

Significant computational power



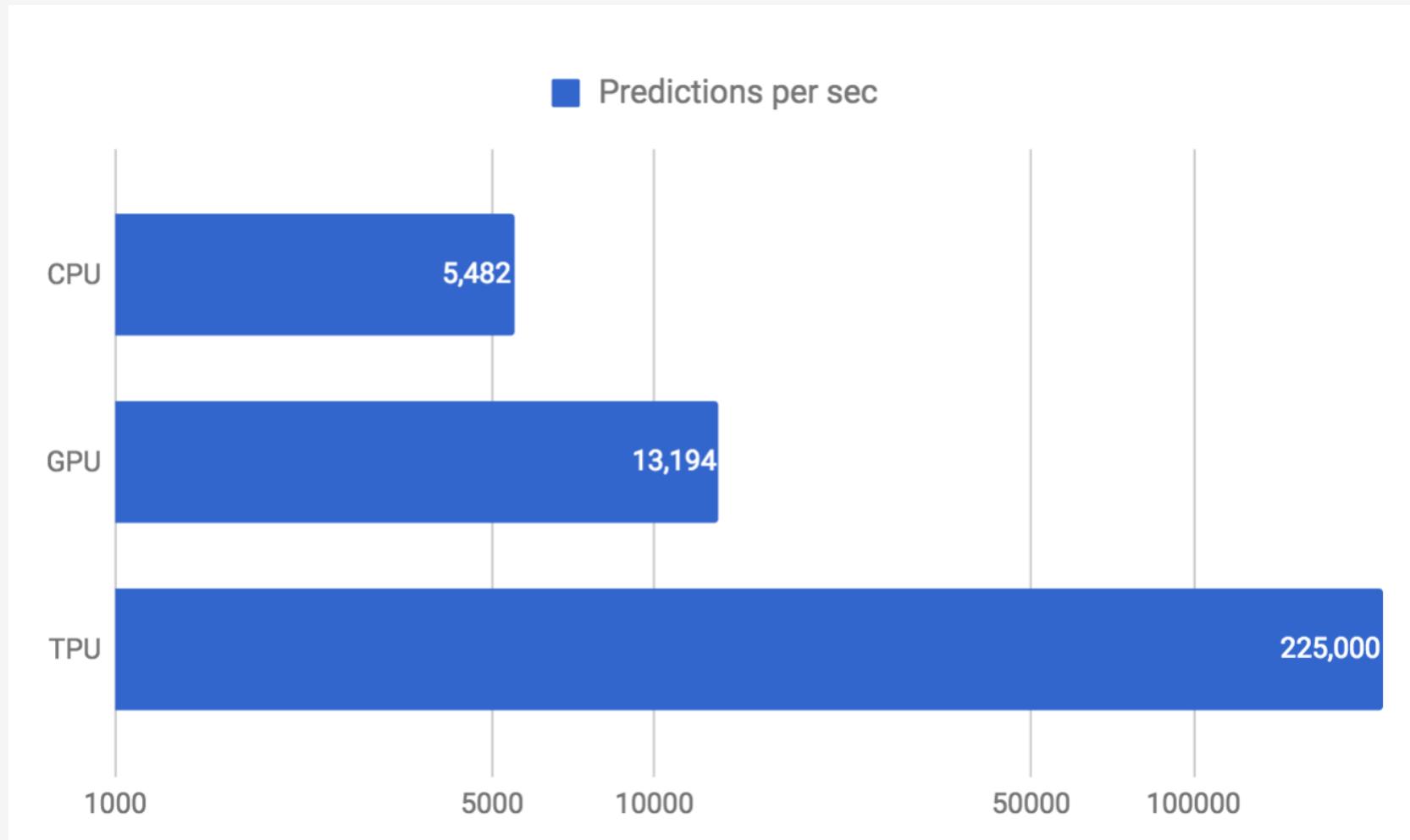
...



# Is it new ?

Not really ... but we can finally exploit it!

Significant computational power  
TPU ... always faster



# Is it new ?

---

Not really ... but we can finally exploit it!

Explosion of DataSets



**1627**  
Hosted  
Datasets

**276**  
Commercial  
Competitions

**1919**  
Student  
Competitions

**1M**  
Data  
Scientist

**4M**  
ML models  
submitted

# Is it new ?

---

Not really ... but we can finally exploit it!

Active and implicated community of deep learners  
companies, researchers, scientists ...

large choice of available tools !





A black and white photograph of a Parisian skyline, featuring the iconic Eiffel Tower rising above the city's rooftops. The foreground shows the architectural details of several buildings, including a prominent church tower on the left.

# 2

## Challenges motivating Deep Learning: ML vs DL

# Reminders



# Reminders

---

## ML reminders/basics

- Remember DL is a subset of ML
- To understand DL, strong ML basics are necessary:
  - complexity, overfitting and underfitting
  - model selection & model generalization
  - regularization
  - bias/variance tradeoff
  - gradient-descent optimization
- Let's quickly review these key concepts

# Reminders: Complexity, overfitting and underfitting

---

- Central challenge in ML: perform well on *new unseen* inputs (i.e. **generalization**)
- In optimization  $\Rightarrow$  Minimize the **training error**
- In ML  $\Rightarrow$  Minimize **test/generalization error** as well
- In other terms, the goals are to get:
  1. smallest training error
  2. smallest gap between training and test error
- Let's define  $L(y, f(x))$  and  $EPE \equiv E[L(y, f(x))|x = x_0]$

# Reminders: Complexity, overfitting and underfitting

---

- Let's take the example of  $MSE$ :  $L(y, f(x)) = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$
- Training error** can be expressed as:

$$\frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$$

- Test error** can be expressed as:

$$EPE = \sigma^2 + [Bias^2(\hat{f}(x_0)) + Var_{\tau}(\hat{f}(x_0))]$$

- How to assess these error terms ?

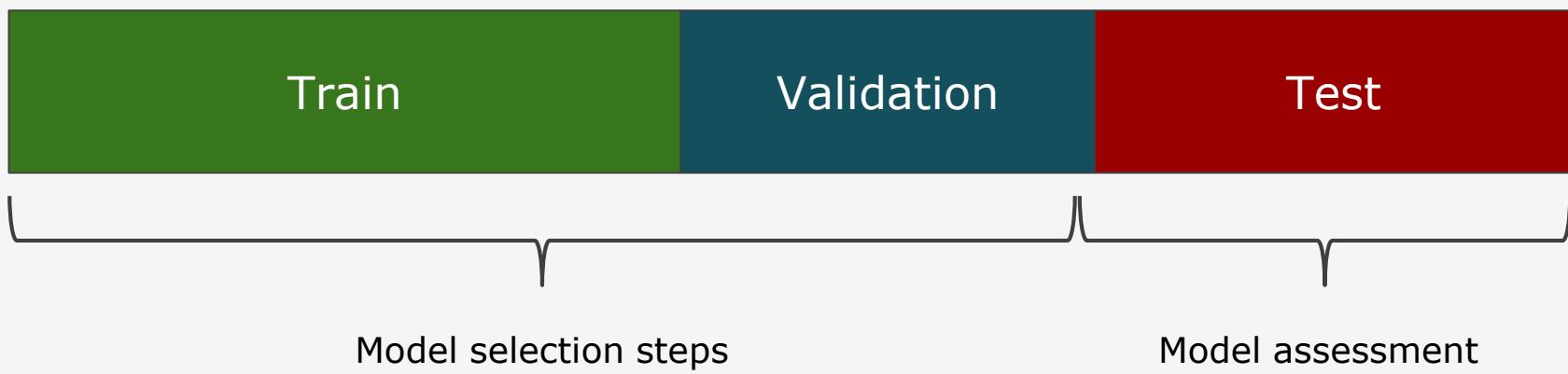
# Reminders: Complexity, overfitting and underfitting

In rich data context

1 - Train all models  
on the train set

2 - Estimate the prediction  
error of all models

3 - Estimate the  
generalization error  
of the final model



$$\text{training error} = \frac{1}{N} \sum_i^N (y_i - \hat{y}_i)^2$$

$$EPE = \sigma^2 + [Bias^2(\hat{f}(x_0)) + Var_{\tau}(\hat{f}(x_0))]$$

# Reminders: Complexity, overfitting and underfitting

## K-folds cross-validation



# Reminders: Complexity, overfitting and underfitting

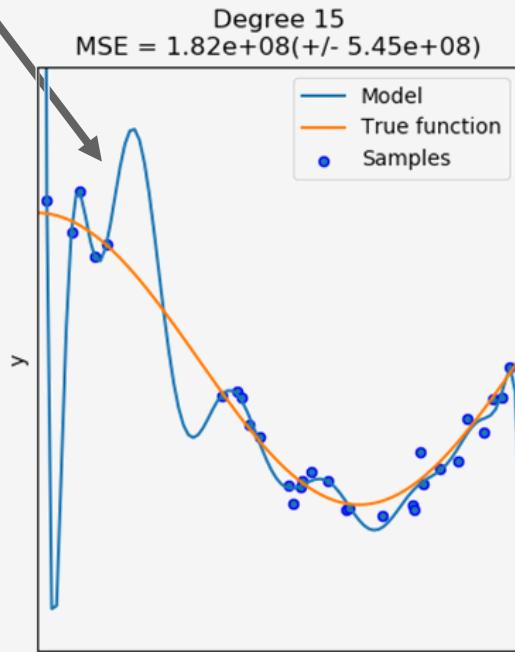
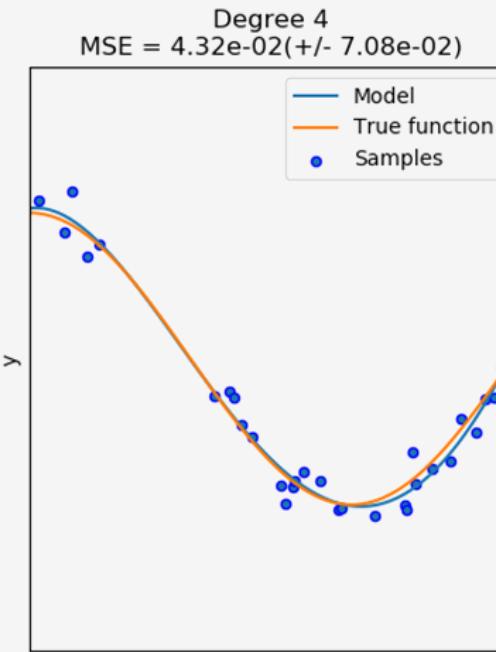
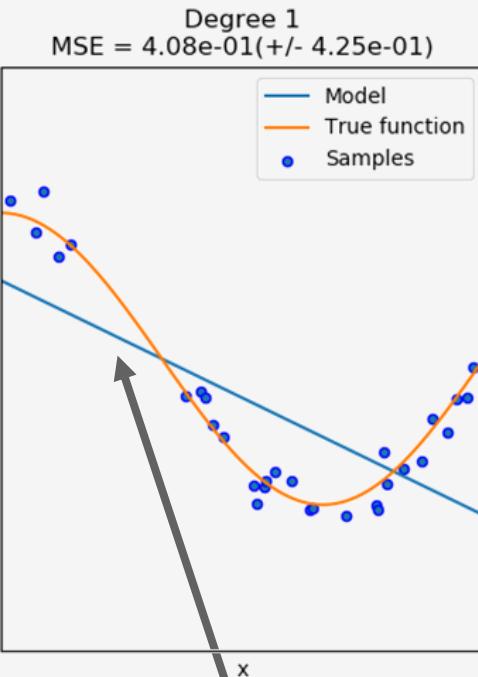
---

- **Underfitting:** "model unable to obtain a sufficiently low training error"
- **Overfitting:** "training error sufficiently low, but gap between training and test errors is too large"
- **Complexity:** model complexity (i.e. capacity)

# Reminders: Complexity, overfitting and underfitting

Illustration: fitting a polynomial regression

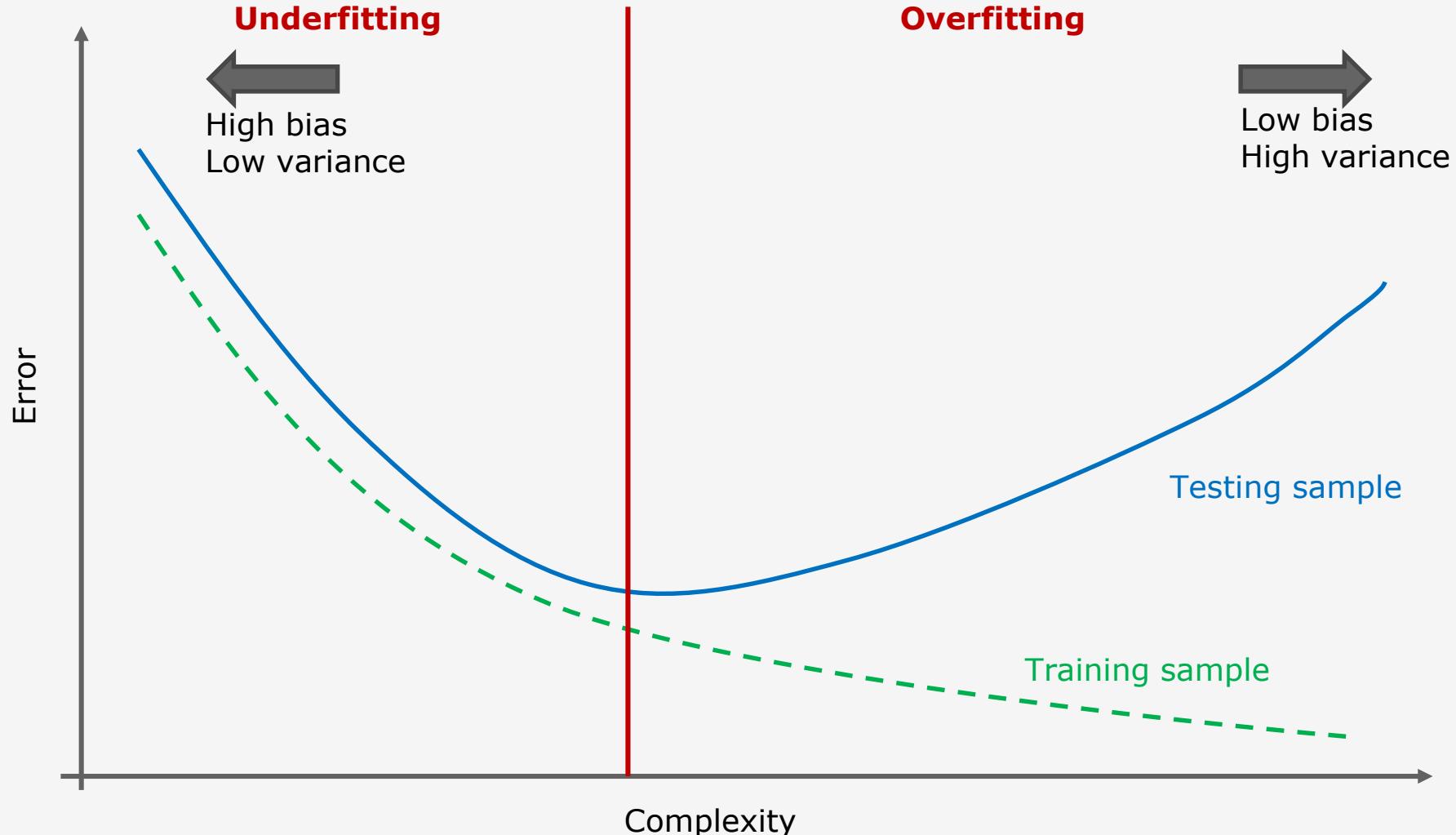
Overfitting



Underfitting

# Reminders: Complexity, overfitting and underfitting

Overfitting and underfitting are linked with complexity



# Reminders: Regularization

---

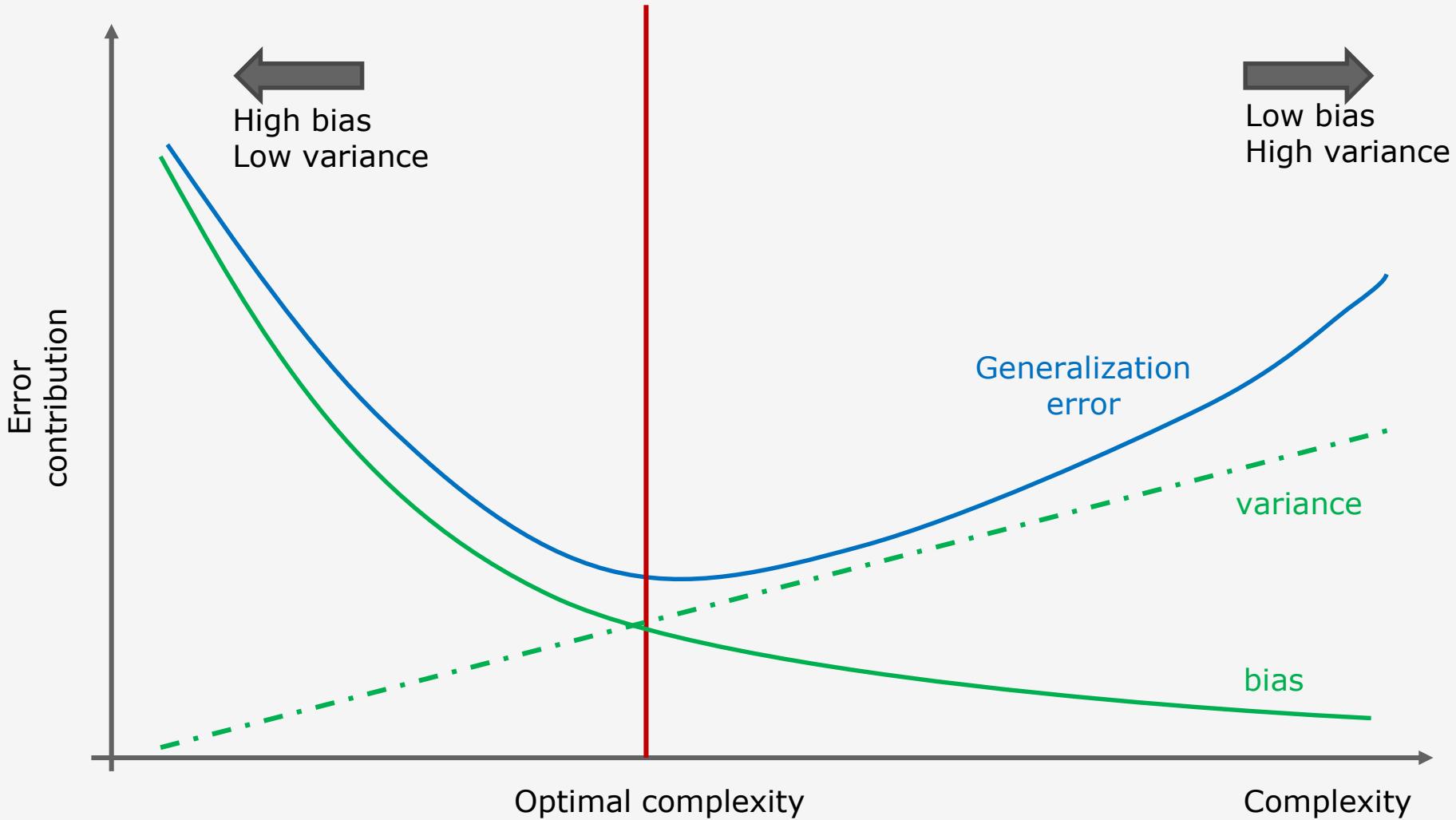
- To tackle overfitting and underfitting:
  - Increase or decrease the model capacity
  - Increase the training examples (more data)
  - Regularization
- **Regularization:** “any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error”
- Regularization to reduce overfitting

$$L(y, f(x)) = MSE_{train} + \lambda w^T w$$

- L1 or L2 regularization

# Reminders: bias/variance tradeoff

A question of 'arbitrage'



# Reminders: Gradient Descent optimization

---

## Basic Gradient Descent

- Gradient-based optimization is very important in Deep Learning
- Gradient Descent to find *Global Minimum* of a given function with parameters
- Let's consider a cost/loss function

$L(\theta)$  with input parameters  $\theta \in \Theta$

- Such as

$$\min L(\theta)$$

- Gradient Descent:

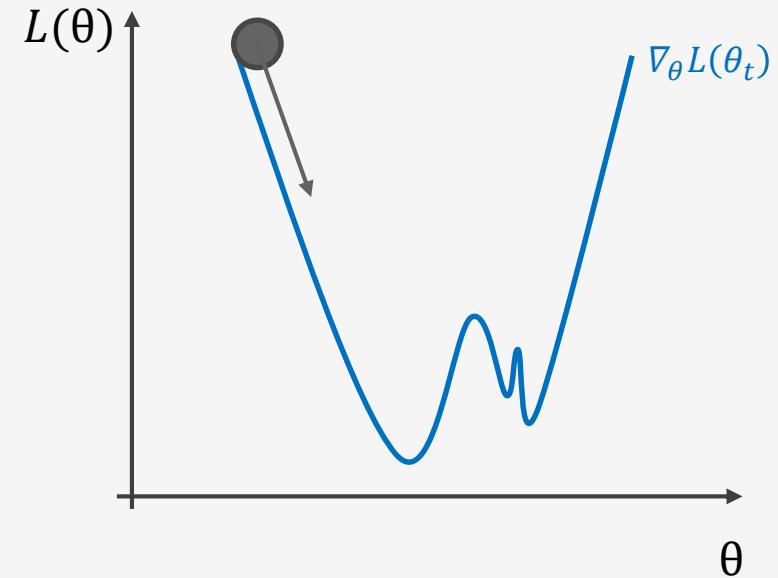
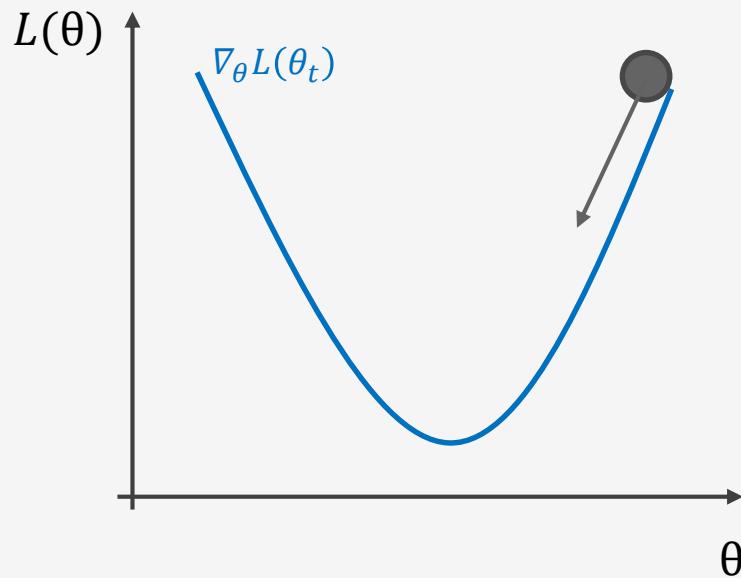
$$\theta_{t+1} = \theta_t - \lambda \nabla_{\theta} L(\theta_t)$$

with  $\nabla_{\theta} L(\theta_t)$  as the gradient of  $L(\theta)$  w.r.t. to  $\theta$   
and  $\lambda$  as the learning rate

# Reminders: Gradient Descent optimization

## Basic Gradient Descent

See it as a ball rolling down ...

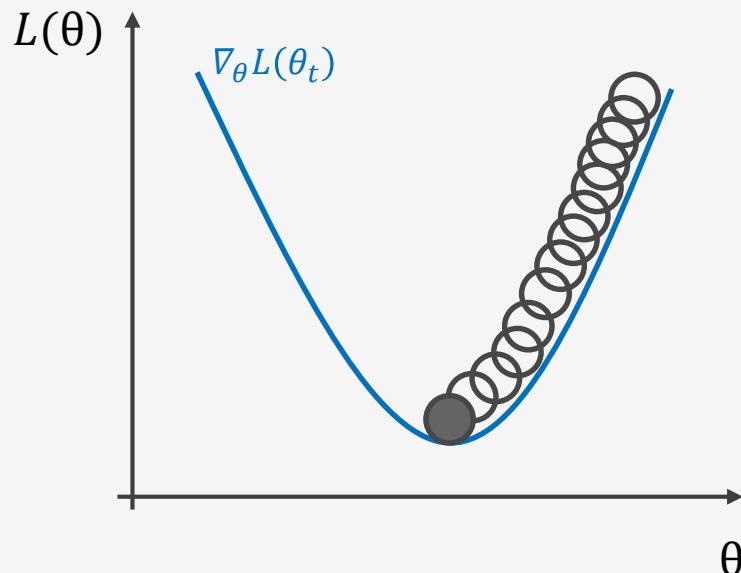


# Reminders: Gradient Descent optimization

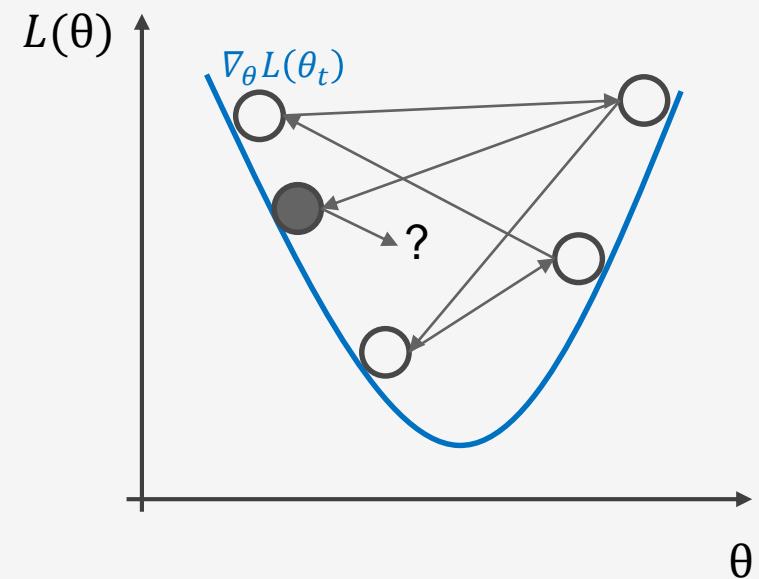
## Basic Gradient Descent

Choice of the learning rate  $\lambda$

$\lambda$  too **small** .... Descent too **slow**



$\lambda$  too **large** .... Target **missed**

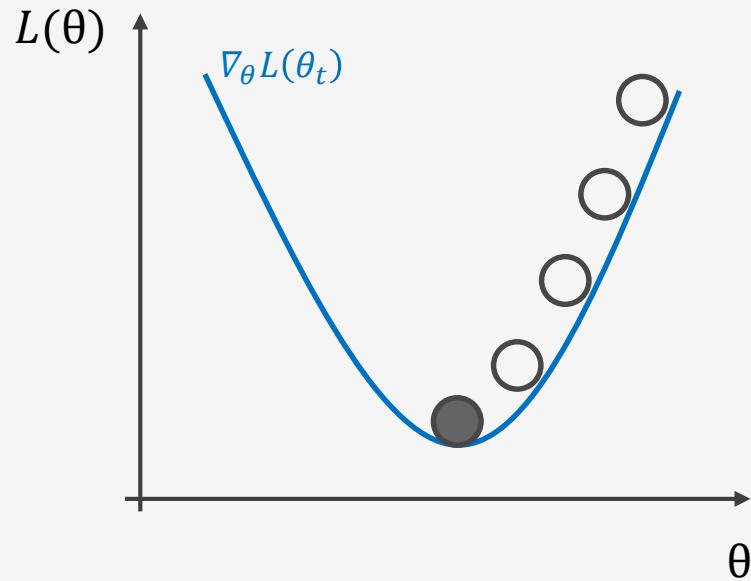


# Reminders: Gradient Descent optimization

## Basic Gradient Descent

Choice of the learning rate  $\lambda$

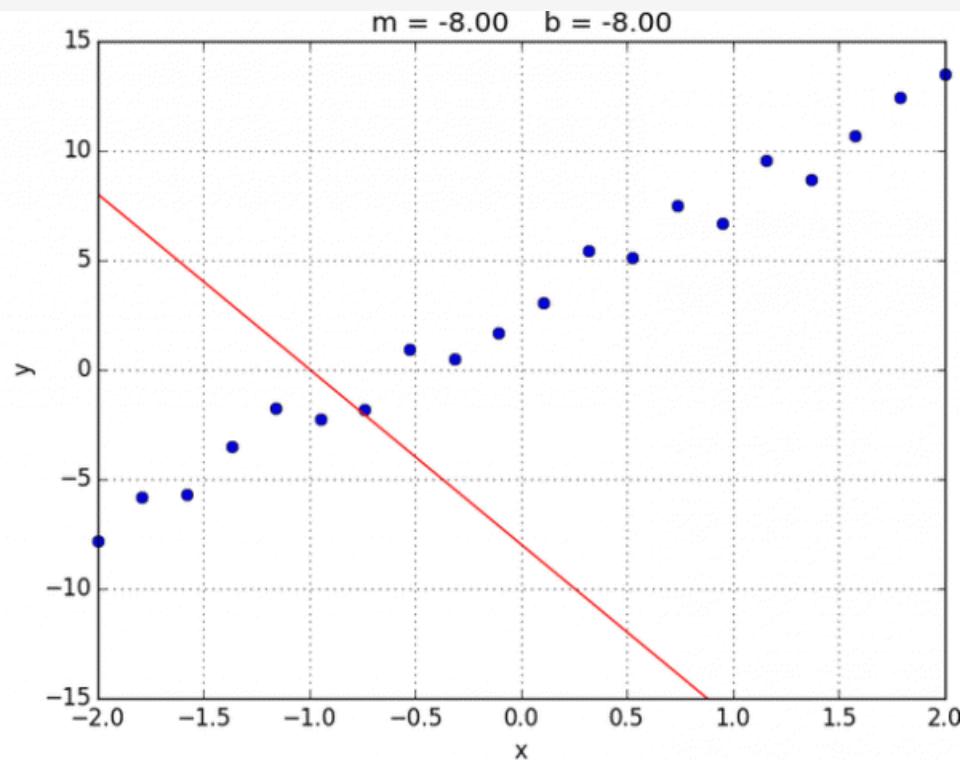
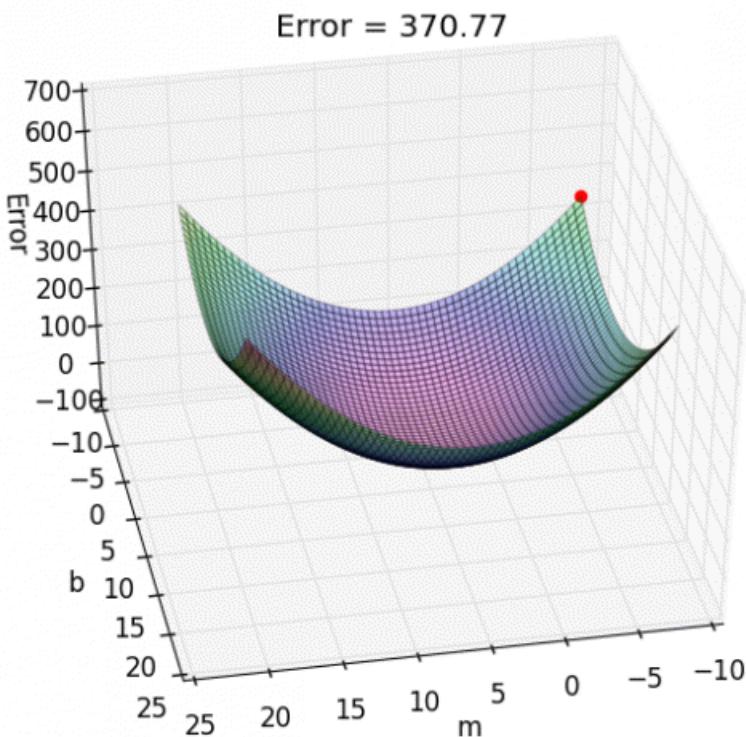
Tradeoff between  
*speed and precision*



# Reminders: Gradient Descent optimization

## Illustration of Gradient Descent

- Let's consider a linear regression with  $f(x) = m * x + b$
- With  $m, b \equiv \theta$  and  $L(\theta) = MSE$



# The Challenges motivating DL



# Challenges motivating Deep Learning

---

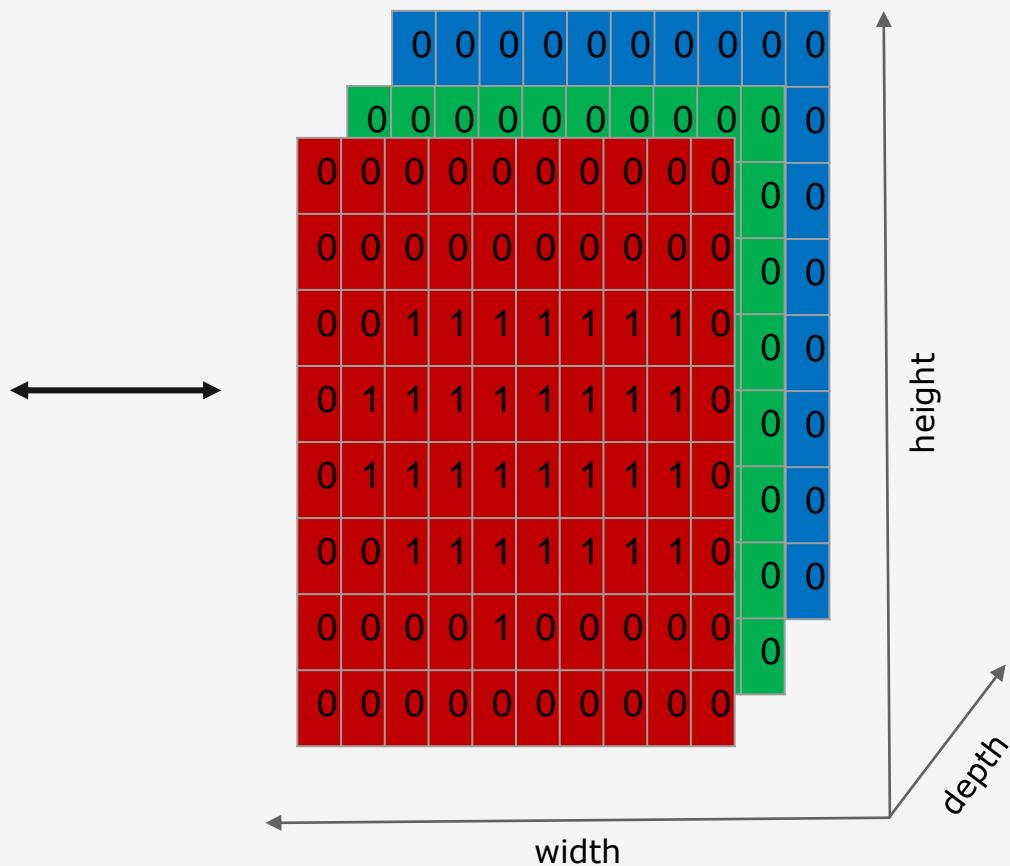
- ML works well in wide variety of problems
- Unfortunately, they haven't succeed in solving core AI problems (e.g. image recognition, speech recognition, etc ...)

But Why ?

# Challenges motivating Deep Learning

## Failure of ML: example of image recognition

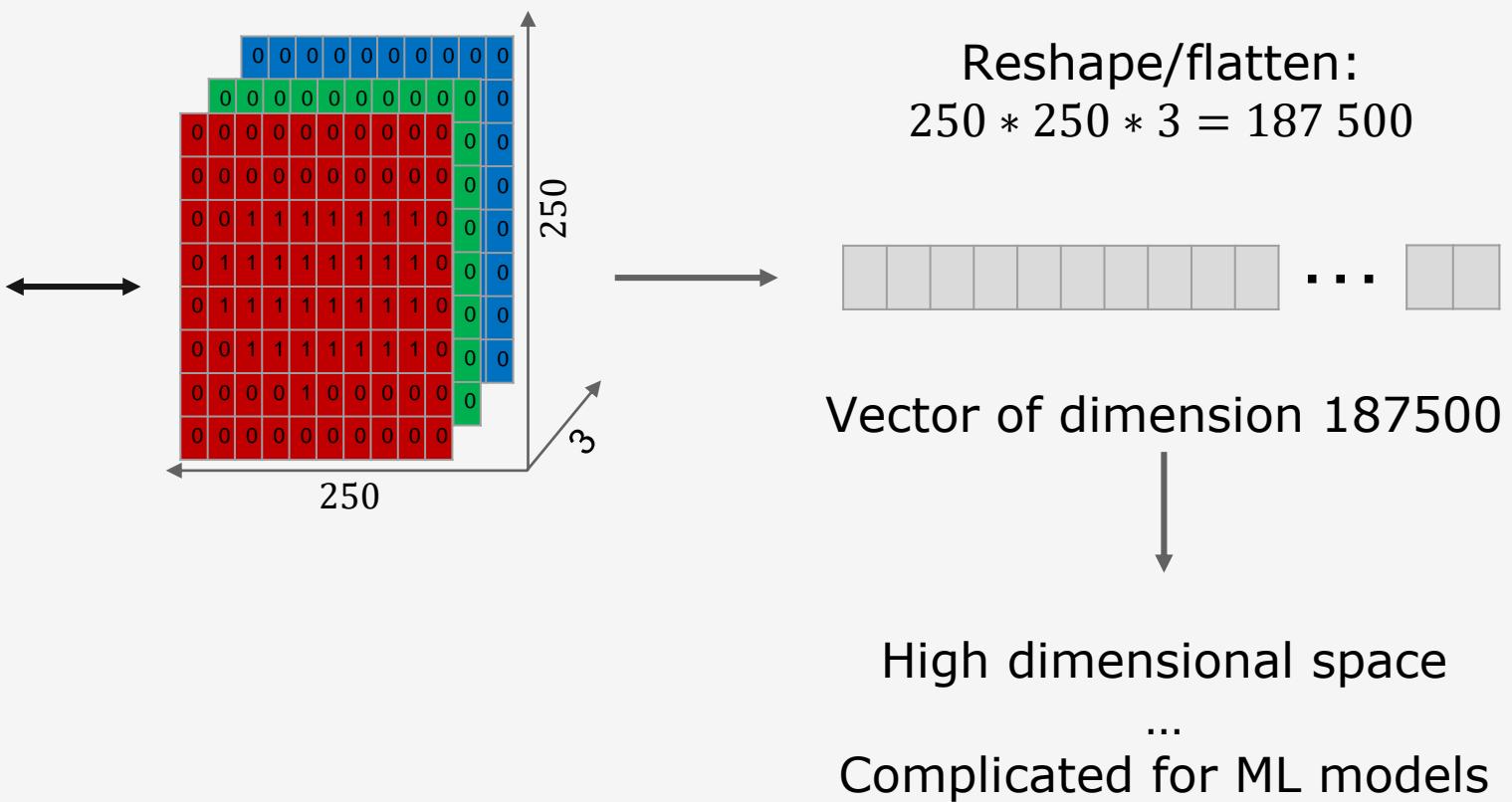
Image = RGB matrix



# Challenges motivating Deep Learning

Failure of ML: example of image recognition

With classical ML model



# Challenges motivating Deep Learning

---

## Challenge 1: The curse of dimensionality

- Many ML problems' complexity increase with the number of dimensions in the data
- The number of possible set of configurations increases exponentially with the number of variables
- When can we qualify the issue of high-dimensional ?

*number of variables( $x$ )  $\gg$  number of training examples ( $n$ )*

# Challenges motivating Deep Learning

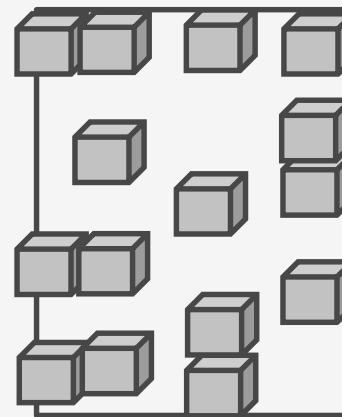
## Challenge 1: The curse of dimensionality

1 dim



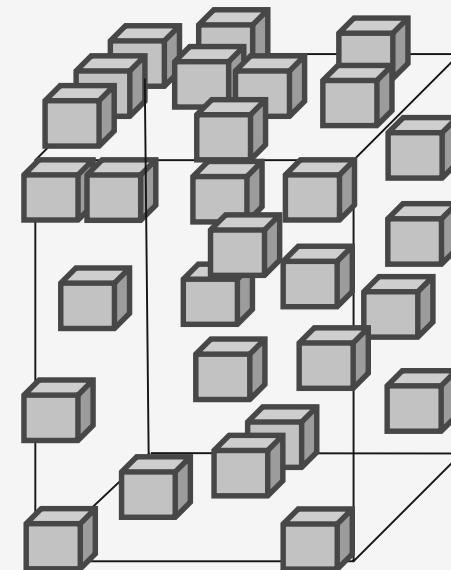
4  
regions of  
interest

2 dims



$4 * 4 = 16$   
regions of  
interest

3 dims



$4 * 4 * 4 = 64$   
regions of  
interest

# Challenges motivating Deep Learning

---

## Challenge 1: The curse of dimensionality

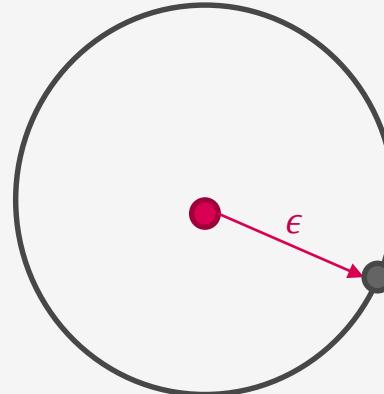
- Traditional ML approach insufficient for these spaces
- **DL challenge:** generalize well in high-dimensional spaces while keeping the computational cost low

# Challenges motivating Deep Learning

## Challenge 2: Local constancy and smoothness regularization

- ML algorithms must be guided by *prior beliefs* about the kind of function to learn
- **Local constancy prior** : “the function we learn should not change very much within a small region”:

$$f^*(x) \approx f^*(x + \epsilon)$$



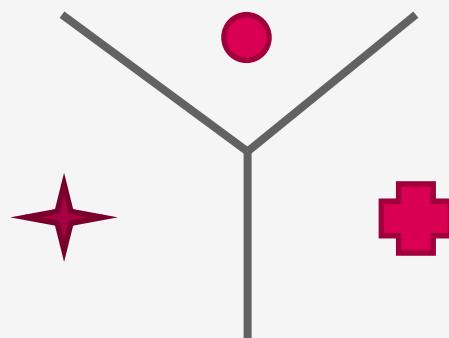
- Intuition: if we know a good answer for an input  $x$ , the answer is also good in the small neighborhood of  $x$

# Challenges motivating Deep Learning

## Challenge 2: Local constancy and smoothness regularization

- **Limitation:** “to distinguish  $O(k)$  regions in the input space, we need  $O(k)$  examples”
- *Nearest neighbors, local kernel and tree based* methods suffer from the limitations of this prior

Example of knn



# Challenges motivating Deep Learning

---

## Challenge 2: Local constancy and smoothness regularization

- Challenge
  - Can we represent a complicated function efficiently?
  - Can it generalize well to new inputs ?

- DL allows to represent

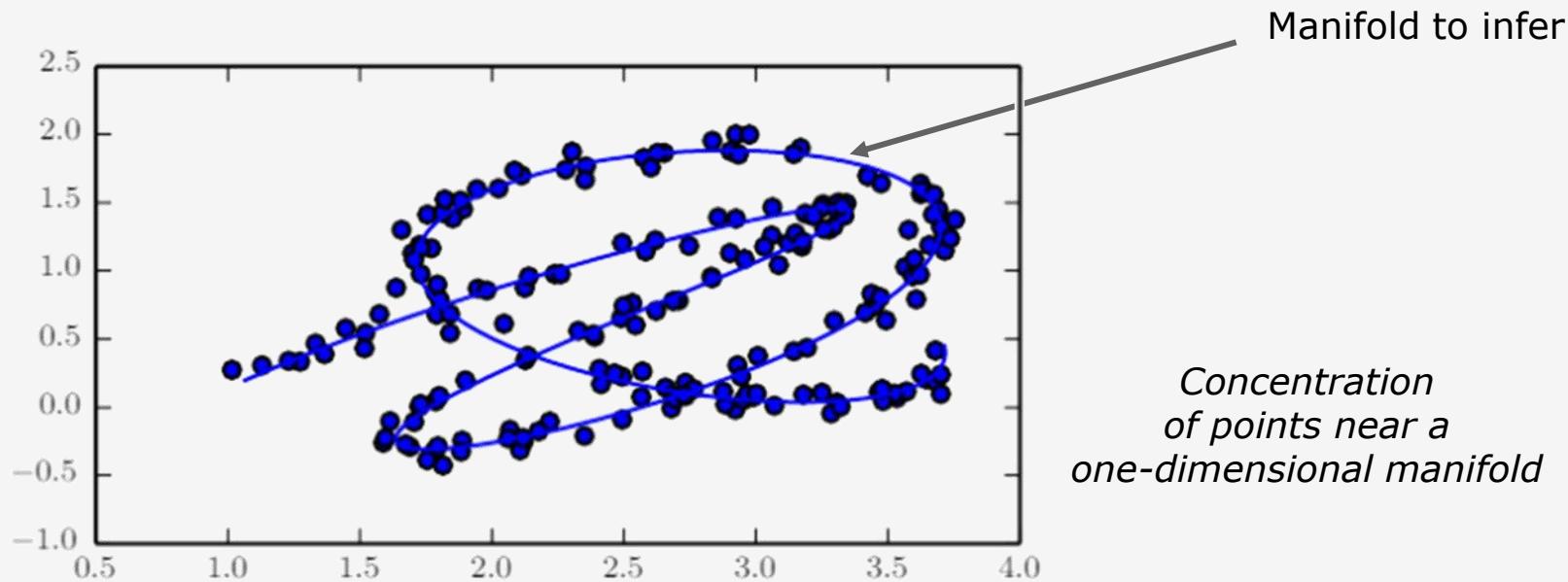
$$O(2^k) \text{ regions with } O(k) \text{ examples}$$

as it can introduce additional assumptions about the underlying data-generating distribution

# Challenges motivating Deep Learning

## Challenge 3: Manifold learning

- **Manifold** is a connected region: a set of points associated with a neighborhood around each point. It is an Euclidean space.
- It is possible to move from neighborhood to another one on the manifold



# Challenges motivating Deep Learning

---

## Challenge 3: Manifold learning

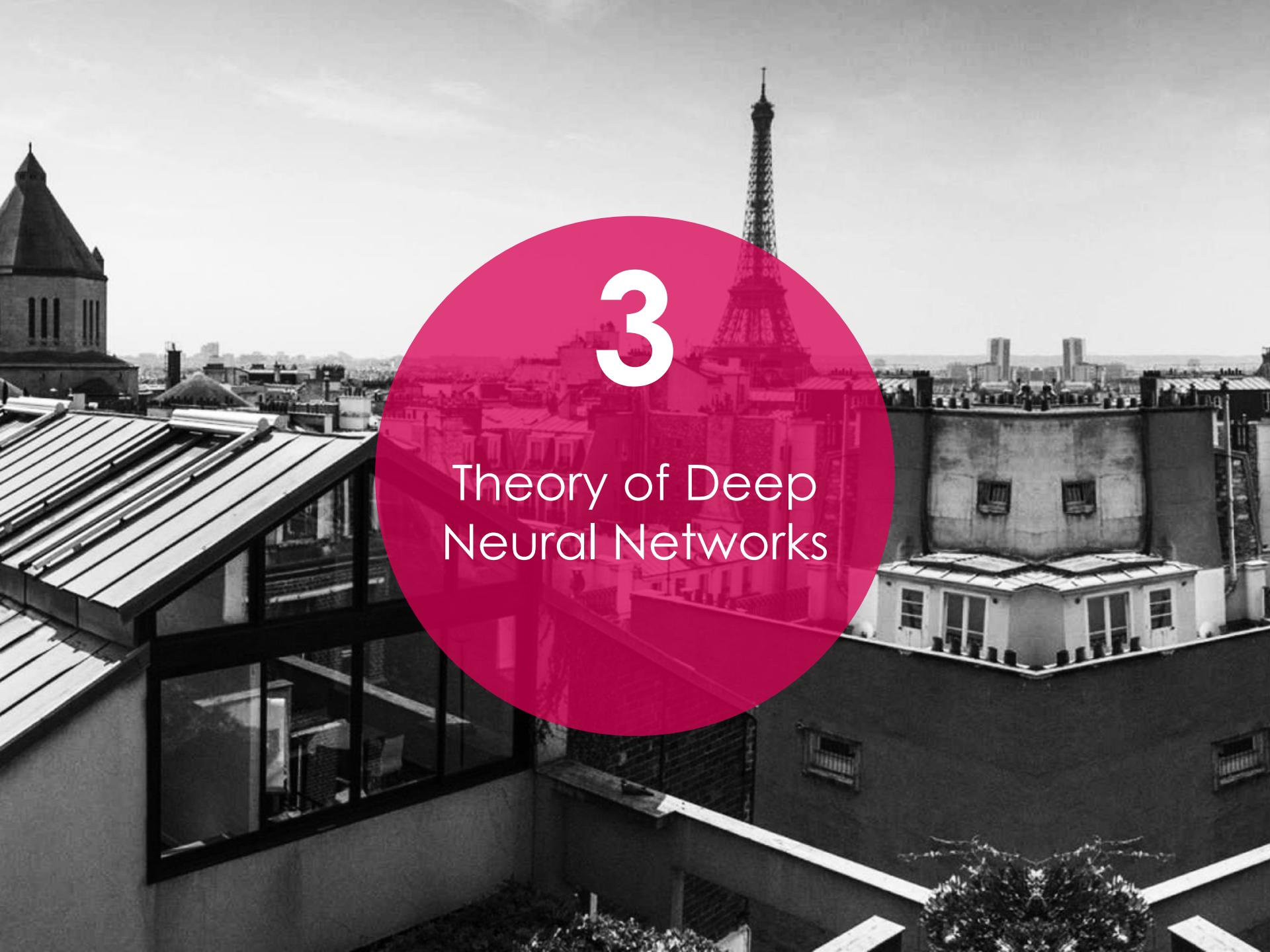
- ML seem hopeless if want to learn interesting variations across all  $R^n$
- Manifold learning algorithms consider that
  - $R^n$  inputs are invalid
  - Interesting variations occur only along a collection of manifolds
    - Along the directions of the manifold
    - When passing from one manifold to another
- Necessary assumption for processing images, audio or texts

# Challenges motivating Deep Learning

---

To summarize so far ...

- Deep learning is motivated by its ability to
  - Learn in high-dimensional spaces
  - Learn very complicated functions efficiently and still be able to generalize well
  - Allow manifold learning
- Solve core AI issues
- We've seen:
  - Basic reminders of ML
  - Reasons to interest in DL



3

## Theory of Deep Neural Networks

# Deep Forward Neural Networks



# Deep Forward Neural Networks

---

## Perceptron

- Simplest neuron possible
- Introduced by *Rosenblatt* in the late 50's
- « *A device that makes decisions by weighting up evidence* »
- Takes input  $X$ , transforms it and fires a binary output (0 or 1)

# Deep Forward Neural Networks

---

## Perceptron

What does it mean in maths ?

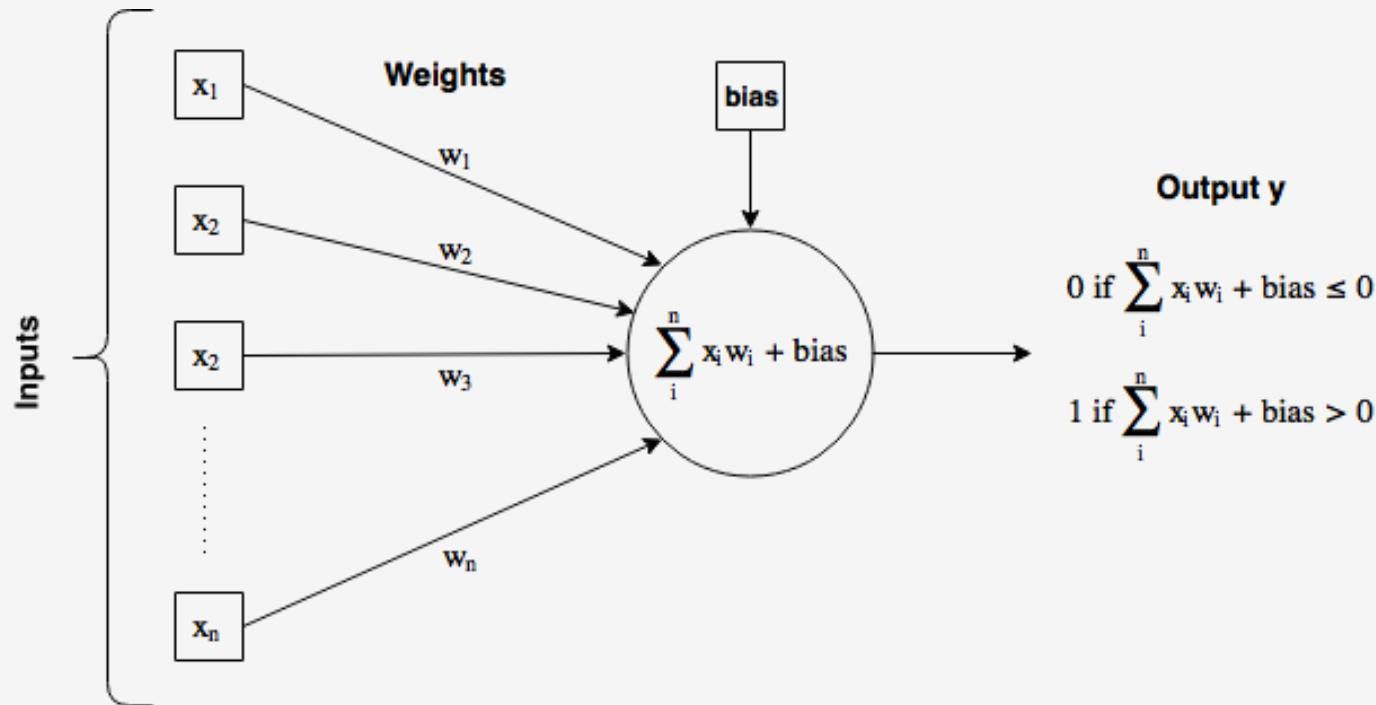
- Let's consider:
  - Inputs  $X(x_1, x_2, x_3, \dots, x_n)$
  - Binary output  $f(x)$
  - Set of weights  $W(w_1, w_2, w_3, \dots, w_n)$
  - Bias  $b$
- The perceptron can be expressed:

$$f(x) = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i w_i + b \leq 0 \\ 1 & \text{if } \sum_{i=1}^n x_i w_i + b > 0 \end{cases}$$

# Deep Forward Neural Networks

## Perceptron

Flow of information in a perceptron with weights W and bias b



# Deep Forward Neural Networks

---

## Perceptron

- A perceptron fires only binary outputs
- How can we get continuous outputs ?

# Deep Forward Neural Networks

---

## Neuron with activation

- **Activation** term  $\sigma$
- such as

$$f(x) = \sigma \left( \sum_{i=1}^n x_i w_i + b \right)$$

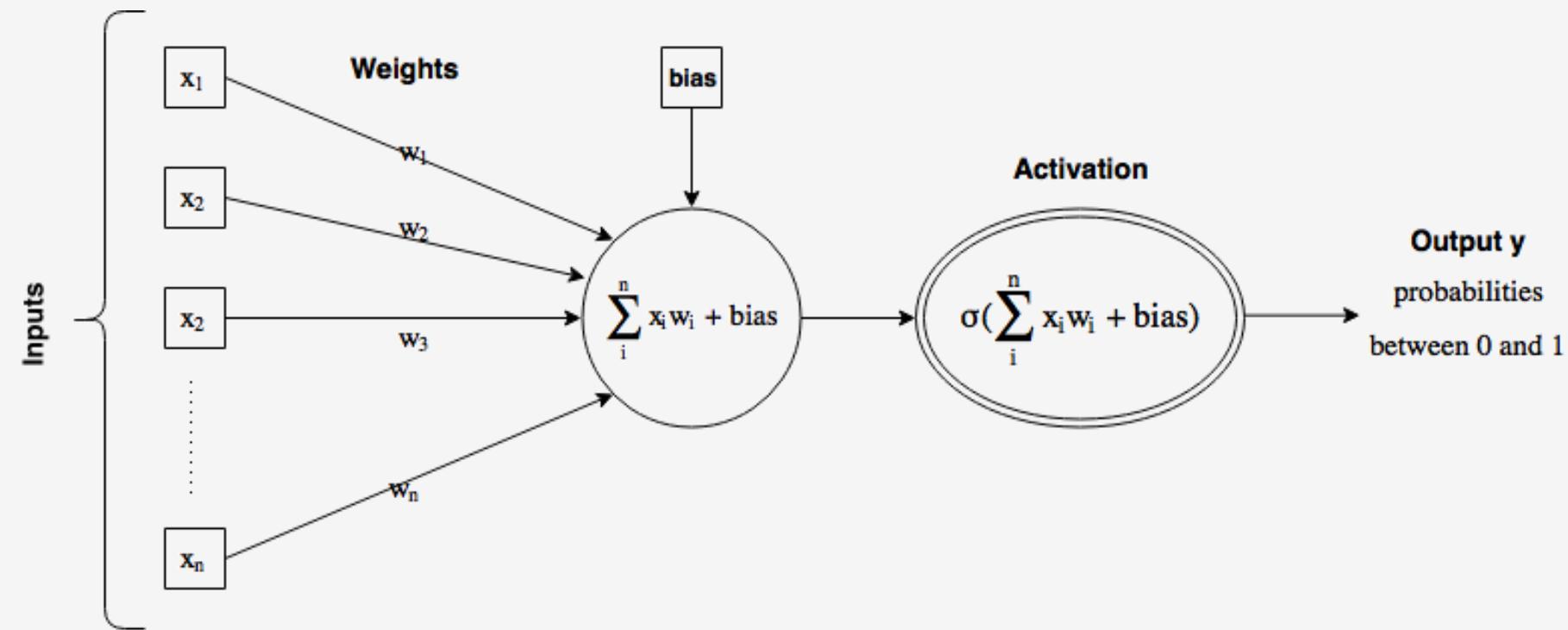
- Sigmoid activation function (i.e. logit)  $\rightarrow$  famous

$$\sigma \left( \sum_{i=1}^n x_i w_i + b \right) = \frac{1}{1 + e^{-\sum_{i=1}^n x_i w_i + b}}$$

# Deep Forward Neural Networks

## Neuron with activation

Flow of information in a neuron with activation



# Deep Forward Neural Networks

---

Neuron with activation

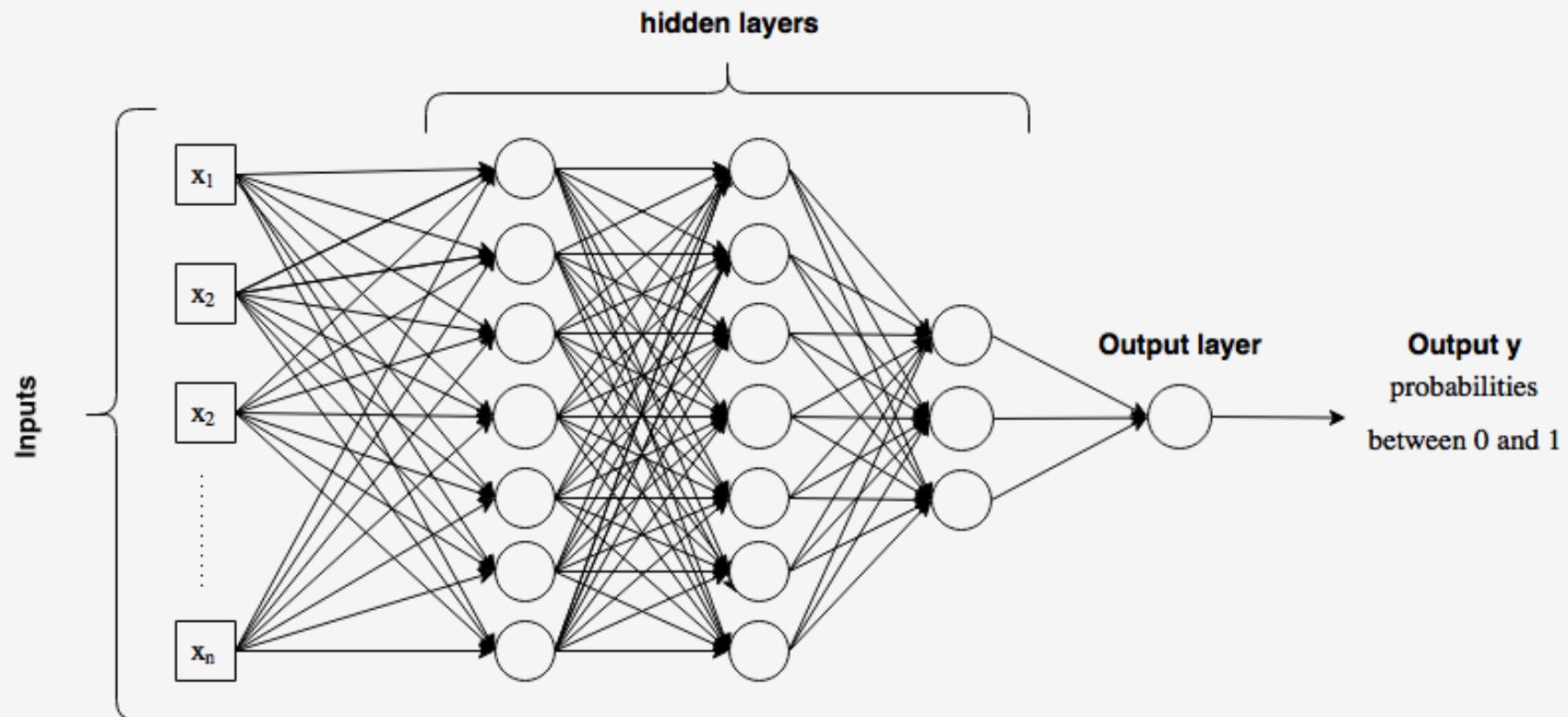
A single neuron might not be the strongest model of decision making

...

However, a set of neurons constituting together a sufficiently complex network could make sophisticated decisions

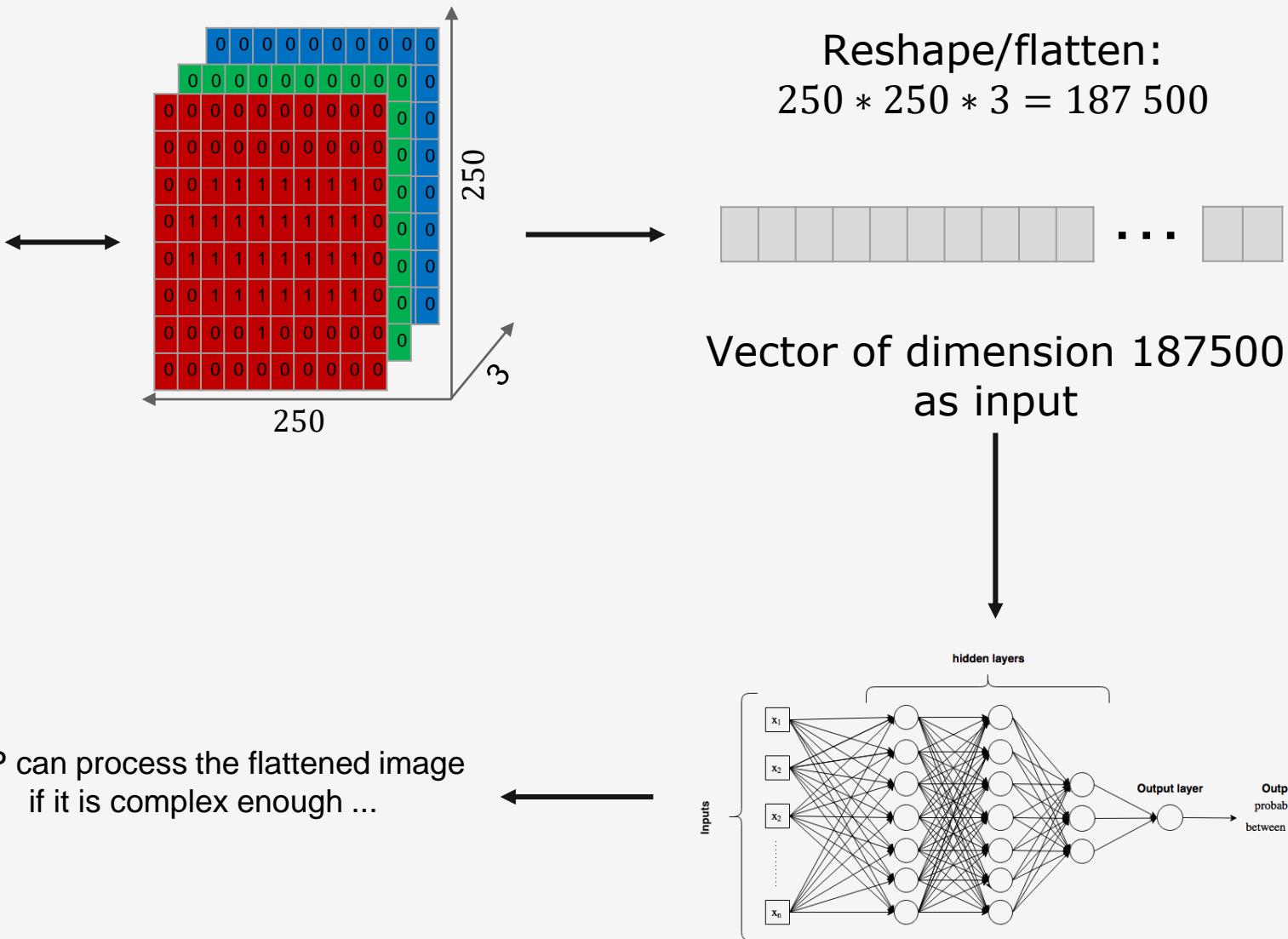
# Deep Forward Neural Networks

## Multilayer Perceptron



# Deep Forward Neural Networks

## Multilayer Perceptron



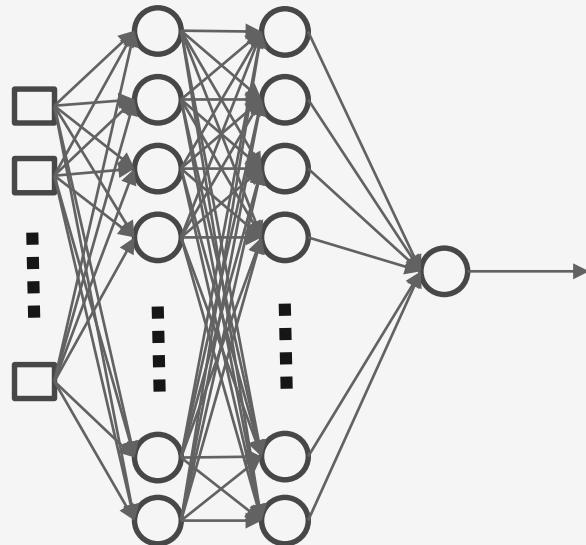
# Deep Forward Neural Networks

Multilayer Perceptron - design

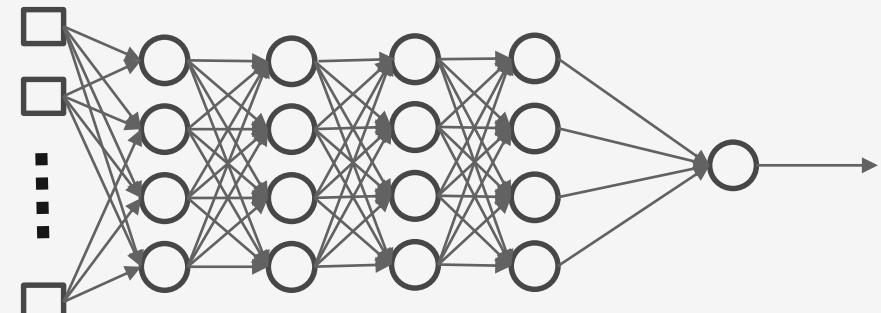
If it is complex enough ?!

...  
How to make it 'more' complex ?

More neurons  
per layer



More layers



# Deep Forward Neural Networks

Multilayer Perceptron – complexity, overfitting and underfitting

Be careful !!

If **too complex**

...

if too many layers,  
too many neurons



Risk of **overfitting**

If **not sufficiently complex**

...

too few layers,  
too few neurons



Risk of **underfitting**

# Deep Forward Neural Networks

---

## Universal approximation theorem

Feedforward networks (such as MLP) allow the learning of highly non-linear function.

"According to the **universal approximation theorem** (Hornik et al. (1989)) any feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) **can approximate any Borel measurable function** from one finite-dimensional space to another with any desired non-zero amount of error, **provided that the network is given enough hidden units**"

# Deep Forward Neural Networks

---

## Universal approximation theorem

- The learning process is critical as it can fail because:
  - Underfitting → unable to find the optimal parameters
  - Overfitting → the algorithm chooses the wrong function
- Can lead to bad accuracy and bad generalization

# Deep Forward Neural Networks

---

## Limitations of Multilayer Perceptron

The overparametrized model has 3 main inconvenient:

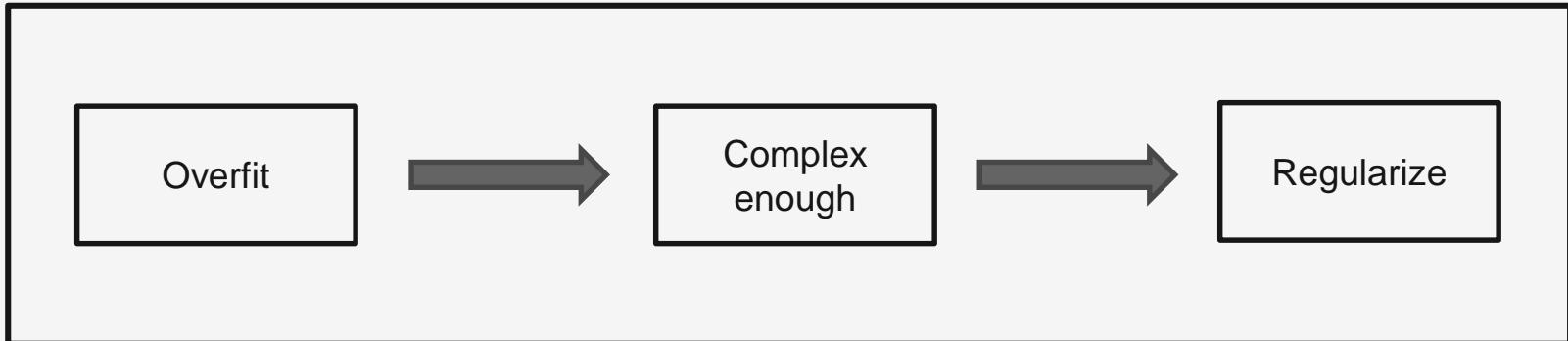
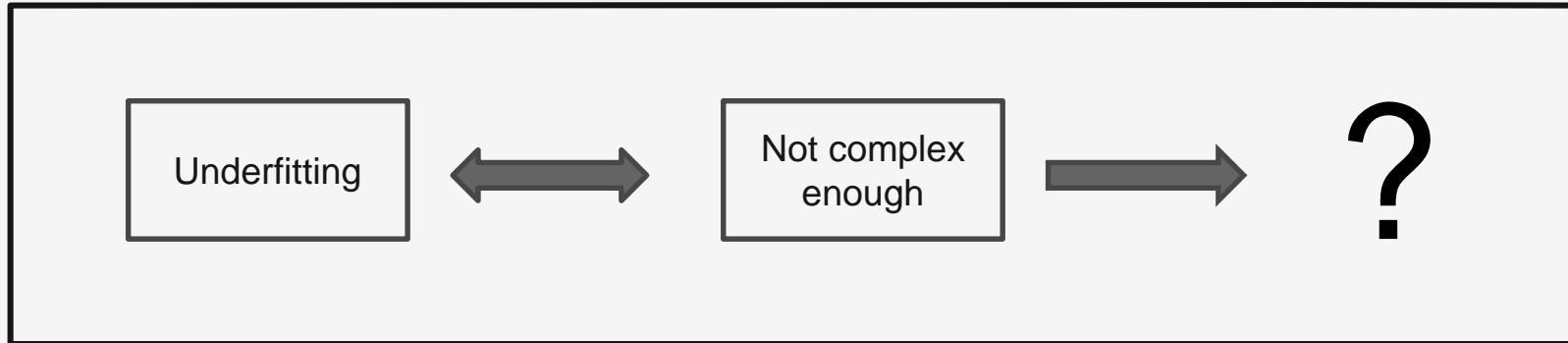
- **Non-convex and unstable loss function**  
more than one local minima possible – random weight initialization can lead to different accuracy
- **Overfitting**
- **Sensitivity to feature scaling**  
between 0 and 1

# Regularization for DL



# Regularization for DL

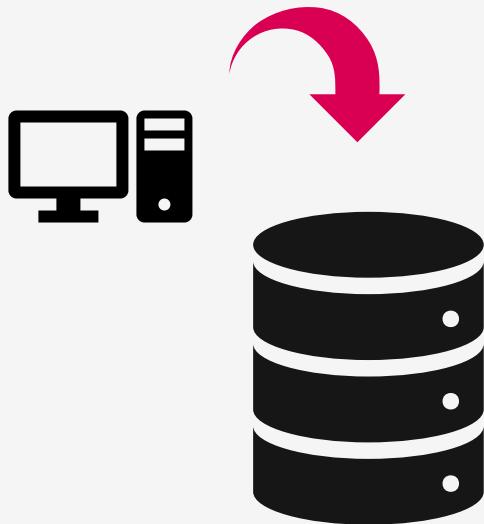
Should we first look to overfit ?



# Regularization for DL

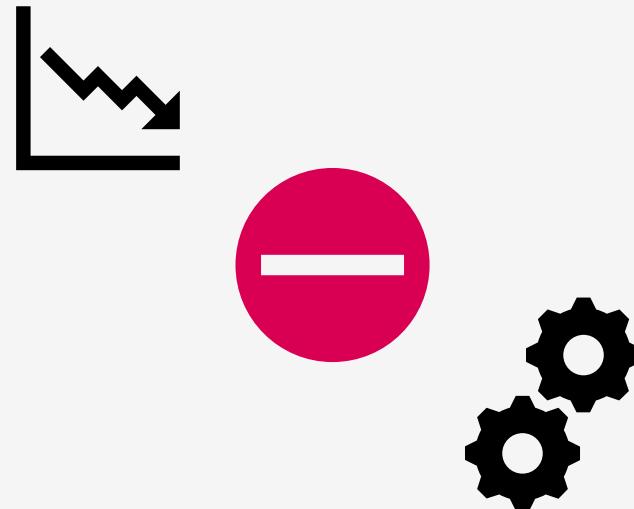
First easy options

Add more data



Not always possible ...

Stop the process when validation accuracy drops or stagnates



# Regularization for DL

## Data Augmentation

Blow artificially your training set !



Random  
transformations



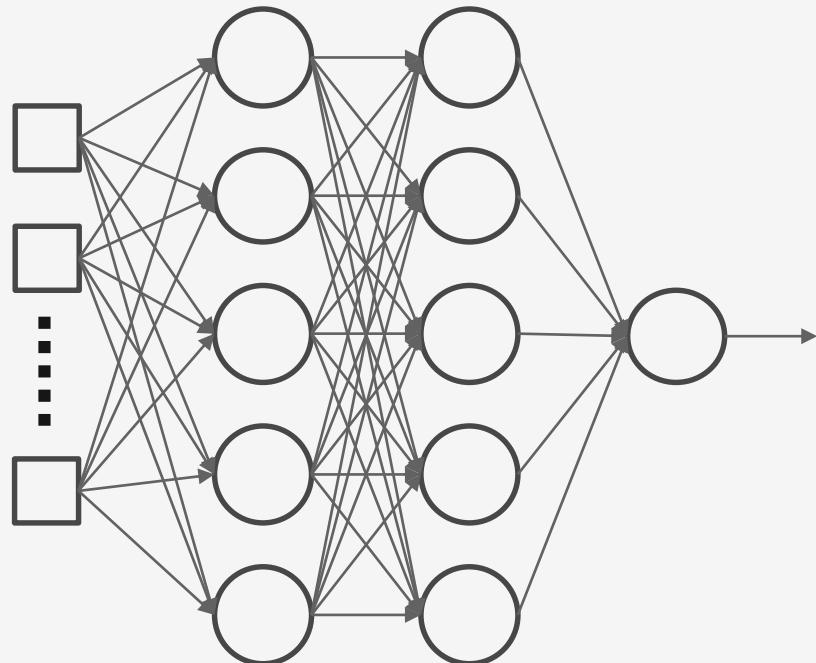
New set of images



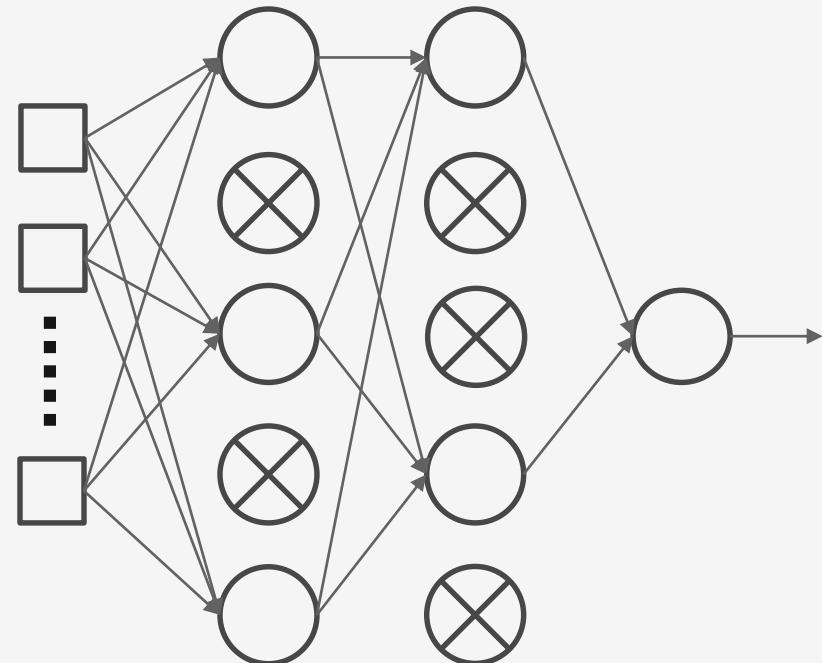
# Regularization for DL

## Dropout

Overfitting model



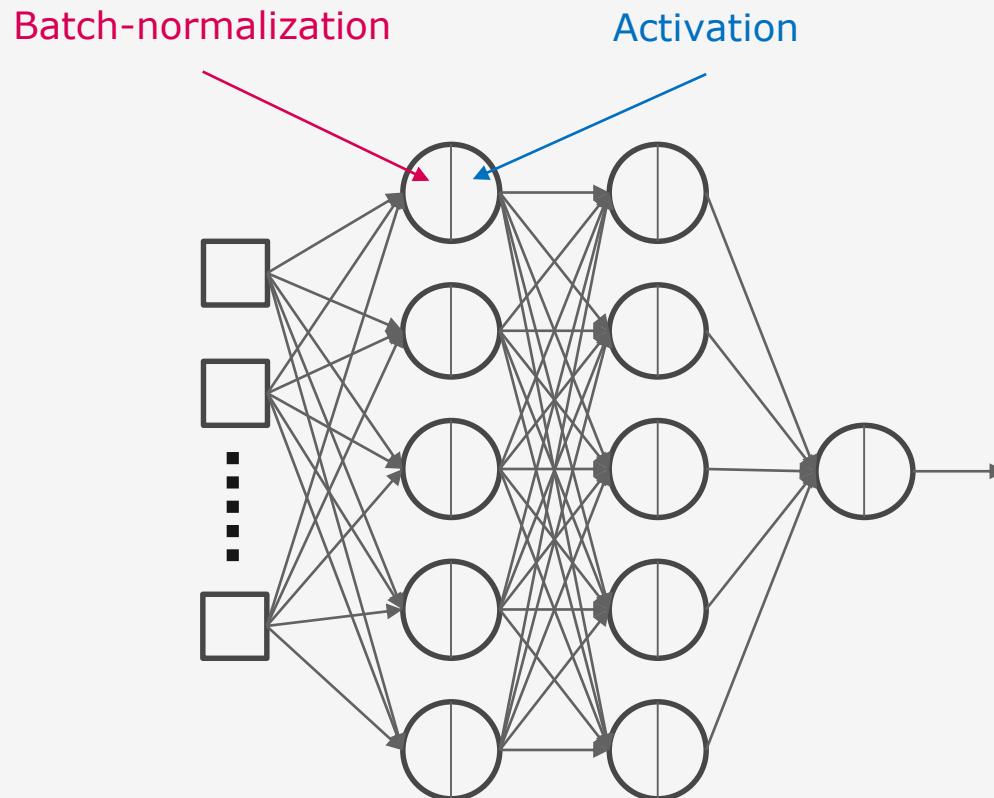
Dropout regularized model



It avoids units  
too co-adapt too  
strongly

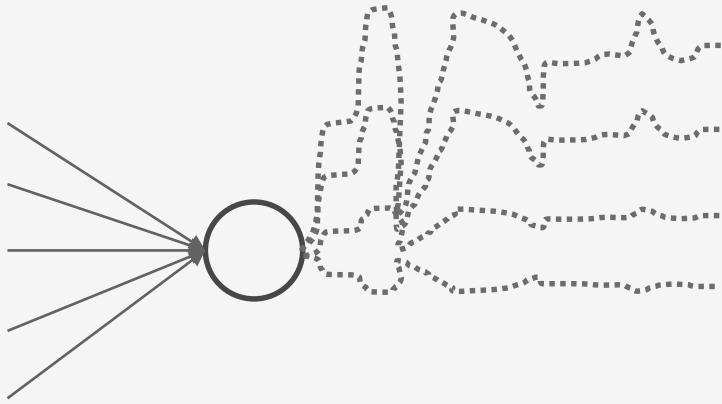
# Regularization for DL

## Batch-Normalization



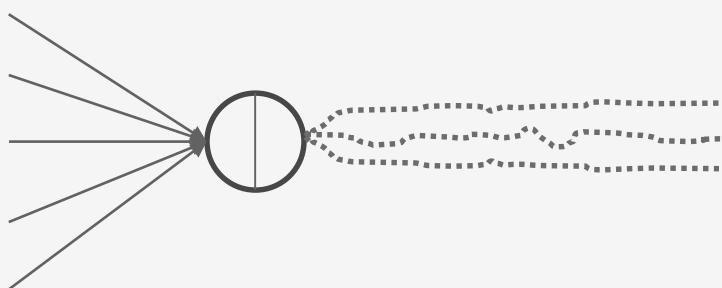
# Regularization for DL

## Batch-Normalization



Normal case without BN

- Internal covariance
  - Layer's need to adapt to inputs' distribution
- Learning requires lower  $\lambda \rightarrow$  slower process



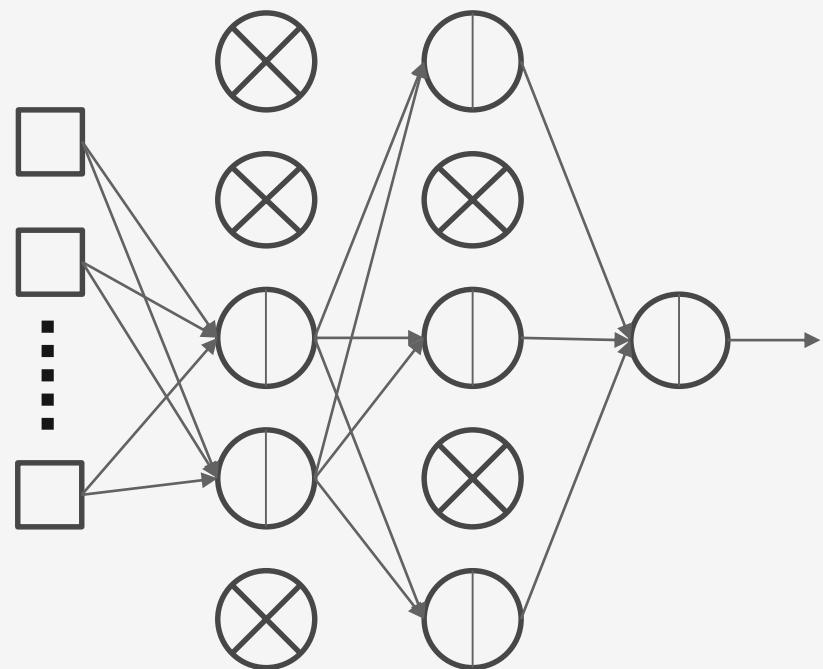
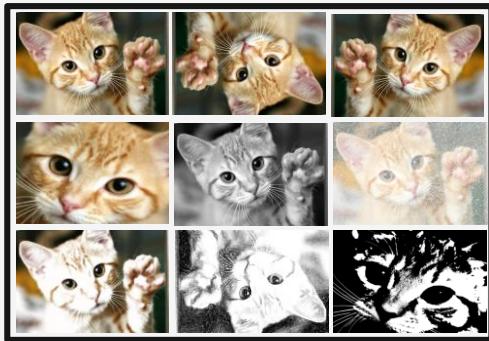
With BN

- Initially treats the internal covariance issue
- Allows higher  $\lambda \rightarrow$  faster learning process
- It has also proven to significantly reduce overfitting

# Regularization for DL

Combine all methods

Data Augmentation + BN + Dropout  
It should work !

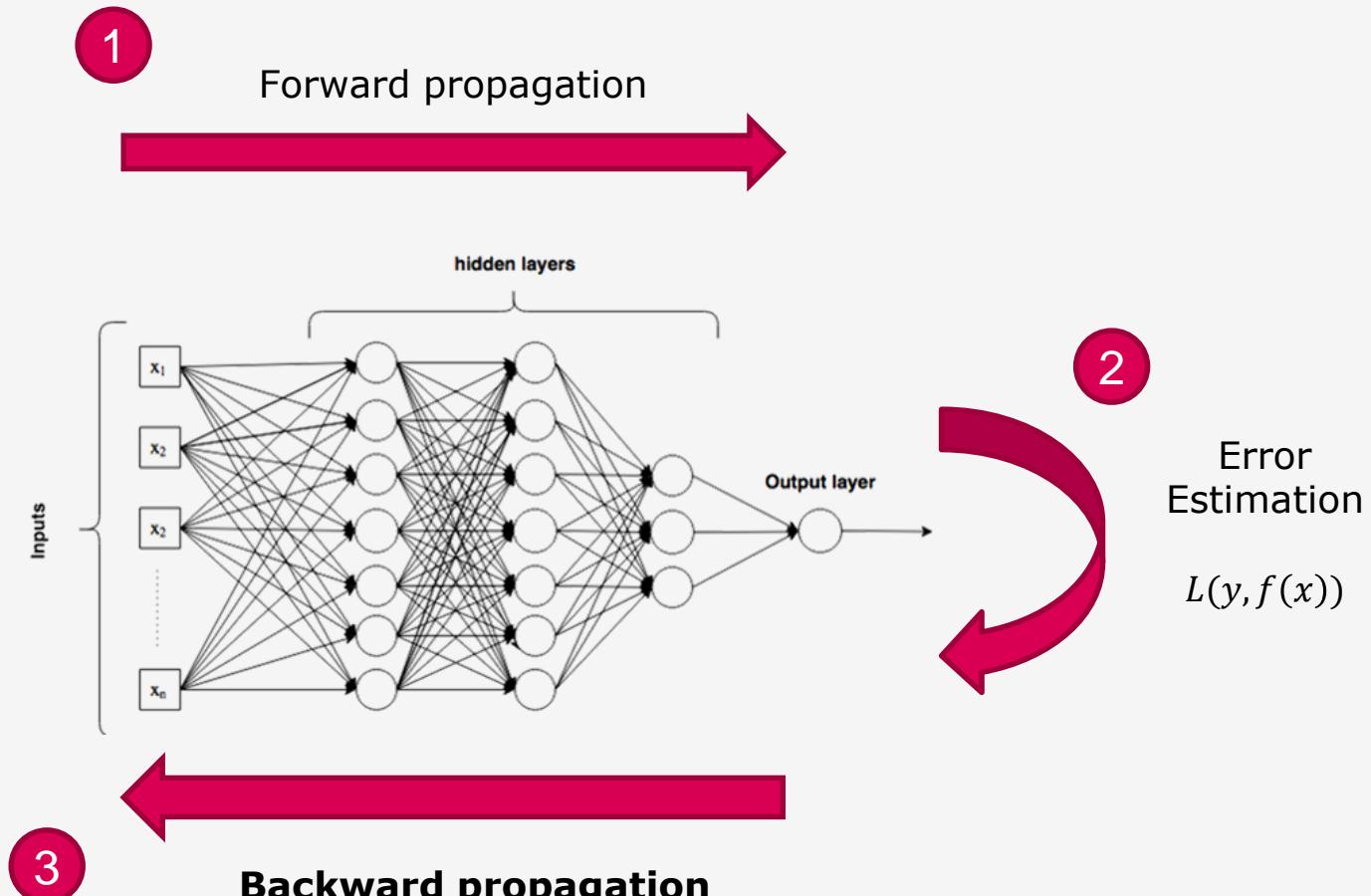


# Training Deep Neural Networks



# Training Deep Neural Networks

## Back-propagation



- **Numerical computation of the gradient**
- **Weight actualization**

# Training Deep Neural Networks

---

## Stochastic Gradient Descent

- Recurrent issue in ML: large training set is good for generalization
- Gradient Descent with large datasets is computationally heavy



**Stochastic Gradient Descent** (SGD) with *mini-batches*

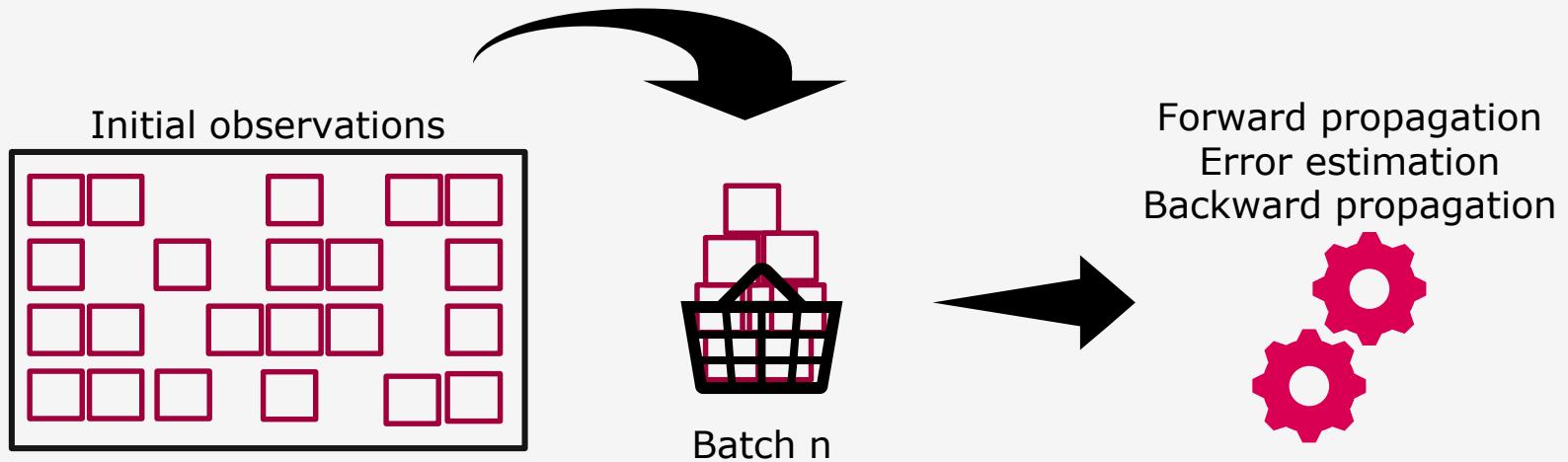
- Update of parameters for each mini-batch  $i$

$$\theta_{t+1} = \theta_t - \lambda \nabla_{\theta} L(\theta_t; x_i; y_i)$$

# Training Deep Neural Networks

## Stochastic Gradient Descent

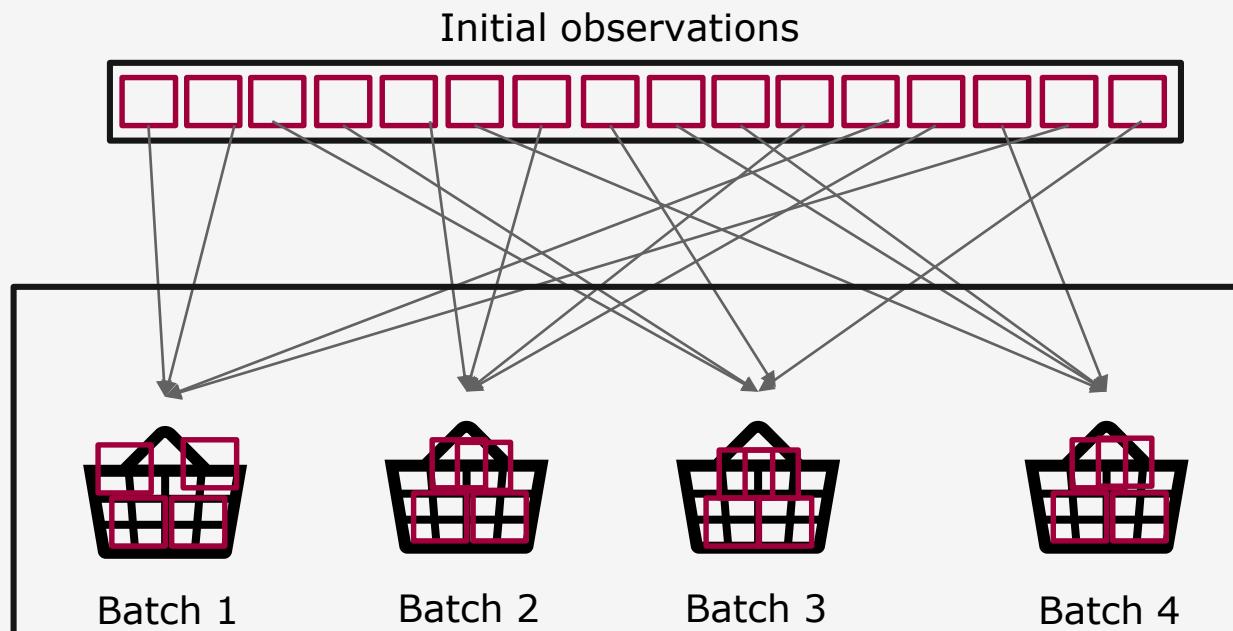
### Batch/mini-batch learning



# Training Deep Neural Networks

## Stochastic Gradient Descent

Constituting an epoch



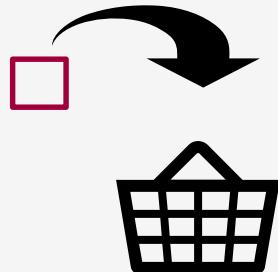
# Training Deep Neural Networks

## Stochastic Gradient Descent

Which batch size (bs) ?

If batch size too small

- Learning too small
- Regularizing effect expected  
(generalization error best with bs 1)



If batch size too large

- Large bs → Better estimate of the gradient
- Faster learning
- Too large → degradation of the gen. accuracy

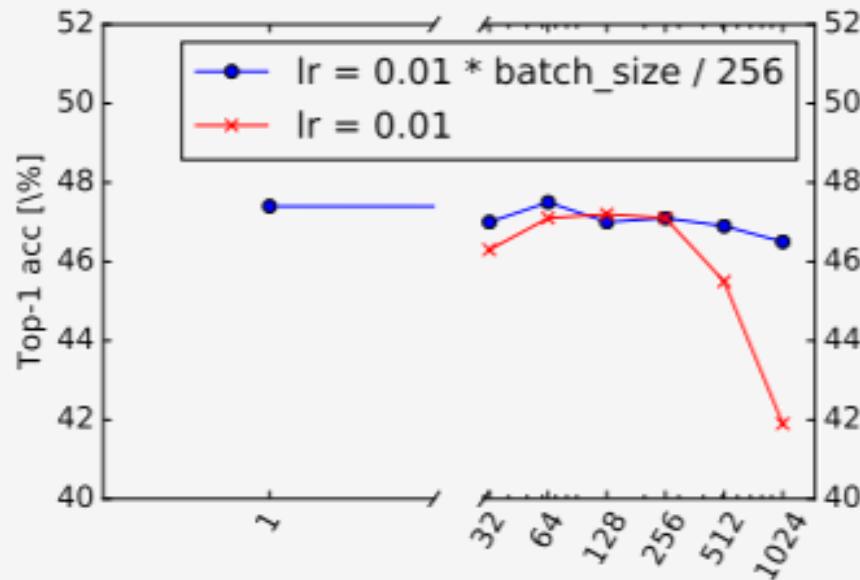


# Training Deep Neural Networks

## Stochastic Gradient Descent

Which batch size (bs) ?

Batch size and initial learning rate  
impact to the accuracy

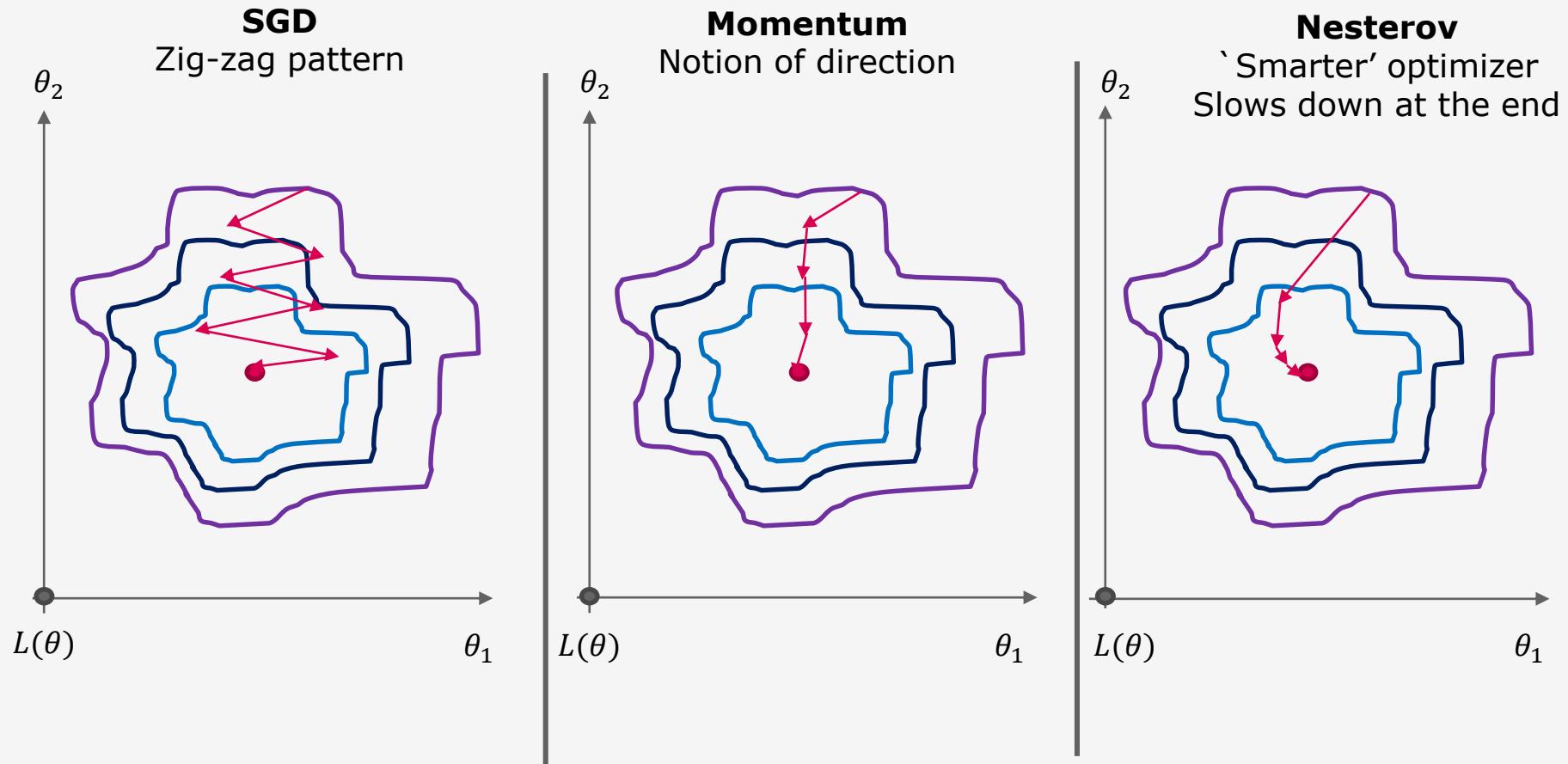


<https://arxiv.org/pdf/1606.02228.pdf>

# Training Deep Neural Networks

## Stochastic Gradient Descent

- Stochastic Gradient Descent is a “first order method”
- Such as **Nesterov** and **Momentum**



# Training Deep Neural Networks

---

## Adaptive optimizers

- *Second order moment* methods
- They minimize  $J(\theta)$  not only by estimating the Gradient but also by estimating the **Hessian matrix** and using the **Newton's method**
- They **adapt the learning rate**  $\lambda$  at each step:

**Adagrad, Adadelta, Adamax, RMSprop, Adam**

.... For detailed description

# Training Deep Neural Networks

## Choosing the right optimization algorithm

### Adaptive (second order moment)

Adapted for  
Sparse input data

No tuning  
Required for  $\lambda$



Faster  
convergence

### First order moment

Slow  
convergence

Can get stuck  
In saddle paths

Too reliant on  
Robust intialization



**Adaptive = best option**  
In particular Adam

# Introducing Keras

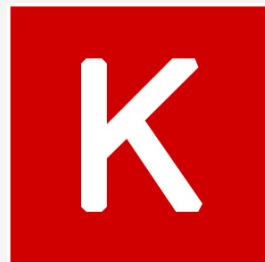
A little bit of practice !



# Introducing Keras

---

Deep Learning specialized library



Wrapper designed for  
fast implementation

On CPU or GPU



 TensorFlow™

The TensorFlow logo features a stylized orange 'T' composed of several small squares, followed by the word "TensorFlow" in a bold, sans-serif font, with a trademark symbol at the end.

 Microsoft  
CNTK

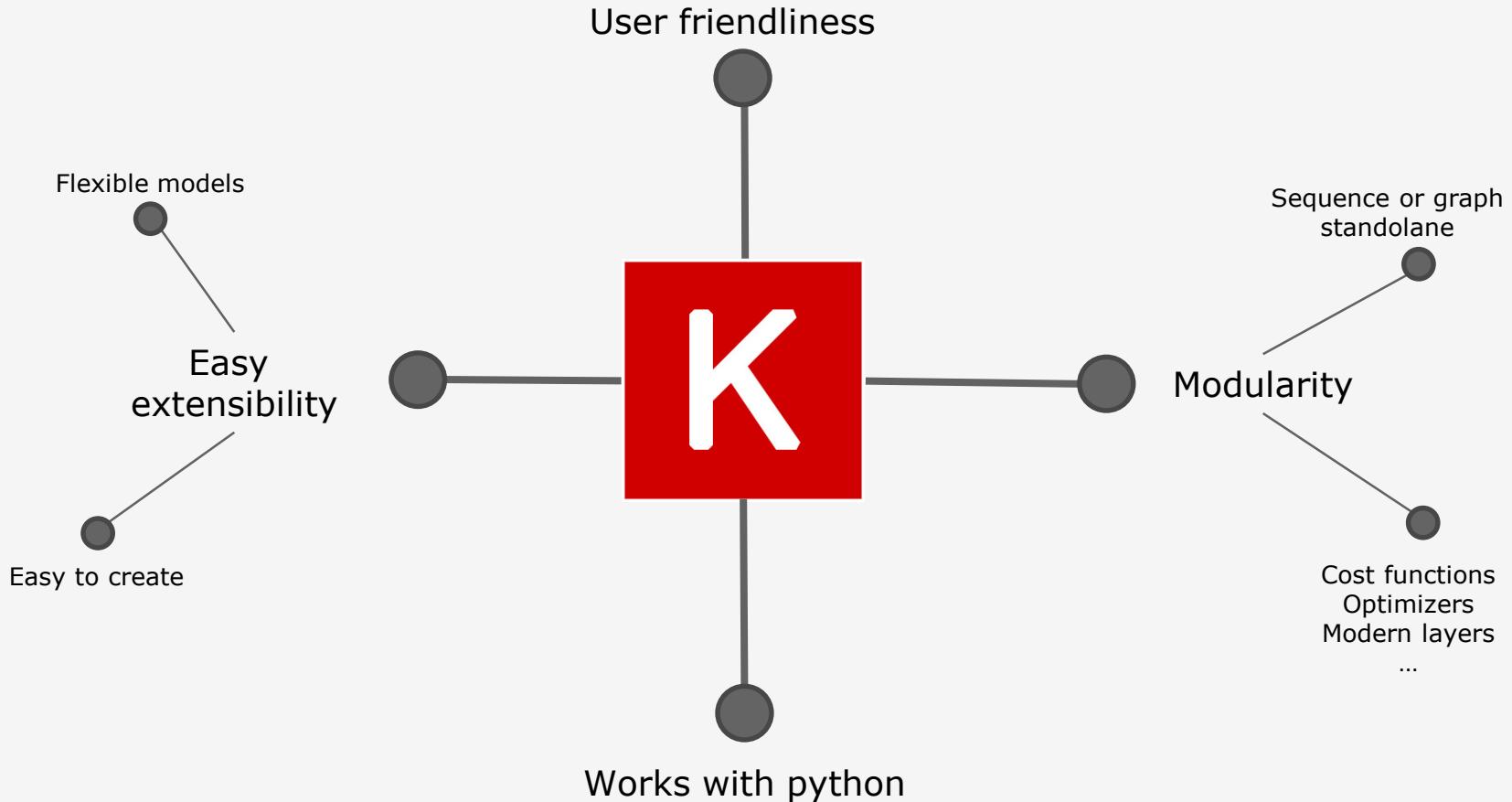
The Microsoft CNTK logo includes the Microsoft color scheme (blue, green, red, yellow squares) followed by the text "Microsoft" and "CNTK".

 theano

The theano logo is written in a large, blue, lowercase sans-serif font.

# Introducing Keras

---



# Introducing keras

## Design, train and predict

### 1 – Declare a **Sequential** model

```
import keras  
from keras.models import Sequential  
  
model = sequential()
```

### 2 – Stack all layers

```
from keras.layers import Dense, Activation  
  
model.add(Dense(units=64,input_dim=100))  
model.add(Activation('Relu'))  
model.add(Dense(units=32))  
model.add(Activation('Relu'))  
model.add(Dense(units=10))  
model.add(Activation('softmax'))
```

### 3 – Configure its learning process

```
model.compile(loss='categorical_crossentropy',  
              optimizer='adam',  
              metrics=['accuracy'])
```

### 4.A – Fit your model – automatic batch

```
model.fit(x_train, y_train,  
          epochs=10,  
          batch_size=32)
```

### 4.B – Or fit on your own batch

```
model.train_on_batch(x_batch,y_batch)
```

### 5 – Evaluate on your validation set

```
loss_metrics = model.evaluate(X_val, y_val,  
                             batch_size=32)
```

### 6 – Make your own predictions

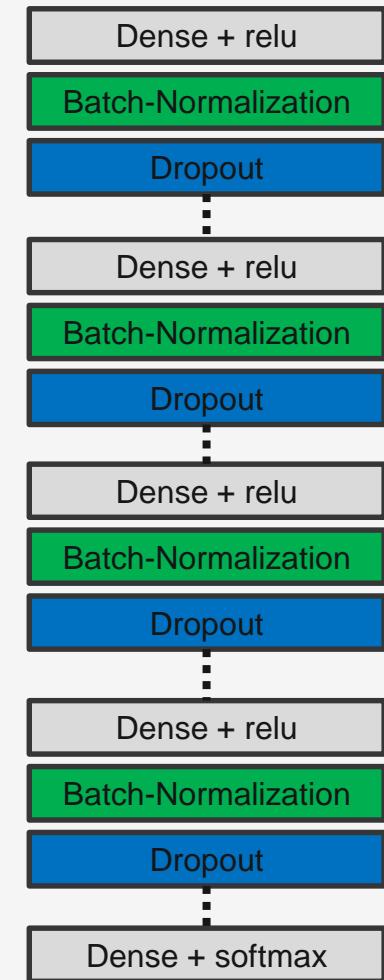
```
preds = model.predict(x_test,  
                      batch_size=32)
```

# Introducing keras

## Data augmentation, Dropout, Batch-Normalization

### A - Dropout and Normalization

```
from keras.layers import Dense, Dropout  
from keras.layers.normalization import BatchNormalization  
  
model.add(Dense(units=128,input_dim=100, activation='Relu'))  
model.add(BatchNormalization())  
model.add(Dropout(0.25))  
  
model.add(Dense(units=128, activation='Relu'))  
model.add(BatchNormalization())  
model.add(Dropout(0.25))  
  
model.add(Dense(units=64, activation='Relu'))  
model.add(BatchNormalization())  
model.add(Dropout(0.5))  
  
model.add(Dense(units=64, activation='Relu'))  
model.add(BatchNormalization())  
model.add(Dropout(0.5))  
  
model.add(Dense(units=10, activation='softmax'))
```



# Introducing keras

## Data augmentation, Dropout, Batch-Normalization

### B – Data Augmentation

1 – Declare all possible transformations

```
from keras.preprocessing.image import ImageDataGenerator  
  
train_datagen = ImageDataGenerator(rescale=1./255,  
                                   shear_range=0.2,  
                                   zoom_range=0.2,  
                                   horizontal_flip=True)  
  
test_datagen = ImageDataGenerator(rescale=1./255)
```

2 – Generate batches (flow from directory)

```
train_generator = train_datagen.flow_from_directory(  
    'data/train',  
    target_size=(150, 150),  
    batch_size=32,  
    class_mode='categorical')  
  
validation_generator = test_datagen.flow_from_directory(  
    'data/validation',  
    target_size=(150, 150),  
    batch_size=32,  
    class_mode='categorical')
```

3 – Fit on these generated batches

```
model.fit_generator( train_generator,  
                     steps_per_epoch=2000,  
                     epochs=50,  
                     validation_data=validation_generator,  
                     validation_steps=800)
```

4 – predict on the test set

```
test_generator = test_datagen.flow_from_directory(  
    'data/test',  
    target_size=(150, 150),  
    batch_size=32,  
    class_mode='categorical')  
  
model.predict_generator( test_generator,  
                       steps=total_size/batch_size)
```

# Introducing keras

---

Design, train and predict

Quite easy right ?!?



Just need to wait the  
model to run !

....

It could take a while



# Study case n°1

---

Design, train and predict



MNIST  
Handwritten Digit  
Recognizer

Using MLP

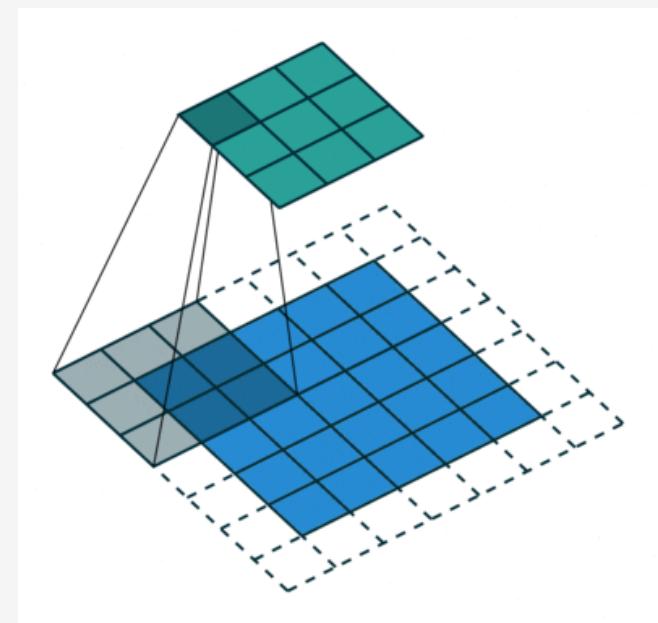
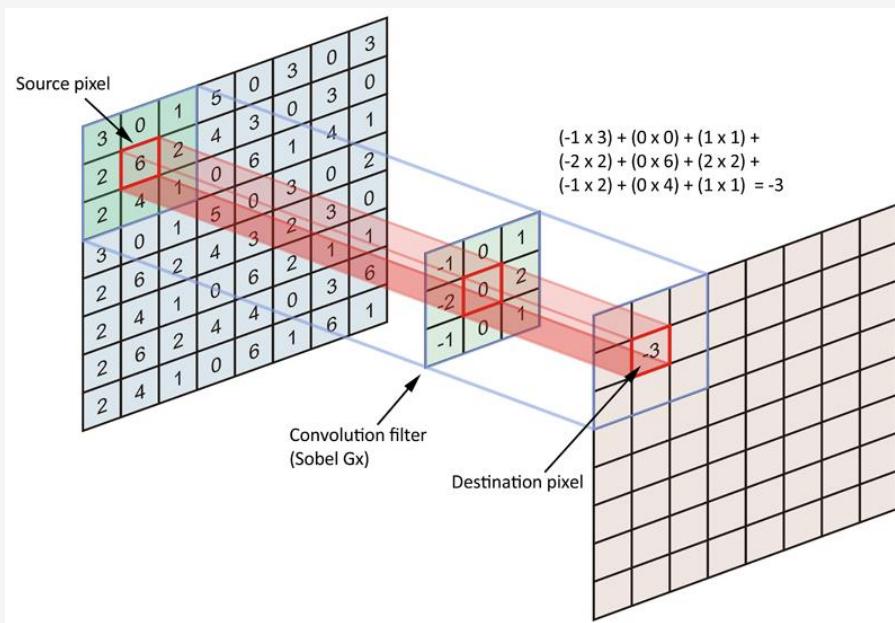
# Convolutional Neural Networks



# Convolutional Neural Networks

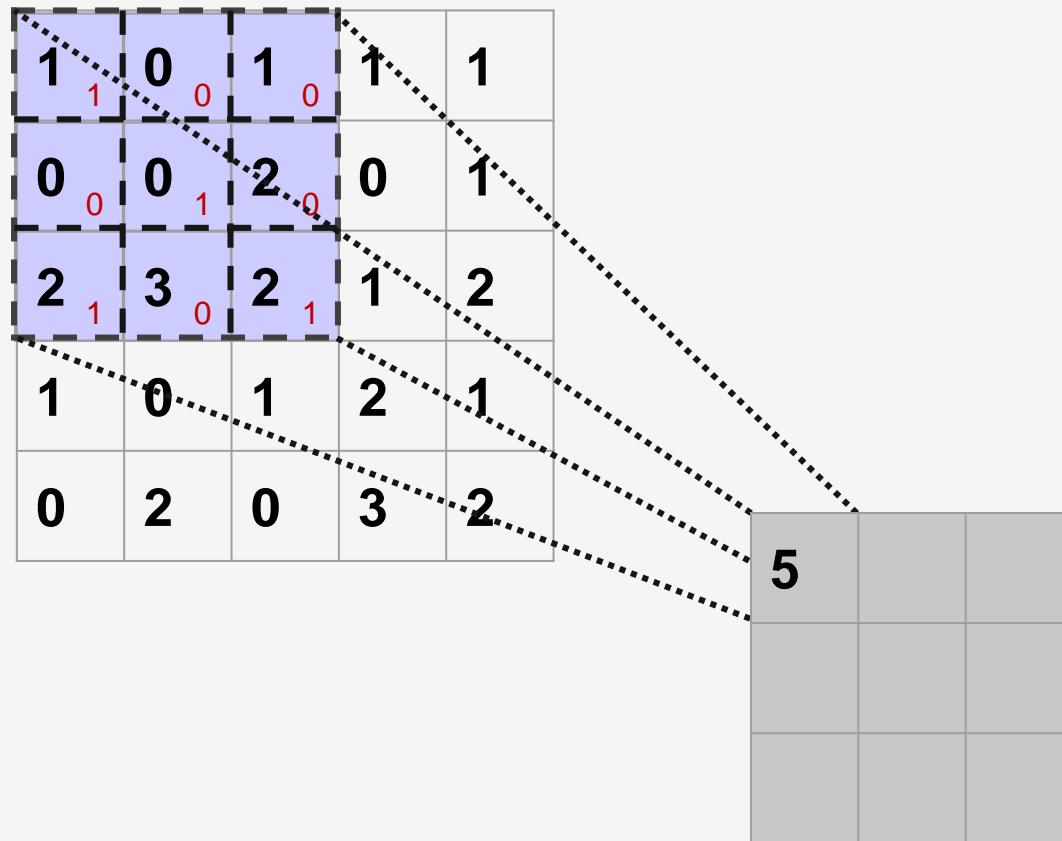
## Convolution as core mechanism of CNNs

Convolution  $\equiv$  Cross-Correlation



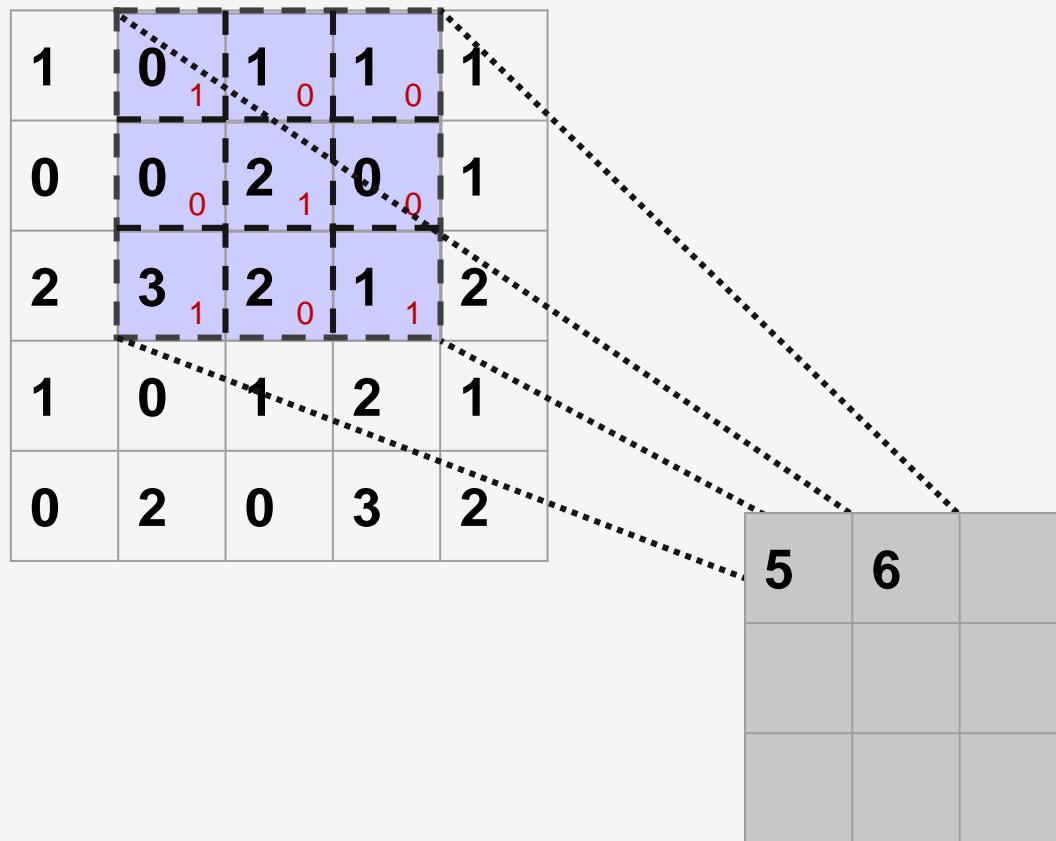
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



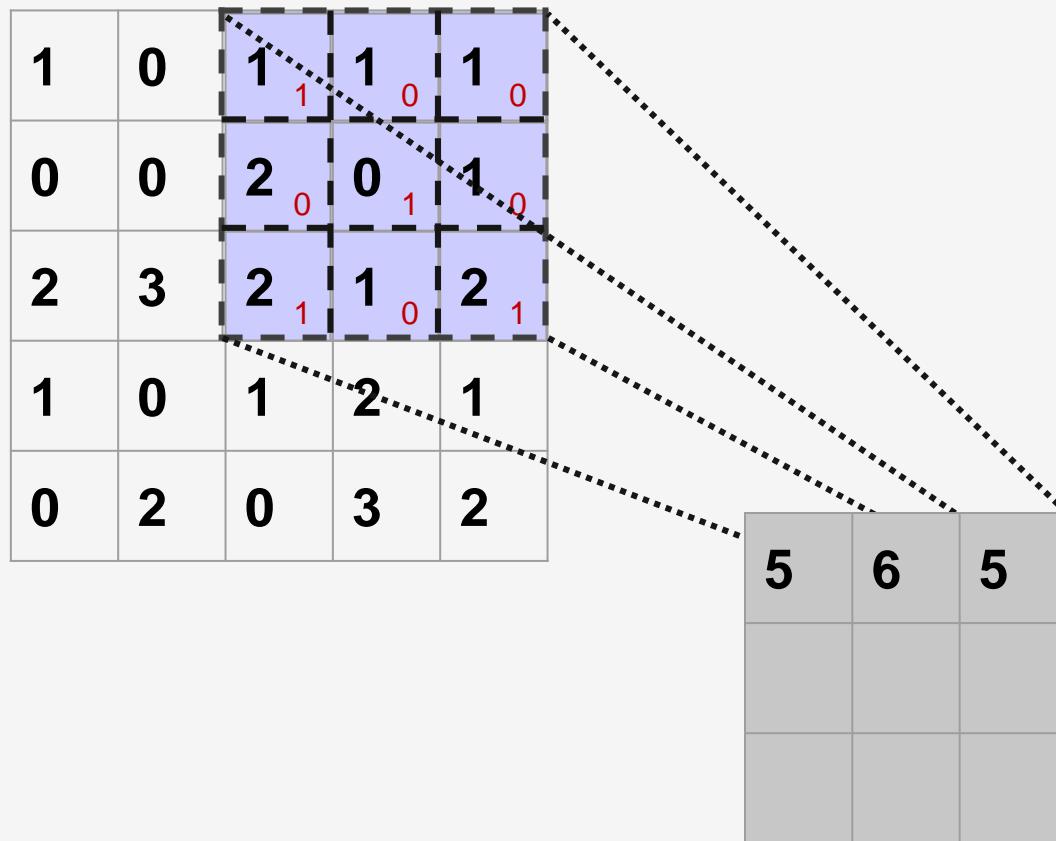
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



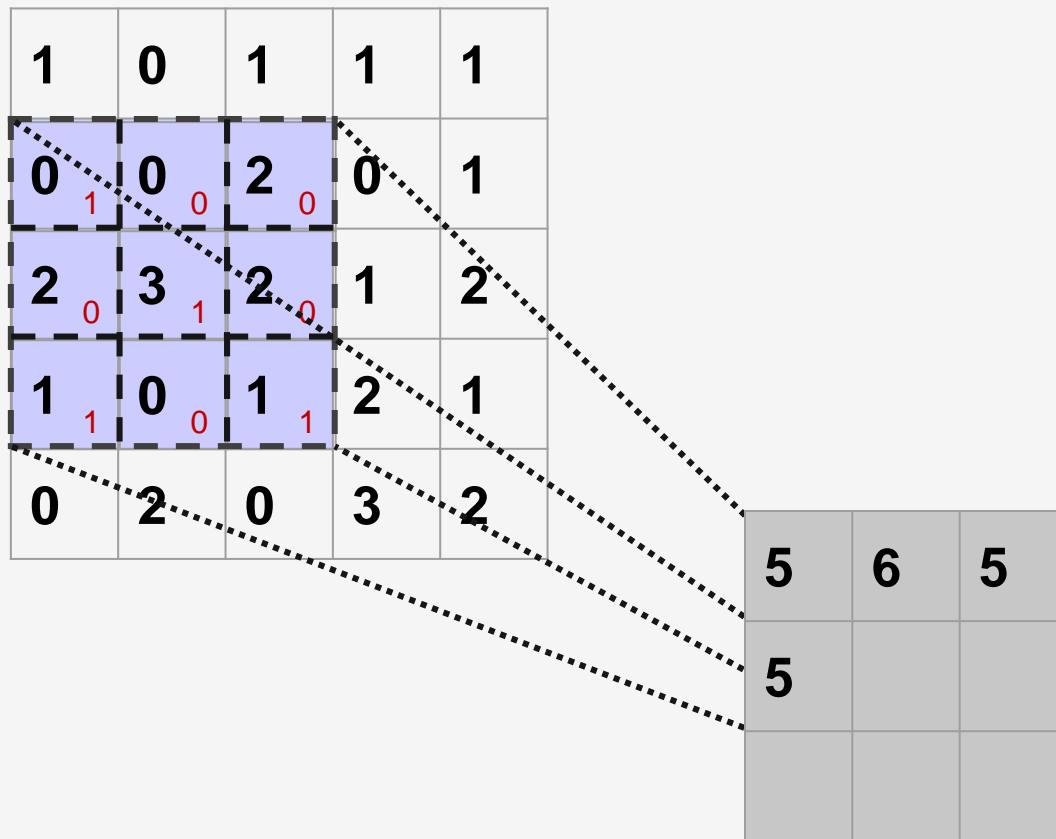
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



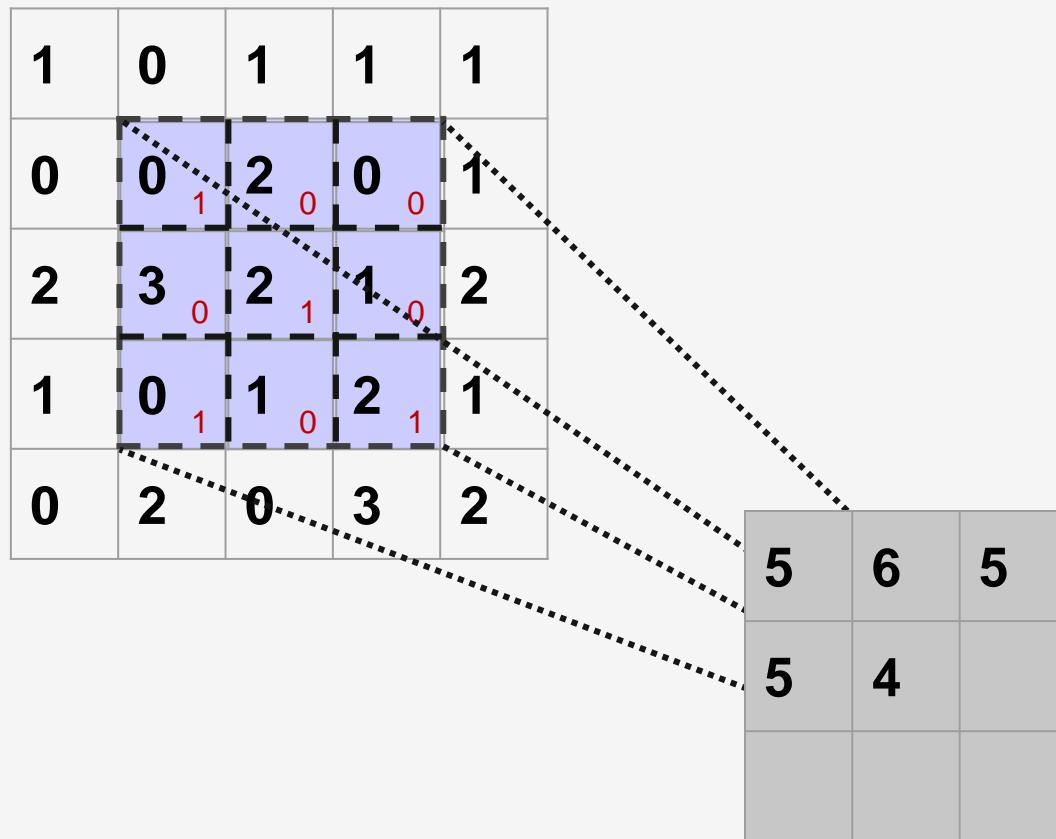
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



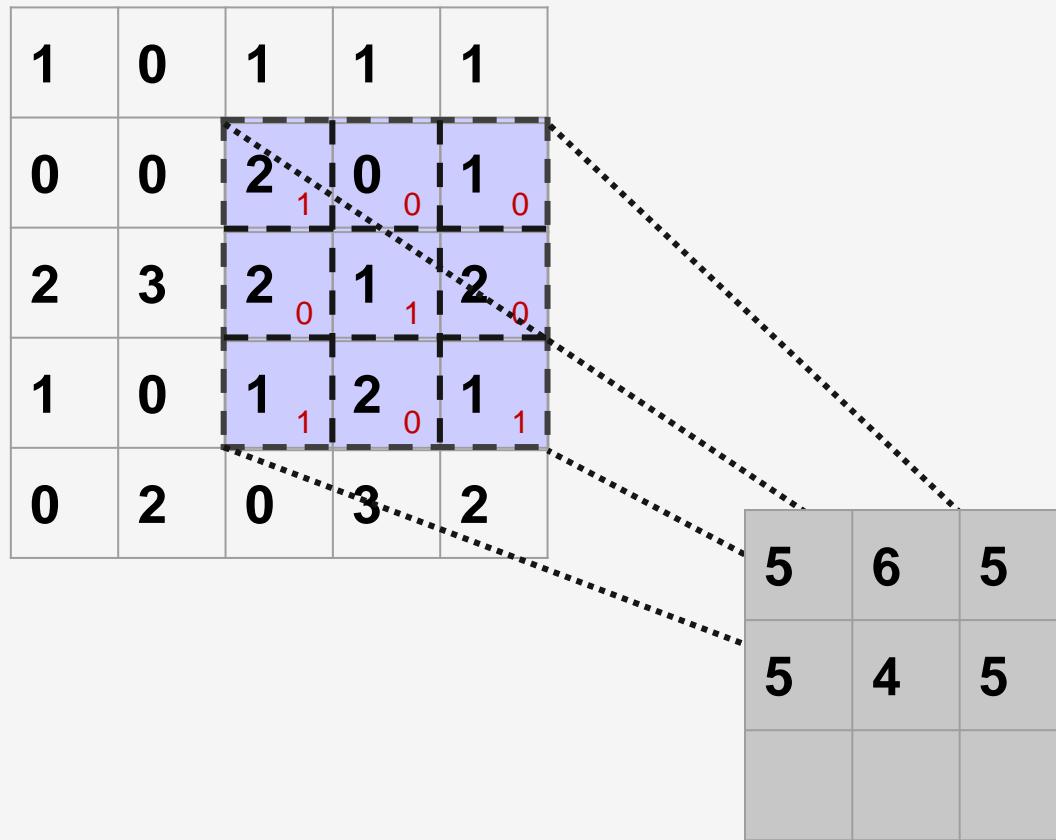
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



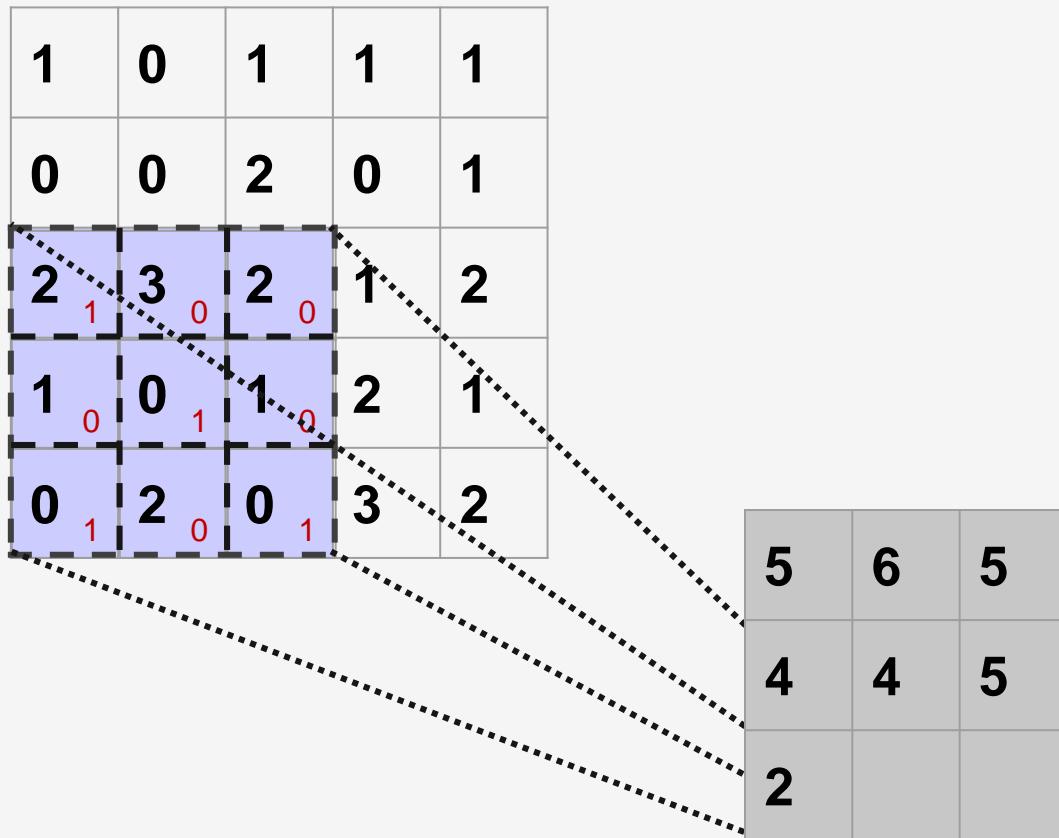
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



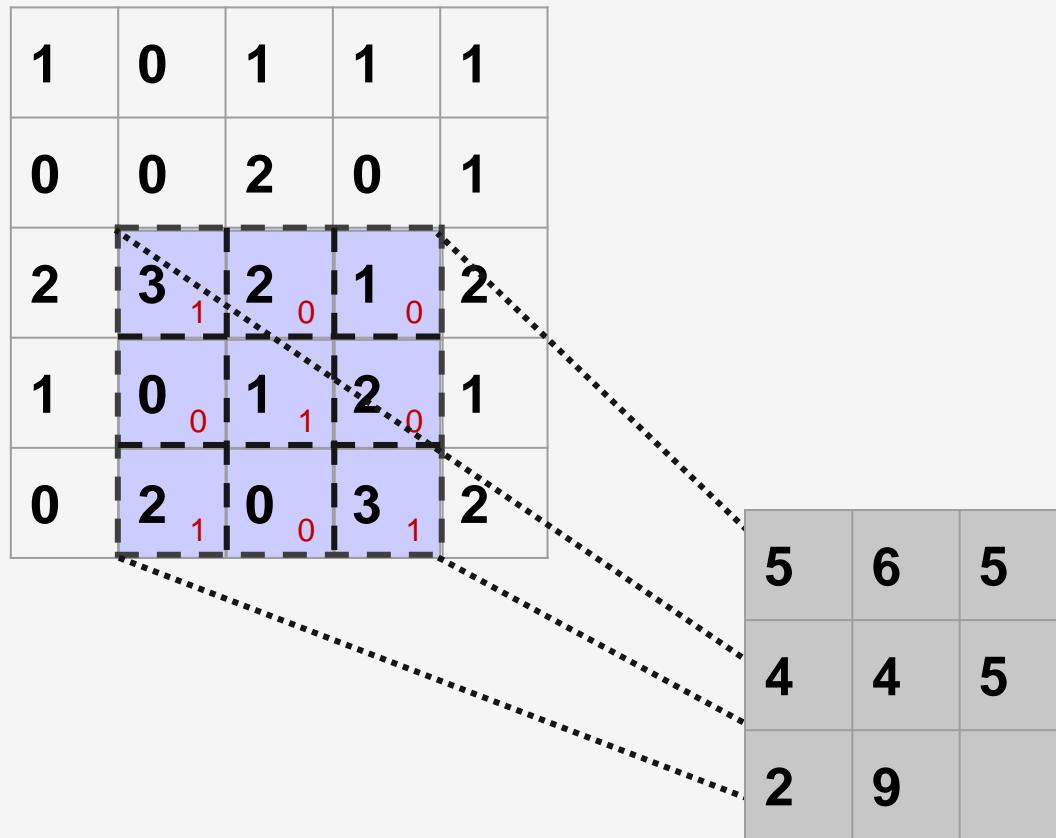
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



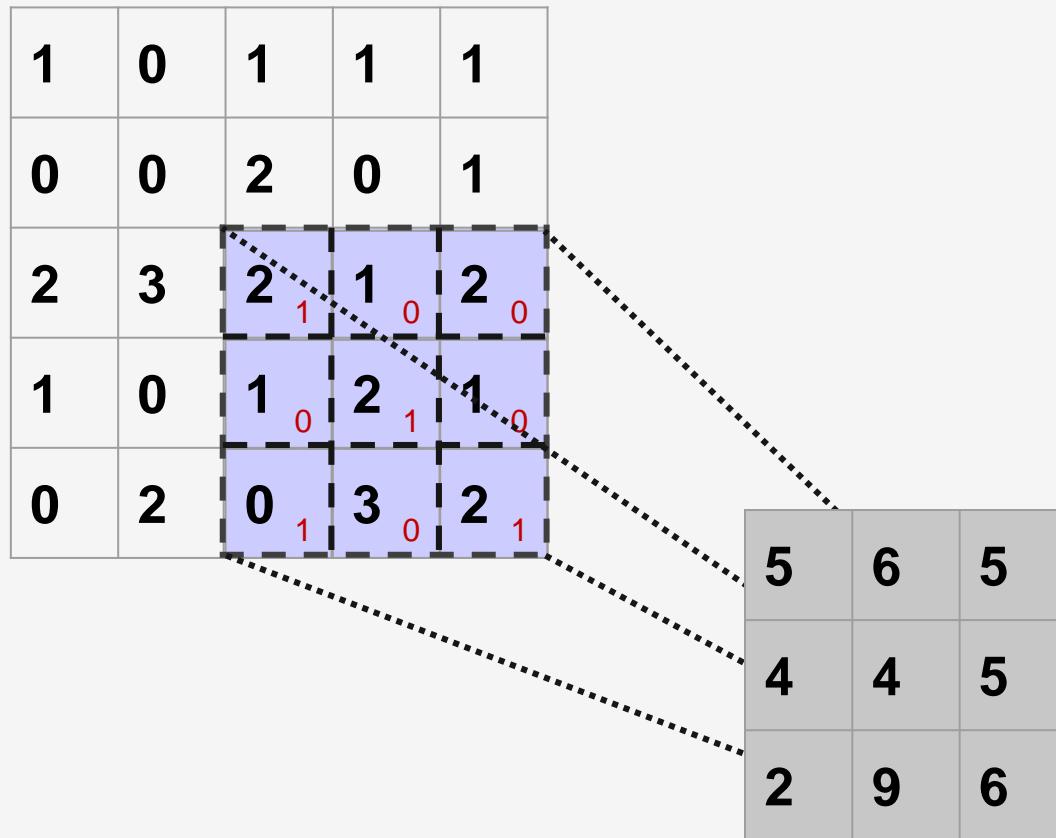
# Convolutional Neural Networks

Convolution as core mechanism of CNNs



# Convolutional Neural Networks

Convolution as core mechanism of CNNs



# Convolutional Neural Networks

---

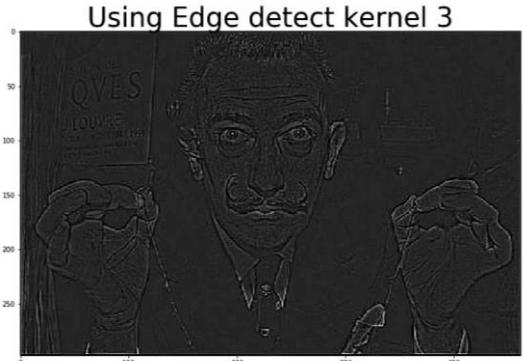
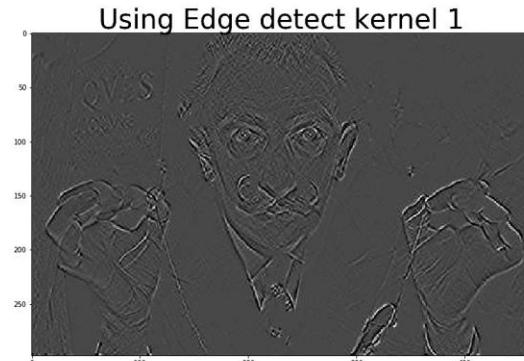
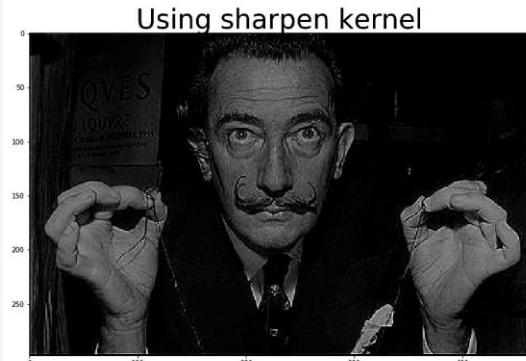
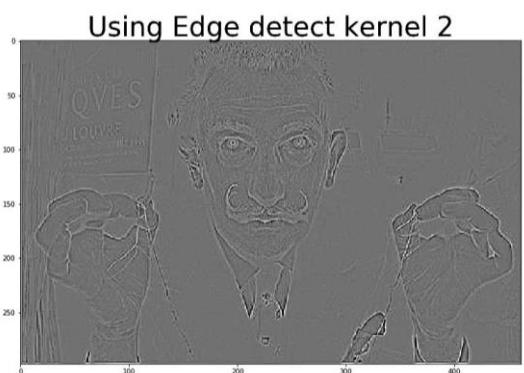
## Convolution as core mechanism of CNNs

- Notion of **moving average**
- Represents a **filter**
- Importance of *shape* and *weights* of the filter matrix (kernel)
- Each kernel extracts a specific kind of features (edge, color, shape ...)

# Convolutional Neural Networks

## Convolution as core mechanism of CNNs

### Using different kernels



# Convolutional Neural Networks

---

## CNN

- Particular kind of deep forward networks  
→ **Weights** and **bias** to learn as well
- Convolutional layers filter inputs for useful information
- They automatically adjust to extract the information useful for the Computer Vision task (classification, object detection, ...)
- Basic CNN are based on three main ideas:
  - Local receptive fields
  - Shared weights and bias
  - Pooling layers

# Convolutional Neural Networks

## CNN – Basics

Input image

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 2 | 0 | 1 |
| 2 | 3 | 2 | 1 | 2 |
| 1 | 0 | 1 | 2 | 1 |
| 0 | 2 | 0 | 3 | 2 |

Local receptive field

MaxPooling

Shared weights  
and bias

Output after  
convolution

Output of MaxPooling

|   |  |
|---|--|
| 6 |  |
|   |  |

# Convolutional Neural Networks

## CNN – Basics

Input image

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 2 | 0 | 1 |
| 2 | 3 | 2 | 1 | 2 |
| 1 | 0 | 1 | 2 | 1 |
| 0 | 2 | 0 | 3 | 2 |

Local receptive field

MaxPooling

Shared weights  
and bias

Output after  
convolution

Output of MaxPooling

|   |   |   |
|---|---|---|
| 5 | 6 | 5 |
| 4 | 4 | 5 |
| 2 | 9 | 6 |

|   |   |
|---|---|
| 6 | 6 |
|   |   |

# Convolutional Neural Networks

## CNN – Basics

Input image

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 2 | 0 | 1 |
| 2 | 3 | 2 | 1 | 2 |
| 1 | 0 | 1 | 2 | 1 |
| 0 | 2 | 0 | 3 | 2 |

Local receptive field

Shared weights  
and bias

Output after  
convolution

MaxPooling

|   |   |
|---|---|
| 6 | 6 |
| 9 |   |

Output of MaxPooling

# Convolutional Neural Networks

## CNN - Basics

Input image

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 2 | 0 | 1 |
| 2 | 3 | 2 | 1 | 2 |
| 1 | 0 | 1 | 2 | 1 |
| 0 | 2 | 0 | 3 | 2 |

Local receptive field



MaxPooling

|   |   |   |
|---|---|---|
| 5 | 6 | 5 |
| 4 | 4 | 5 |
| 2 | 9 | 6 |

Output after convolution

|   |   |
|---|---|
| 6 | 6 |
| 9 | 9 |

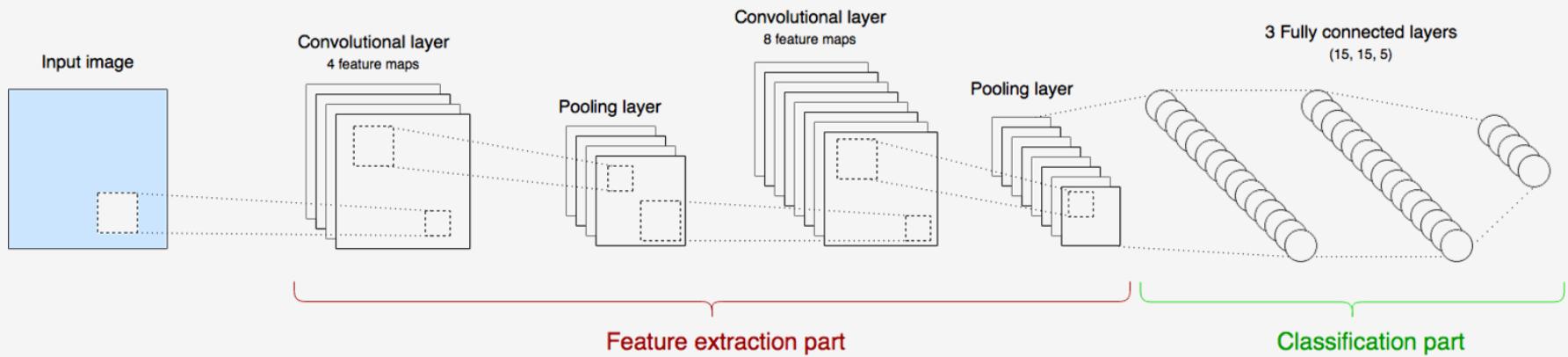
Output of MaxPooling

Shared weights  
and bias

# Convolutional Neural Networks

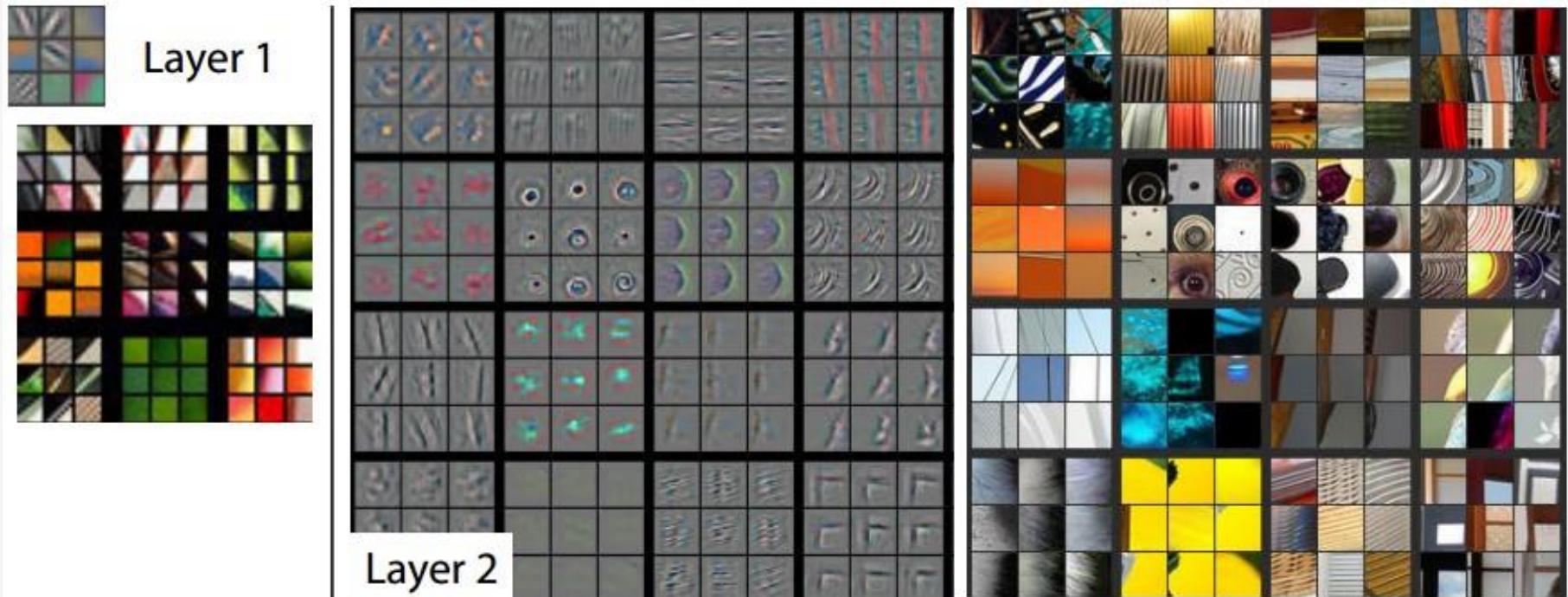
## CNN

Illustration of a small CNN



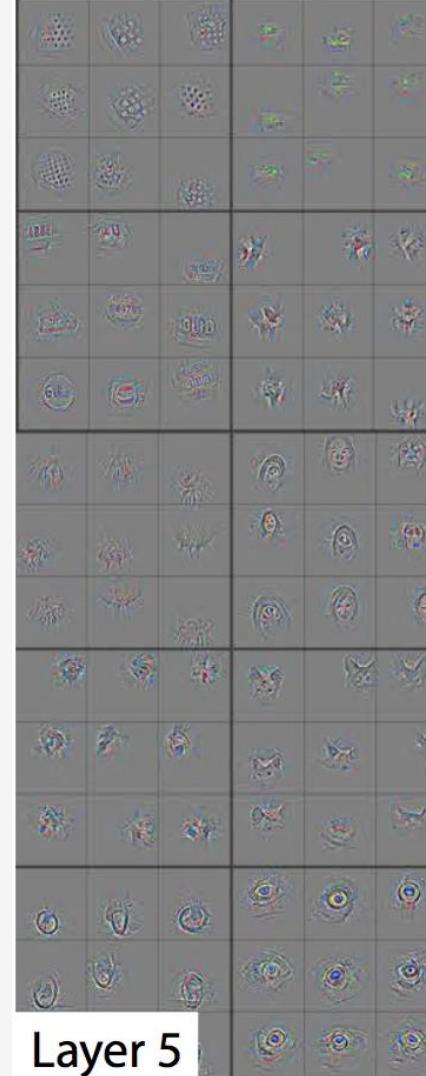
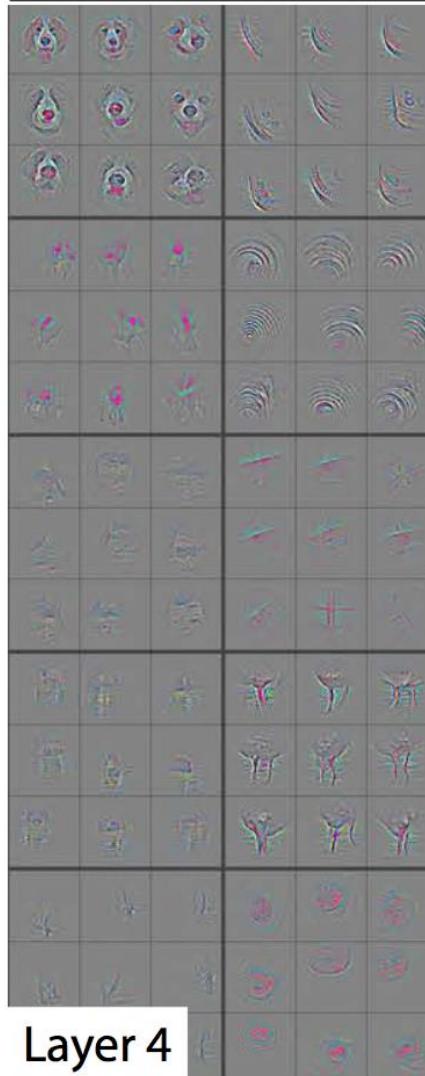
# Convolutional Neural Networks

CNN – illustration of what layers see



# Convolutional Neural Networks

CNN – illustration of what layers see



# Convolutional Neural Networks

---

Famous CNN architectures



## **Large Vision Recognition Challenge (ILSVRC 2010 – 2017)**

<http://www.image-net.org/>

**1000**  
Object  
classes

**+ 1,2 M**  
Training  
images

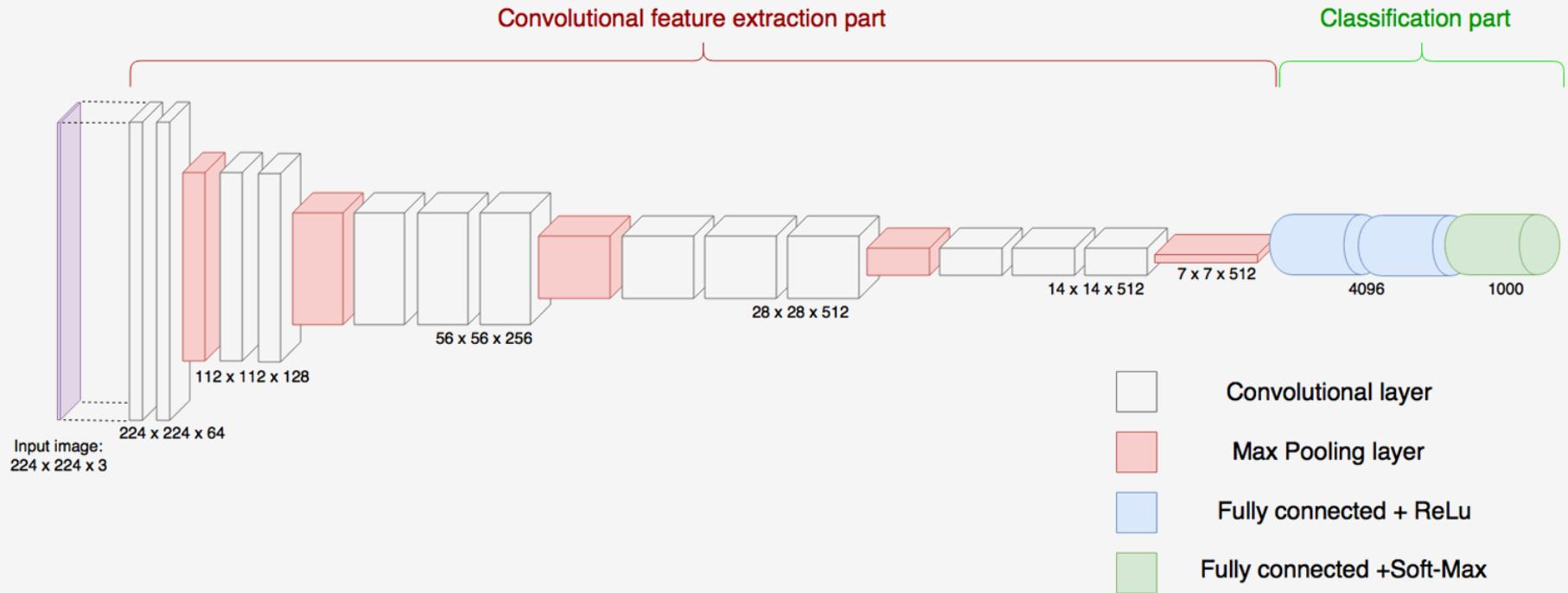
**+ 100 k**  
Test  
images

# Convolutional Neural Networks

## Famous CNN architectures

### Very Deep CNN – VGG16

*top-5 test error of around 8% (2014)*

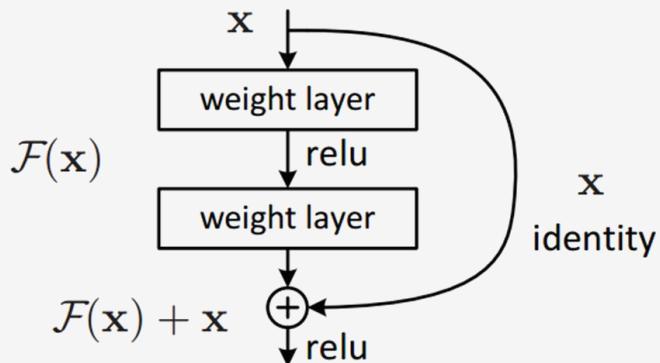


# Convolutional Neural Networks

## Famous CNN architectures

### Residual Networks - ResNet

*top-5 test error of around 3.57 % (2015)*



- with 152 layers it is **8 times** deeper than VGG16
- Residual Learning intuition: multiple non-linear function can asymptotically approximate the residual function  $F(x) = H(x) - x$
- It is then possible to fit:  $H(x) := F(x) + x$
- Ease training of **critically Deep models**
- Tackles the degradation of the learning process we usually meet with critically deep networks

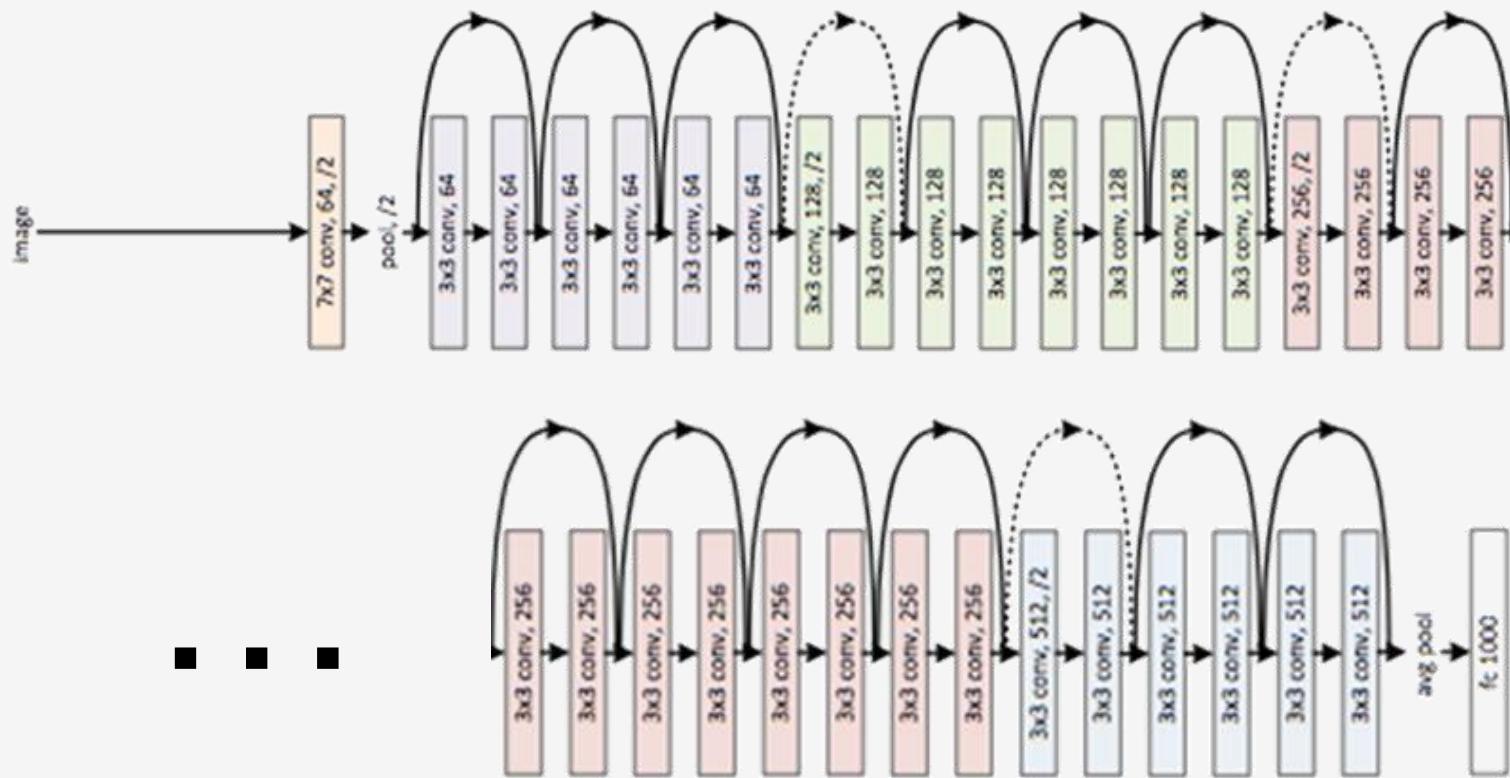
# Convolutional Neural Networks

## Famous CNN architectures

### Residual Networks - ResNet

*top-5 test error of around 3.57 % (2015)*

34-layer residual



# Convolutional Neural Network

## Building a CNN

```
from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten

def deep_conv_model():
    model = Sequential()
    model.add(Conv2D(64, (5,5), input_shape = (28,28,1), activation = 'relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

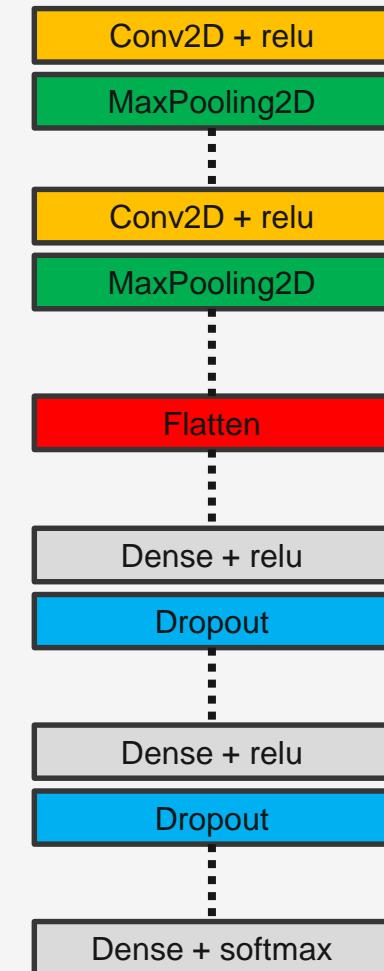
    model.add(Conv2D(64, (5, 5),activation = 'relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(10, activation='softmax'))

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model

model = deep_conv_model()
model.summary()
```



# Study case n°2

---

Design, train and predict

MNIST  
Handwritten Digit  
Recognizer

Using CNN



# Transfer Learning



# Transfer Learning

## General intuition

*"Inspired by human beings' capabilities to transfer knowledge across tasks, transfer learning aims to leverage knowledge from a source domain to improve the learning performance or minimize the number of labeled examples required in a target domain." (Wei et al. (2014))*

If able to recognize lions and tigers



After training the  
classification part of the  
CNN

Transfer Learning approach

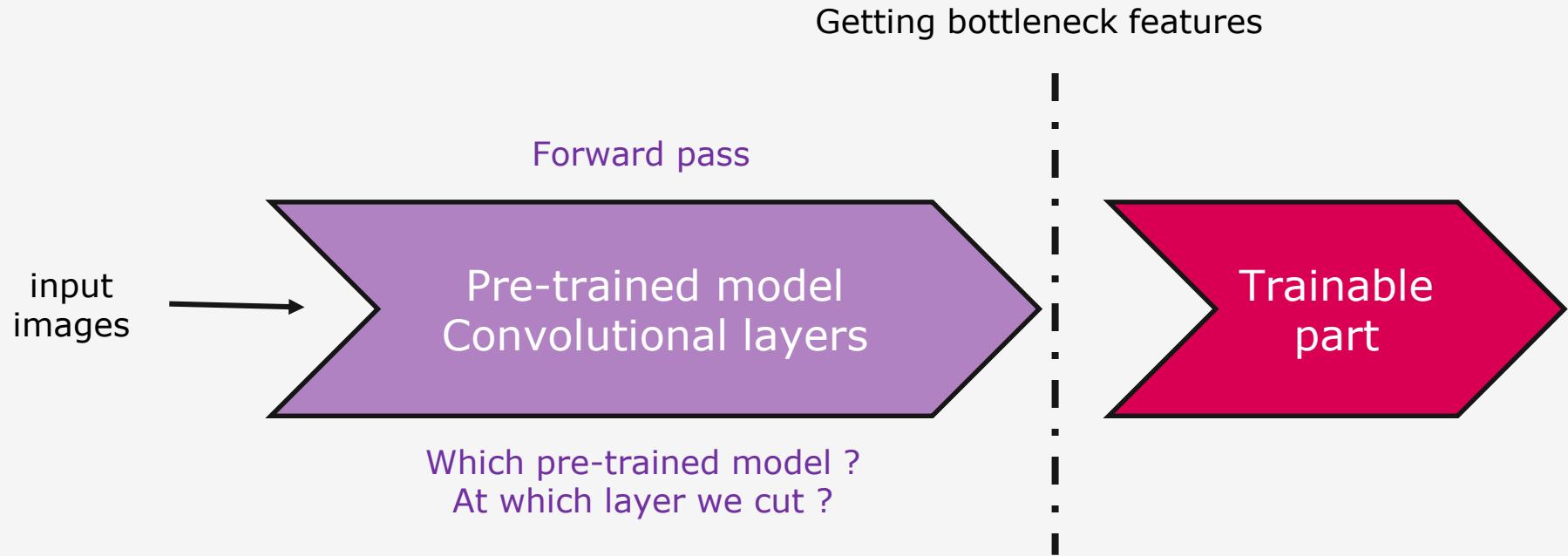
You can also recognize cats and dogs



# Transfer Learning

## General Scheme

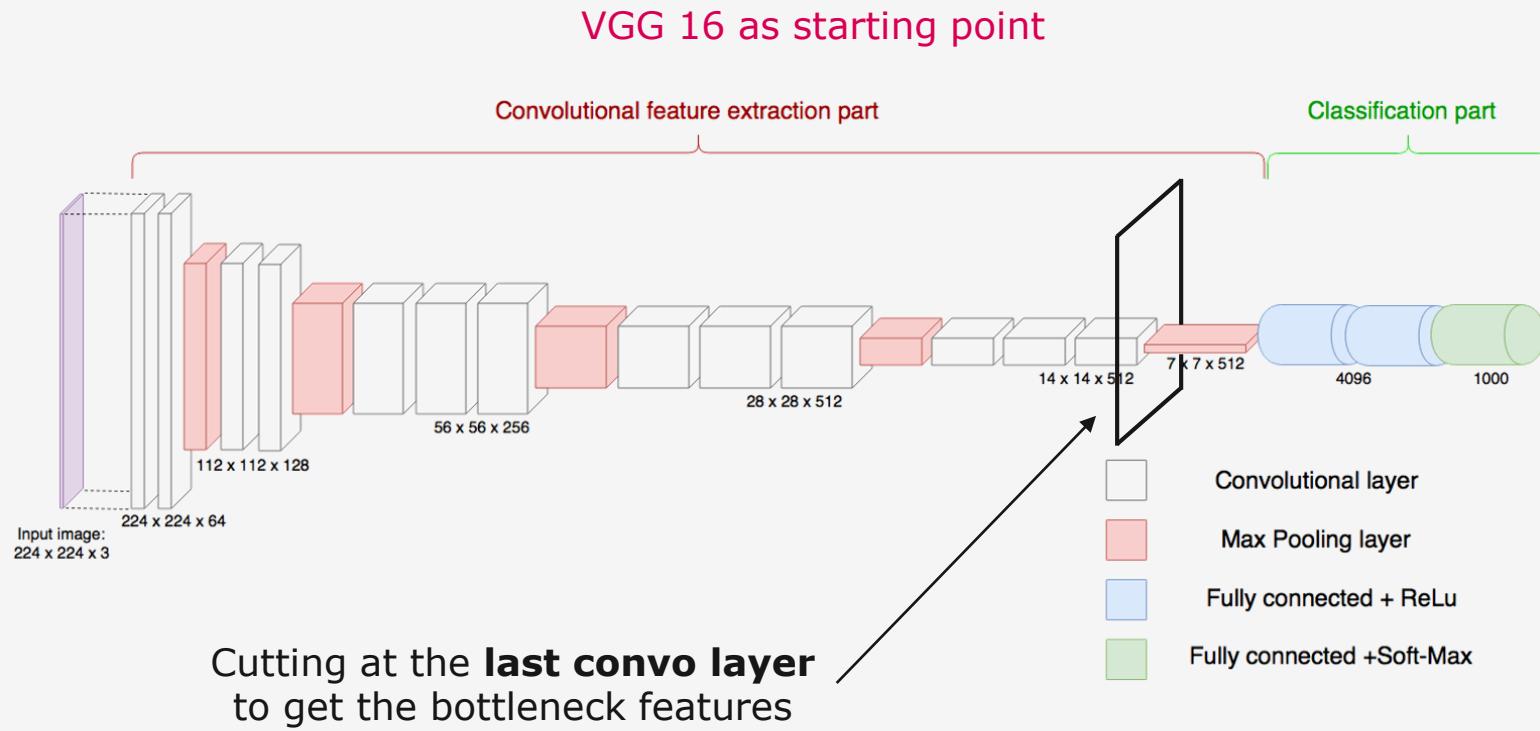
Transfer knowledge from one domain-to-another  
in the case of CNN



# Transfer Learning

## General Scheme

- Models famous for having outperformed state-of-the-art results in ImageNet competition
- Weights for VGG16, VGG19, Inception, ResNet50 and Xception are available in keras



# Study case n°3

---

Design, train and predict

Cats vs Dogs

Using Transfer  
Learning



# Recurrent Neural Networks



# Recurrent Neural Networks

---

What's a RNN?

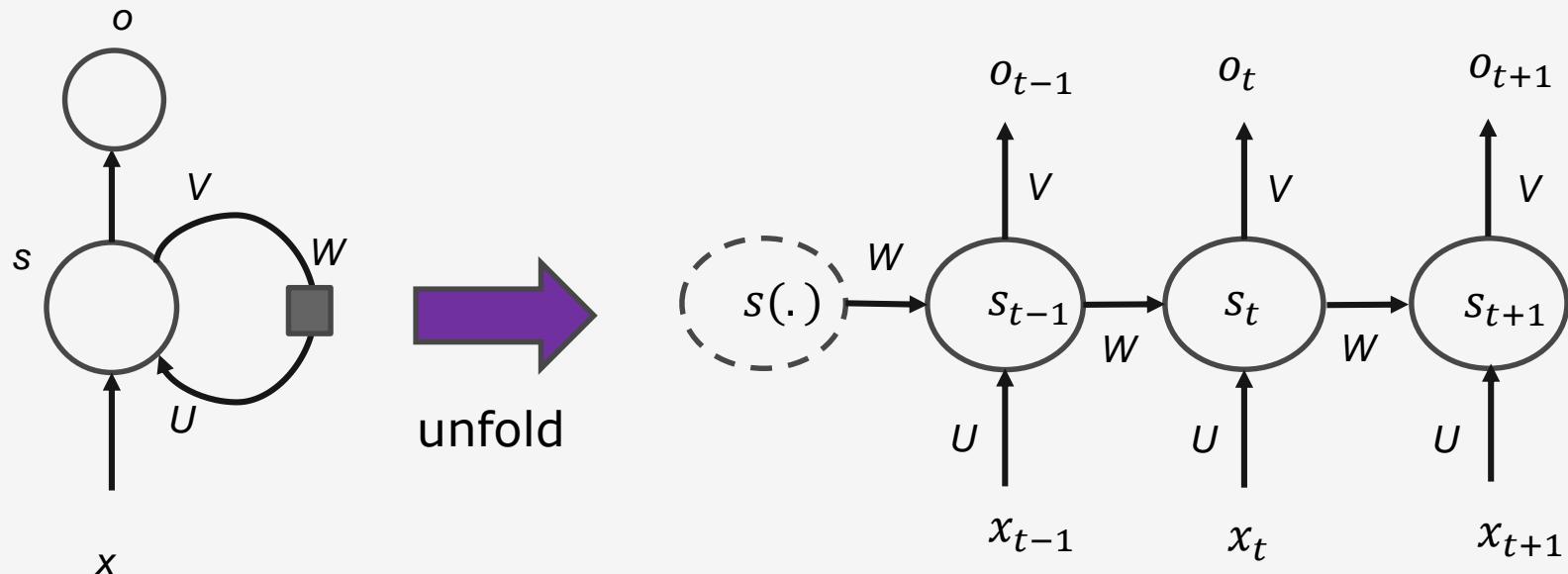
- Adapted to sequence treatment tasks: NLP, etc ...
- Parameters  $U$ ,  $V$  and  $W$
- Use of same parameters ( $U$ ,  $V$ ,  $W$ ) across all steps. → reflects the fact that we are performing the same task at each steps, just with different inputs.
- Sequence of inputs  $X(x_1, x_2, x_3, \dots, x_T)$  and outputs  $O(o_1, o_2, o_3, \dots, o_T)$
- States  $S(s_1, s_2, s_3, \dots, s_T)$  (also called  $H$  sometimes →  $s \equiv h$  )
- RNN can be expressed as:

$$s_t = \tanh(Ux_t + Ws_{t-1} + b)$$
$$o_t = \text{softmax}(Vs_t)$$

# Recurrent Neural Networks

What's a RNN?

- Inputs are not independent anymore → notion of memory



# Recurrent Neural Networks

---

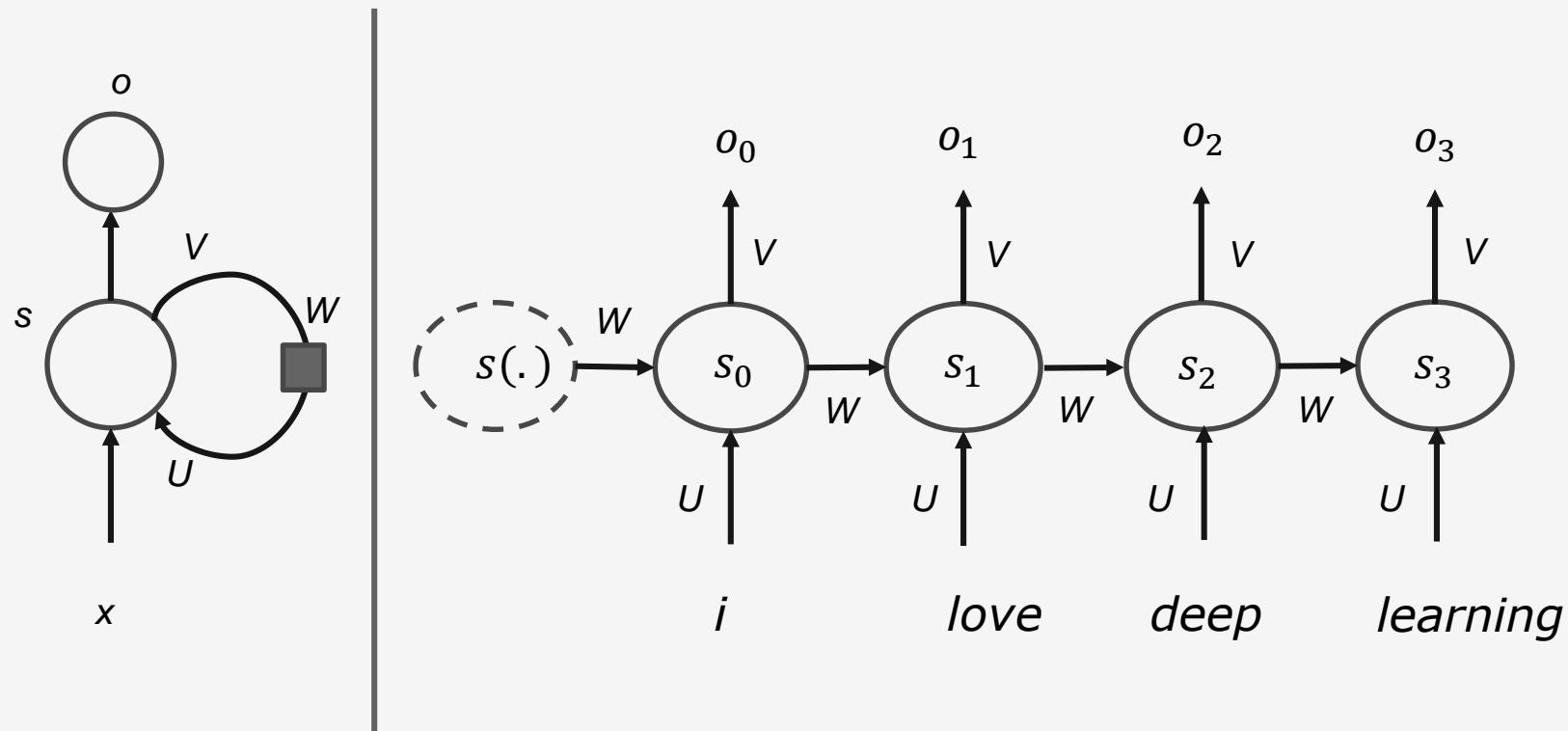
## What's a RNN?

- Let's take an example:
  - Input dim = 8000 → the dimension of the input vectors
  - Hidden layer size H = 100 → consider it as the "memory"
- Then we have:
  - $x_t \in R^{8000}$
  - $o_t \in R^{8000}$
  - $s_t \in R^{100}$
  - $U \in R^{100 * 8000}$
  - $V \in R^{8000 * 100}$
  - $W \in R^{100 * 100}$
- The number of parameters to compute:  $2HC + H^2$
- In our example: it makes 1 610 000 parameters

# Recurrent Neural Networks

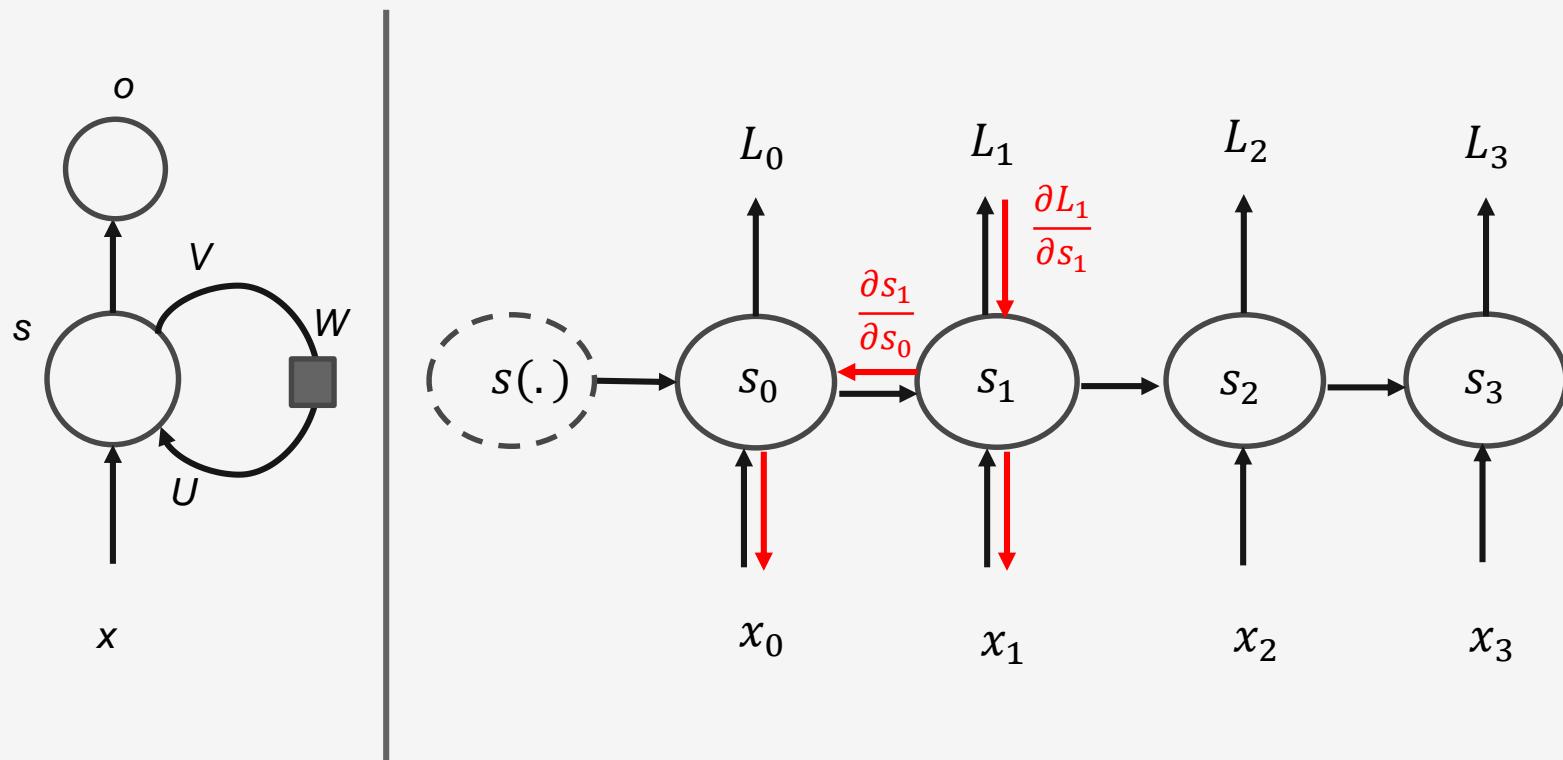
Illustration: example with a sentence

Sentence: " I love deep learning"



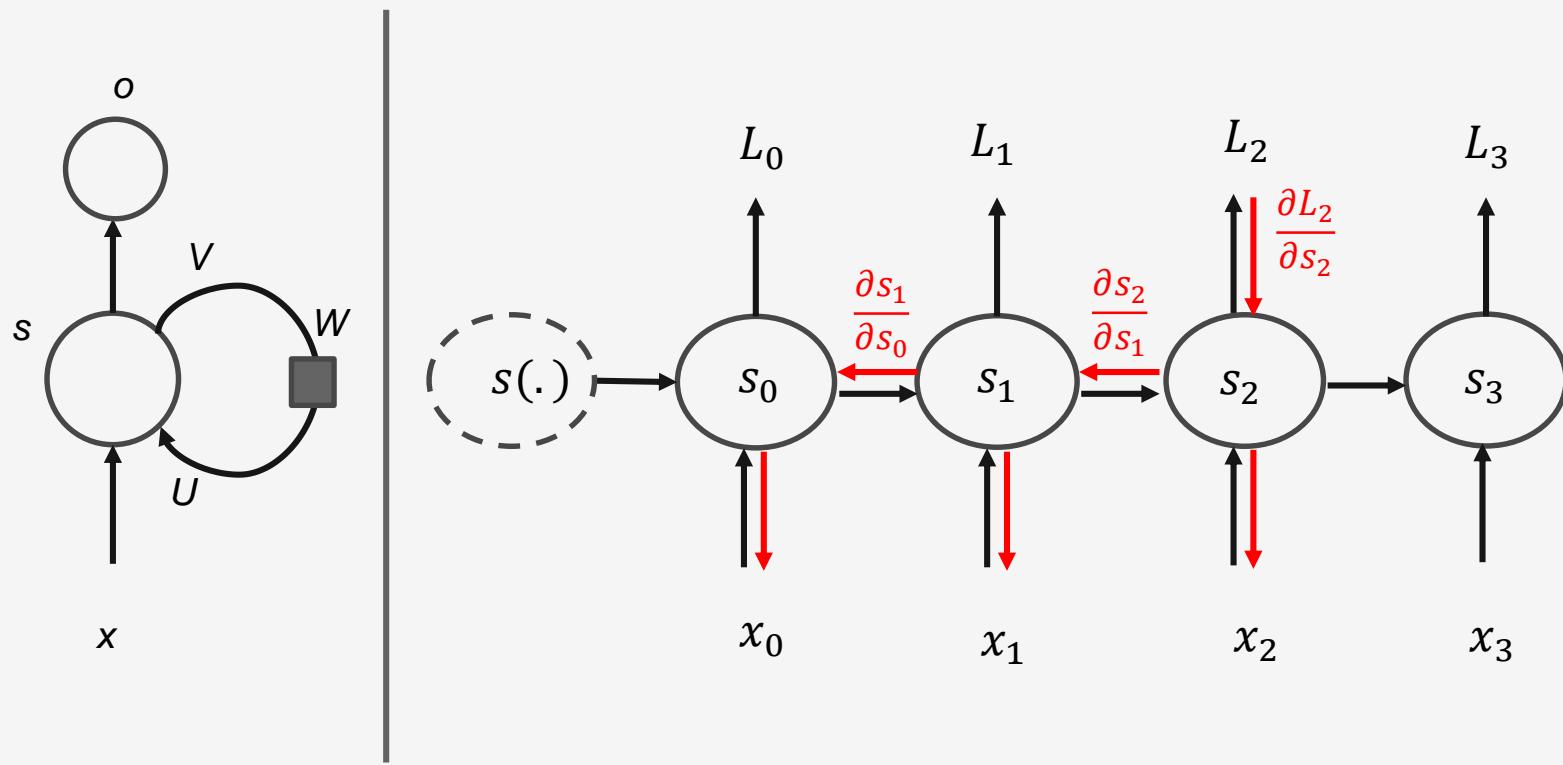
# Training a Recurrent Neural Networks

## Back-Propagation Through Time (BPTT)



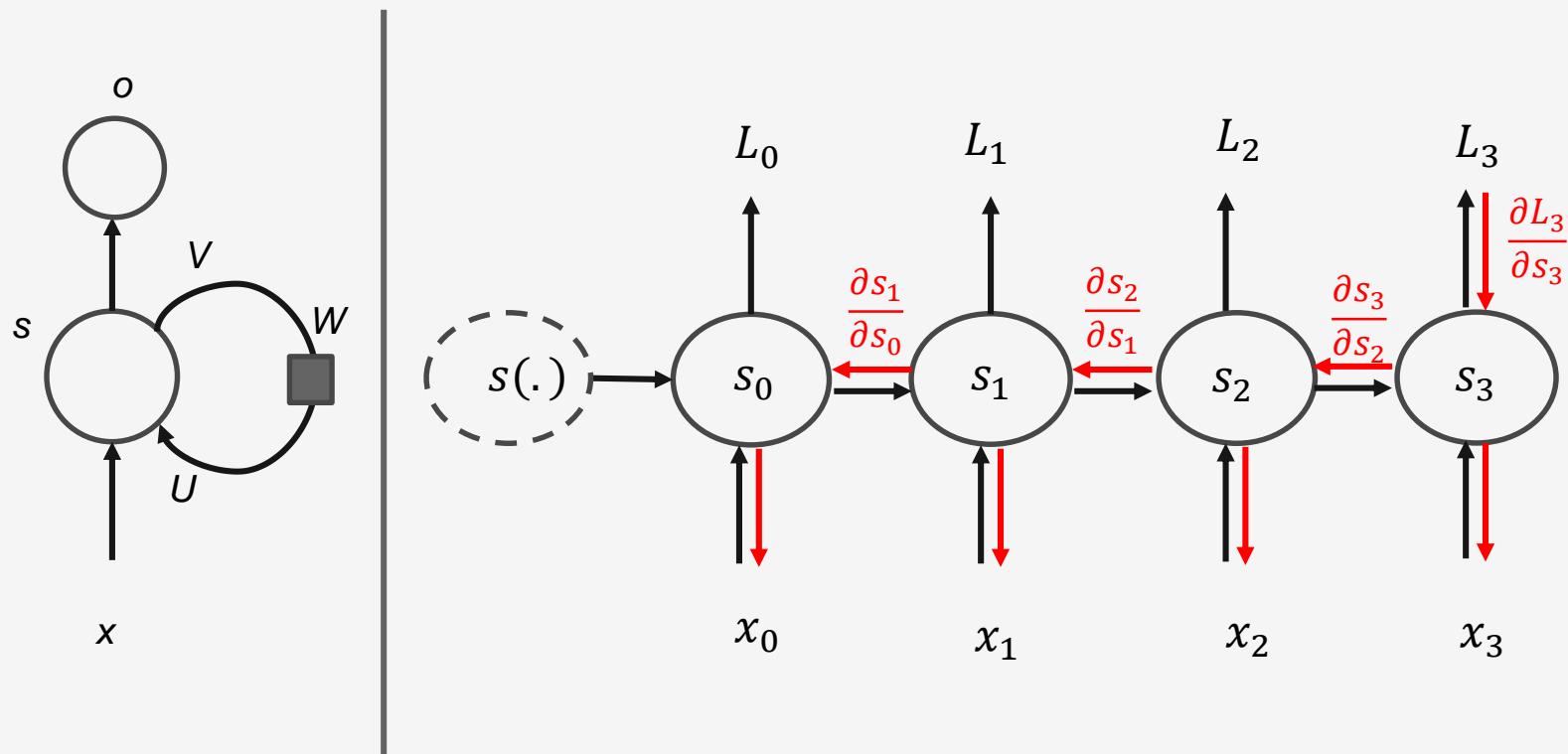
# Training a Recurrent Neural Networks

## Back-Propagation Through Time (BPTT)



# Training a Recurrent Neural Networks

## Back-Propagation Through Time (BPTT)



# Training a Recurrent Neural Networks

---

## Back-Propagation Through Time (BPTT)

For the numerical computations of gradients  
Please refer to the book  
« Deep Learning » (Bengio et al.)

...

# Complexified RNNs

## Gated Recurrent Units (GRU)

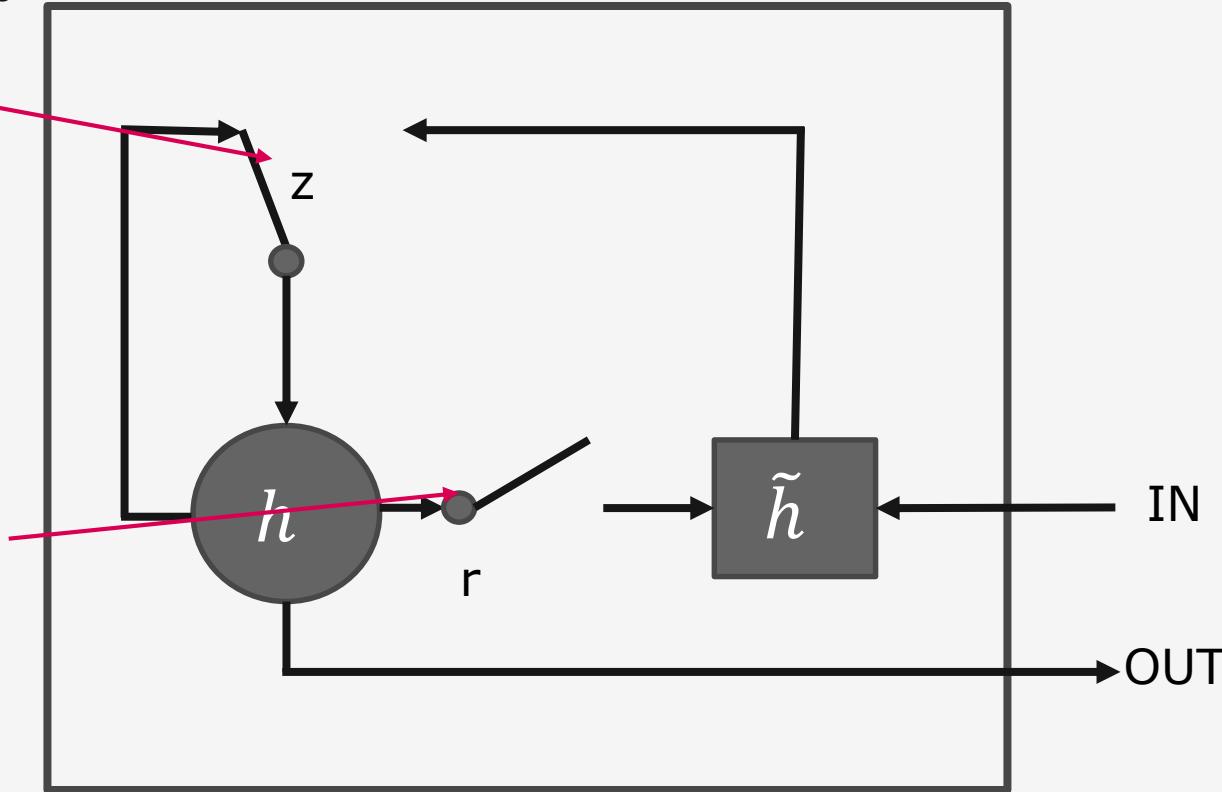
**Z = update gate**

*How much of the previous memory to keep around*

**r = reset gate**

*How much of the new input to combine with the previous memory*

Illustration of a GRU



# Complexified RNNs

## Gated Recurrent Units (GRU)

- With the two gates, we can express GRU as follows:

$$\begin{aligned} z_t &= \sigma(x_t U_z + h_{t-1} W_z + b_z) \\ r_t &= \sigma(x_t U_r + h_{t-1} W_r + b_r) \end{aligned}$$

Gates' equations

$$\begin{aligned} \tilde{h}_t &= \tanh(x_t U_h + (r_t \odot h_{t-1}) W_h + b_h) \\ h_t &= (1 - z_t) \odot \tilde{h}_t + z_t \odot h_{t-1} \end{aligned}$$

Update equations

...

- The number of parameters for LSTM is expressed as:

$$p = 3 (CH + H^2 + H)$$

# Complexified RNNs

## Gated Recurrent Units (GRU) - Implementation

```
import keras.backend as k

def forward_prop_step(x_t, s_t1_prev):
    # This is how we calculated the hidden state in a simple RNN. No longer!
    #  $s_t = \tanh(U[:, x_t] + k \cdot \text{dot}(W, s_{t1\_prev}))$ 

    # Get the word vector
    x_e = E[:, x_t]

    # GRU Layer
    z_t1 = k.hard_sigmoid(k.dot(U[0], x_e) + k.dot(W[0], s_t1_prev) + b[0])
    r_t1 = k.hard_sigmoid(k.dot(U[1], x_e) + k.dot(W[1], s_t1_prev) + b[1])
    c_t1 = k.tanh(k.dot(U[2], x_e) + k.dot(W[2], s_t1_prev * r_t1) + b[2])
    s_t1 = (k.ones_like(z_t1) - z_t1) * c_t1 + z_t1 * s_t1_prev

    # Final output calculation
    # The softmax returns a matrix with one row, we only need the row
    o_t = k.softmax(k.dot(V, s_t1) + c)[0]

    return [o_t, s_t1]
```

# Complexified RNNs

## Long Short Term Memory (LSTM)

Illustration of a LSTM

**f = forget gate**

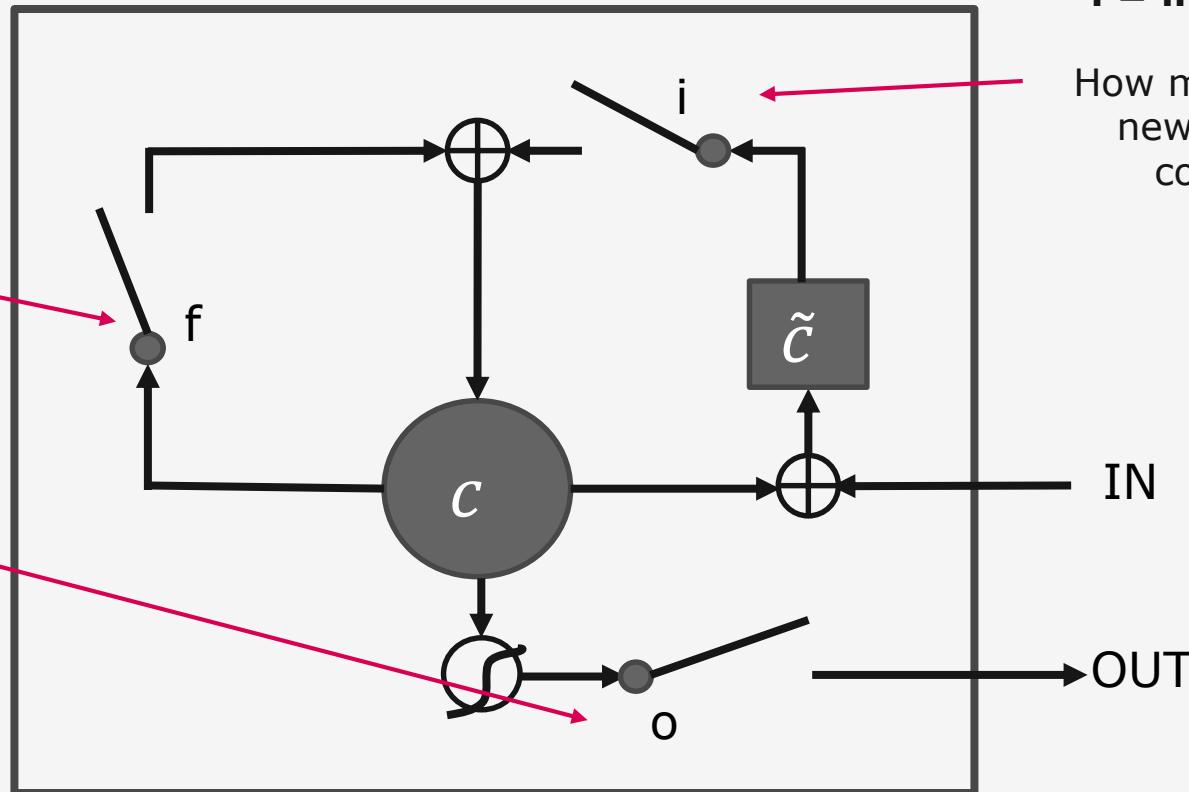
*How much of the previous memory to forget*

**o = output gate**

*Computes the output hidden state s*

**i = input gate**

*How much of the new state to consider*



# Complexified RNNs

## Long Short Term Memory (LSTM)

- With the three gates, we can express LSTM as follows:

$$\begin{aligned} i_t &= \sigma(x_t U_i + h_{t-1} W_i + b_i) \\ f_t &= \sigma(x_t U_f + h_{t-1} W_f + b_f) \\ o_t &= \sigma(x_t U_o + h_{t-1} W_o + b_o) \end{aligned}$$

Gates' equations

$$\begin{aligned} \tilde{c}_t &= \tanh(x_t U_c + h_{t-1} W_c + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

Update equations

...

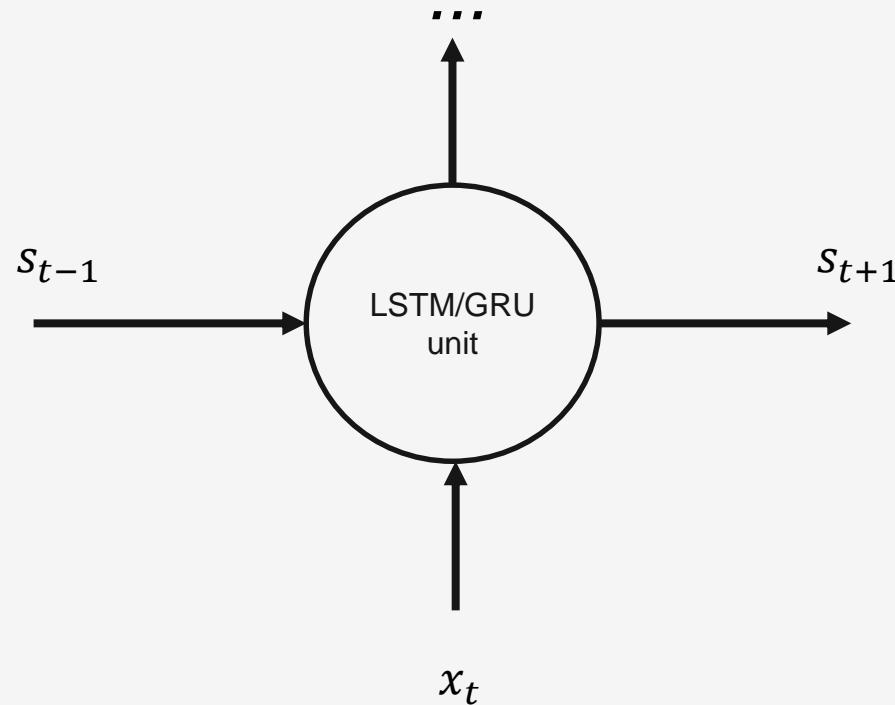
- The number of parameters for LSTM is expressed as:

$$p = 4(CH + H^2 + H)$$

# Complexified RNNs

## LSTM/GRU units

You can also stick to the  
simplest version !!!



# Complexified RNNs

## Sentiment Analysis

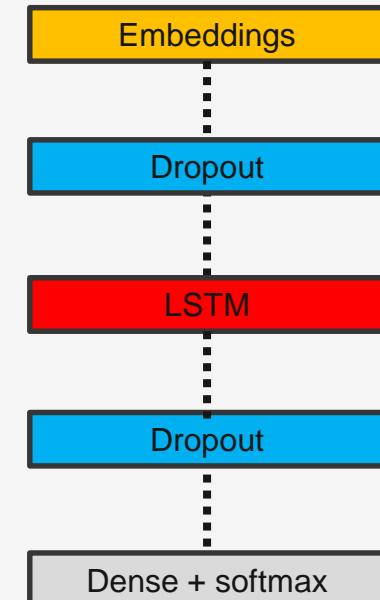
```
from keras.layers import Dense, MaxPooling2D, Conv2D, Flatten

def sentiment_analysis_model():
    model = Sequential()
    model.add(Embedding(vocab_size=2000, 5, input_length=500))
    model.add(Dropout(0.2))
    model.add(LSTM(50))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model

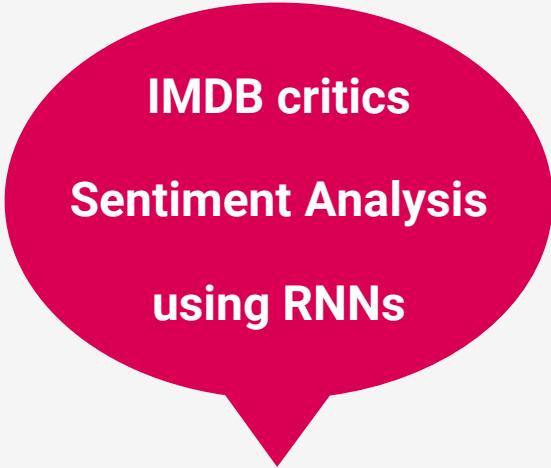
model = sentiment_analysis_model()
model.summary()
```



# Study case n°4

---

Design, train and predict



**IMDB critics**  
**Sentiment Analysis**  
**using RNNs**



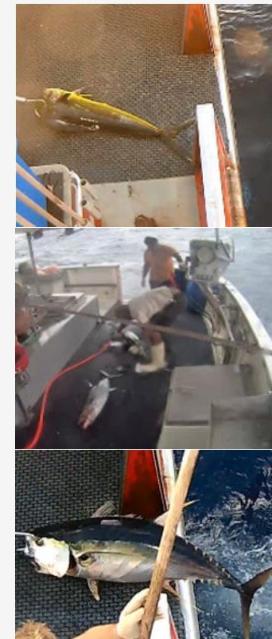


# 4

Practice:  
Design and train  
your own  
network



-  ALB: Albacore tuna (*Thunnus alalunga*)
-  BET: Bigeye tuna (*Thunnus obesus*)
-  DOL: Dolphinfish, Mahi Mahi (*Coryphaena hippurus*)
-  LAG: Opah, Moonfish (*Lampris guttatus*)
-  SHARK: Various: Silky, Shortfin Mako
-  YFT: Yellowfin tuna (*Thunnus albacares*)





5

Conclusion

# Conclusion

---

- Useful in AI
- The most powerful tool in the recent years in the AI environment
- Evolving very fast ... need to constantly update the knowledge

# Tips

---

- Courses online available:
  - <http://course.fast.ai/> , free course given by Jeremy Howard
  - <https://fr.coursera.org/specializations/deep-learning> course given by Andrew NG
- Books:
  - “The Elements of Statistical Learning”, by T. Hastie, R. Tibshirani and J. Friedman
  - “Deep Learning”, by I. Goodfellow, Y. Bengio and A. Courville
  - “Deep Learning with python”, by F. Chollet
- Research papers at: <http://www.arxiv-sanity.com/>

# Thanks

It was a big pleasure to animate this course!  
Sincere thanks!



Thank you

