# eleven - Hackathon

## Deep Learning Basics

To the attention of the Ingénierie Mathématique & Informatique students

September 6th, 2022

École des Ponts ParisTech

eleven
strategy • data • digital

# Artificial neural networks are based on biological neurons because the human brain processes information efficiently

## Biological neurons

- Basis constituent of the nervous system
- Information is propagated by electric signals through dendrites and axons
- A neuron is activated when the input signal is higher than a threshold
- A neuron is equivalent to a Heaviside step function

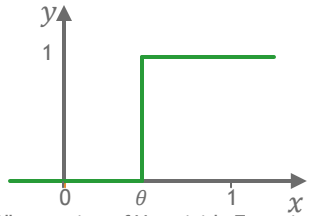


Illustration of Heaviside Function

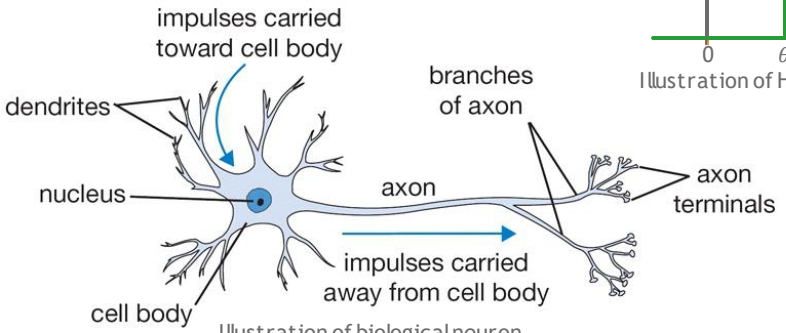
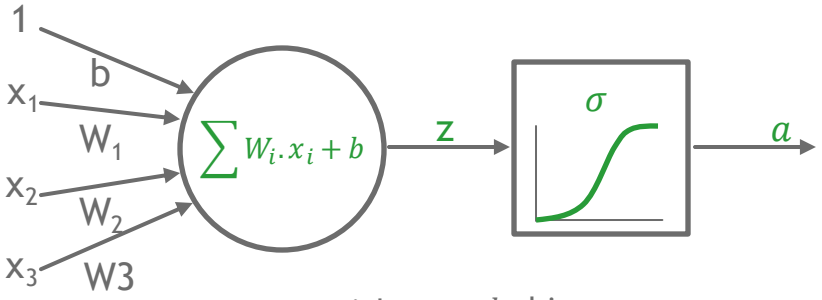


Illustration of biological neuron

## Artificial neuron

- A simplified model of a biological neuron
- An artificial neuron is a linear function, on which an activation function is added
- The step function is not adapted due to its sensitivity to noise
- The common activation function used is the sigmoid function:

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




$W$: weights $b$: biases

$x$: inputs $\sigma$: activation function

**Summary**

- Each neuron processes a bit of information and passes it to its children
- Overall, the network processes raw information into general concepts

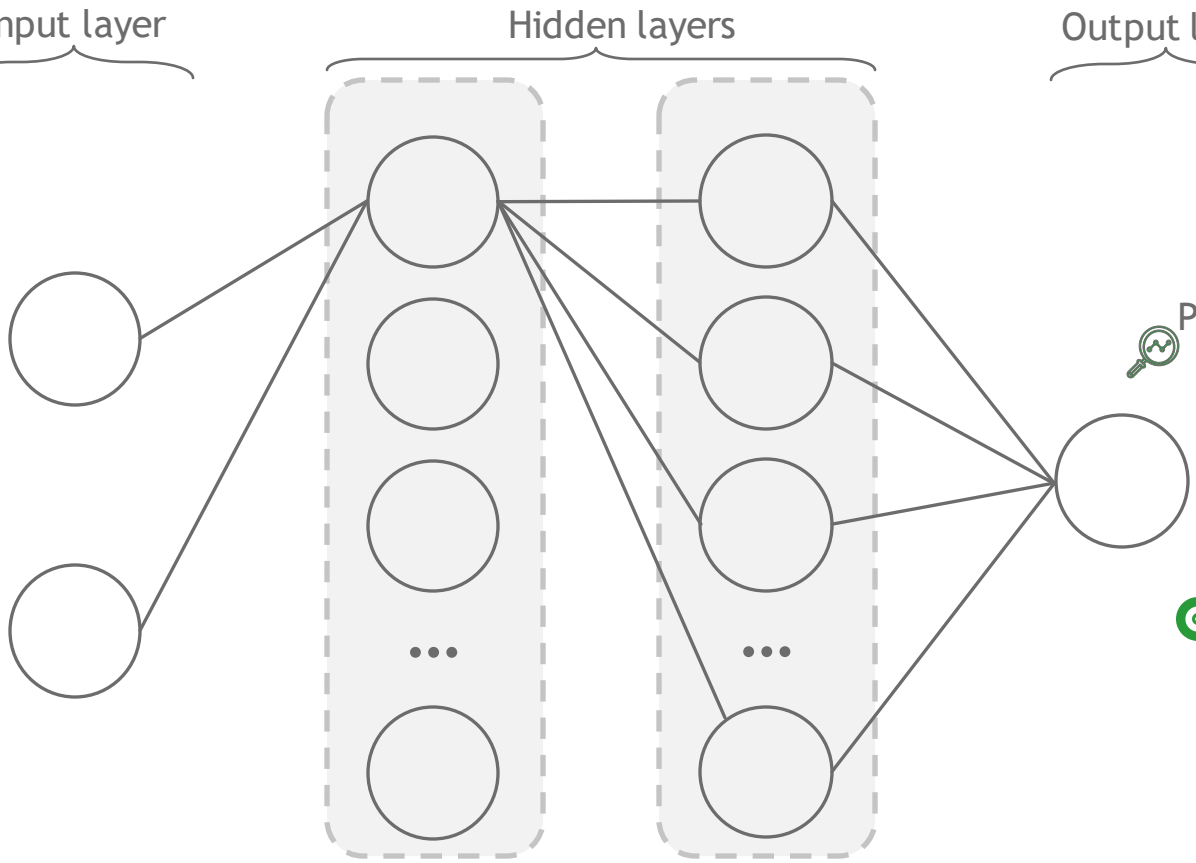
# A neural network learn how to imitate a function from a training dataset

## Computation process: real estate price prediction example

400 m²

Paris

Input features




Predicted value

Real value

1.5M€

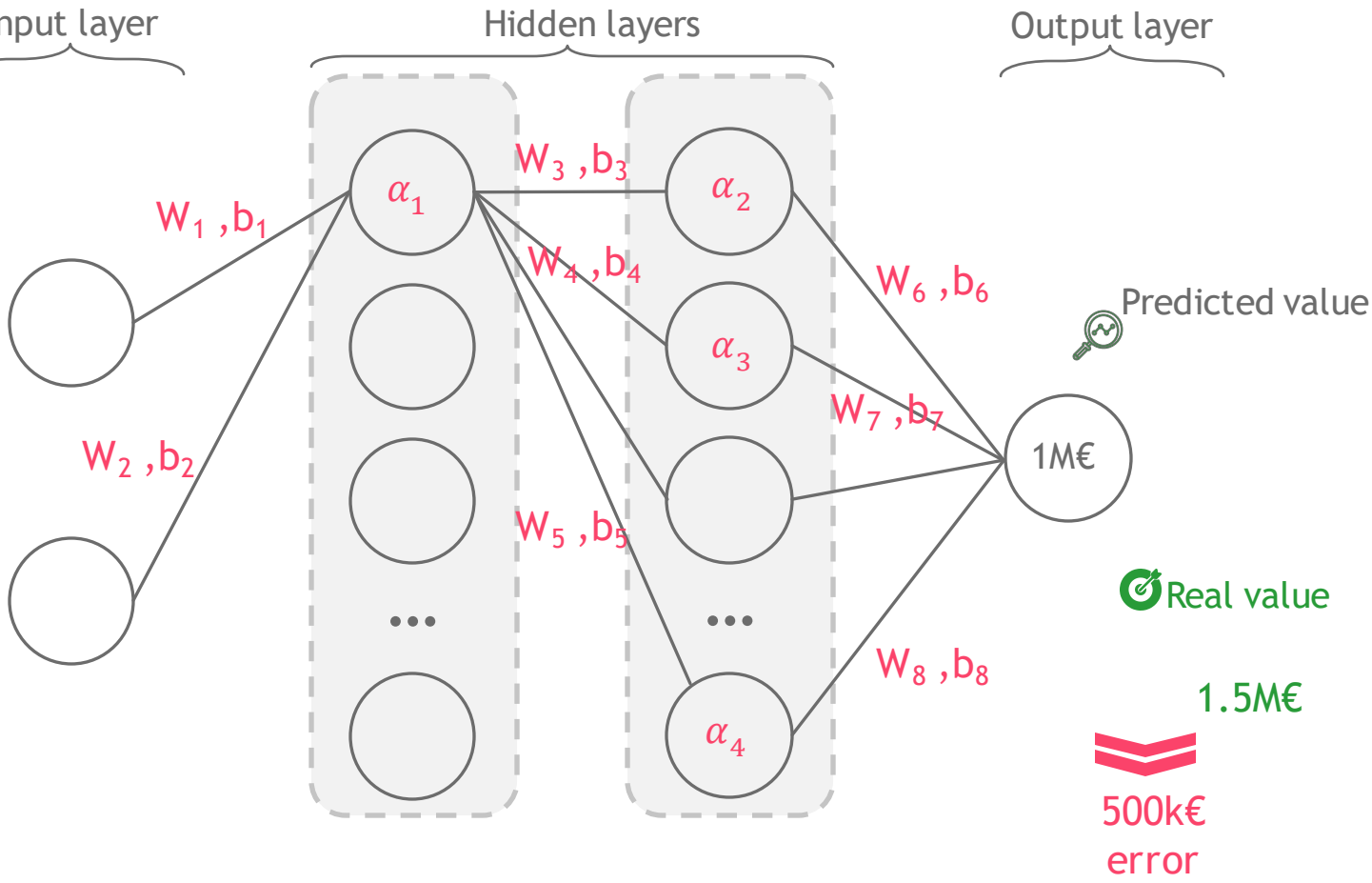
# A neural network learn how to imitate a function from a training dataset

## Computation process: real estate price prediction example

Input features

400 m²

Paris

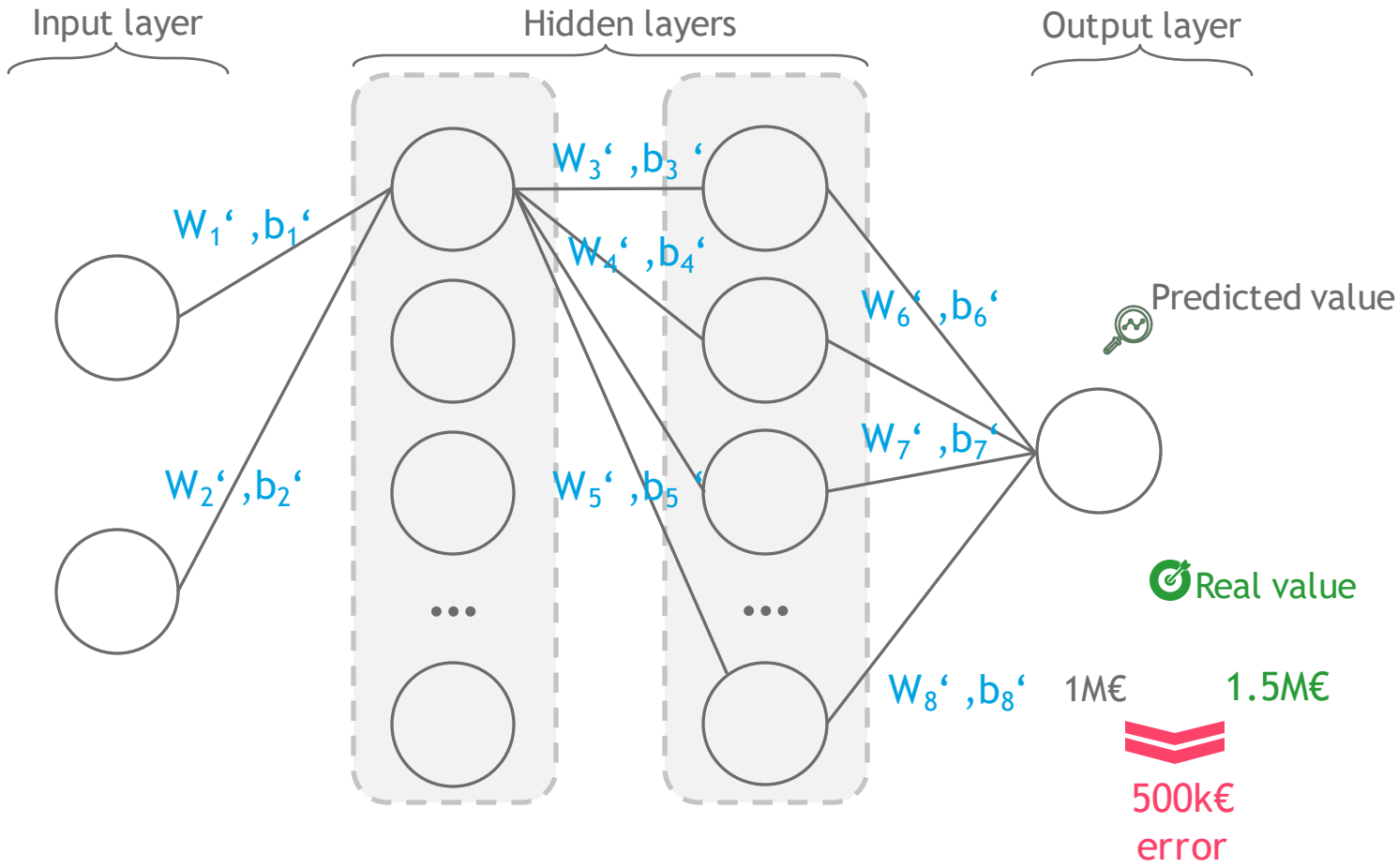



**Note**

- This process is called the forward propagation and allows to compute the loss function of the model
- How can we find the model's optimal parameters for this task ?

# A neural network learn how to imitate a function from a training dataset

## Computation process : real estate price prediction example




**Note**

- This process is called the backward propagation and allows to update the neural network parameters
- Forward and backward propagation are done iteratively until having a high-performance model, by doing a gradient descent to minimize loss

# The gradient descent can be controlled by a hyperparameter of the network: the learning rate

## Overview of the learning rate

- The learning rate $(\alpha)$ controls the speed at which the model learn
- It controls the update of the weights
- If the learning rate is high, the weights will be strongly modified at each epoch
- A technique to speed up the gradient descent is the learning rate decay, which consists to slowly reduce the learning rate over time : allows large weight changes at the start of the learning process, and more fine-tuning towards the end

## Illustration of the learning rate

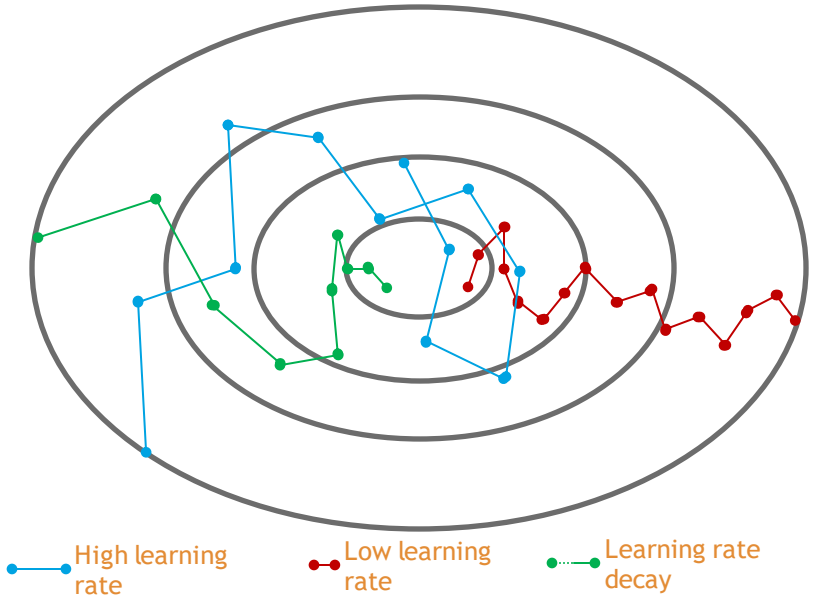



*Illustration of the impact of the learning rate on the gradient descent*

**Note**

- The learning rate is equivalent to the length of the step, and it may be the most important hyperparameter to configure our model, and is generally in [ $10^{-6}$ ,1]
- Adaptive Learning Rates is a learning rate schedule adapting the LR to the performance of the model. Typically, once the performance of the model plateaus, LR can be decreased by a factor of 2

On Python, using an open-source machine learning framework like PyTorch can accelerate the path from research prototyping to production deployment

## Deep Learning computation

PyTorch

- PyTorch is defined as an open-source machine learning library for Python
- It is designed for Deep Learning and used for applications such as NLP, Computer Vison

### Main Features

- Easy Interface:
  - PyTorch offers easy to use API; hence it is considered to be very simple to operate and runs on Python
- Python usage:
  - Pytorch can leverage all the services and functionalities offered by the Python environment
- Computational graphs:
  - PyTorch provides a platform which offers dynamic computational graphs. Thus, a user can change them during runtime

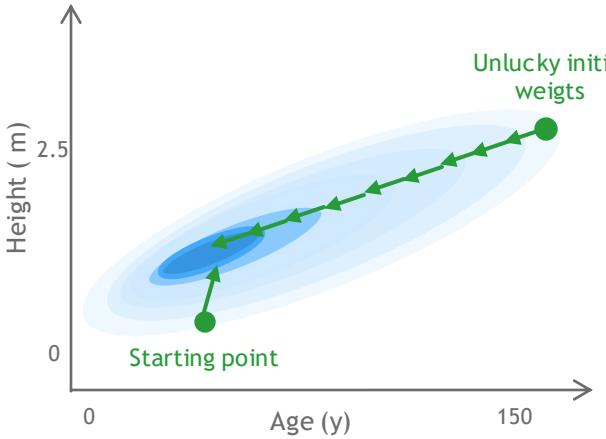
### Key advantages

- Easy to debug
- It includes many layers as Torch
- It includes lot of loss functions
- It can be considered as NumPy extension to GPUs
- It allows building networks whose structure is dependent on computation itself

# Normalisation throughout the neural network helps ease the optimization process

## Why should layers be normalized ?

*Use case of cancer prediction based on Height and Age*

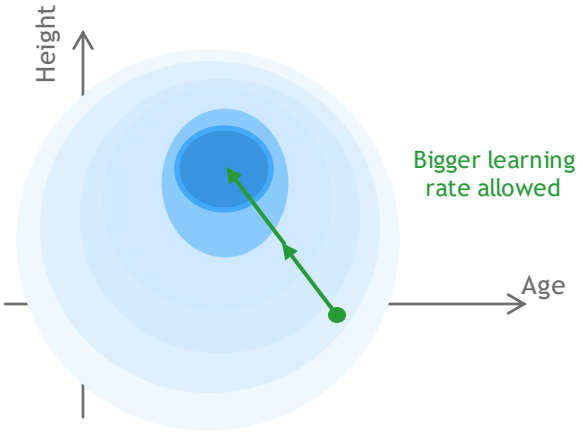



*Loss function of the cancer prediction model before normalization*

- Need of a small learning rate to reach minimum

Slow computation

Importance of initial weights




*Loss function of the cancer prediction model after normalization*

Fast computation

Less importance of initial weights

**Tips**

- Use BatchNorm1d to normalize a layer with PyTorch
- Weight initialization can be “random” by default. Otherwise, it can depend on the type of activation function used : for “Sigmoid” , a xavier initialization method is usually used

*“Xavier” initialization comes from a researcher within Google DeepMind , named Xavier Glorot.*

# Various techniques exist to regularize a neural network and avoid overfitting

## How to prevent Neural Network from overfitting ?

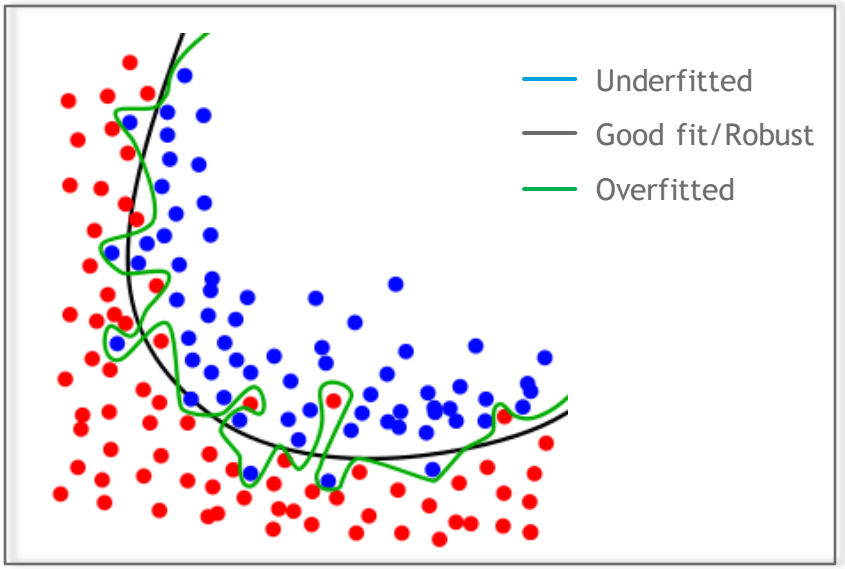



*Illustration of a regression problem and overfitting possibilities*

1. Data augmentation
2. Weights regularization
3. Early Stopping
4. Dropout

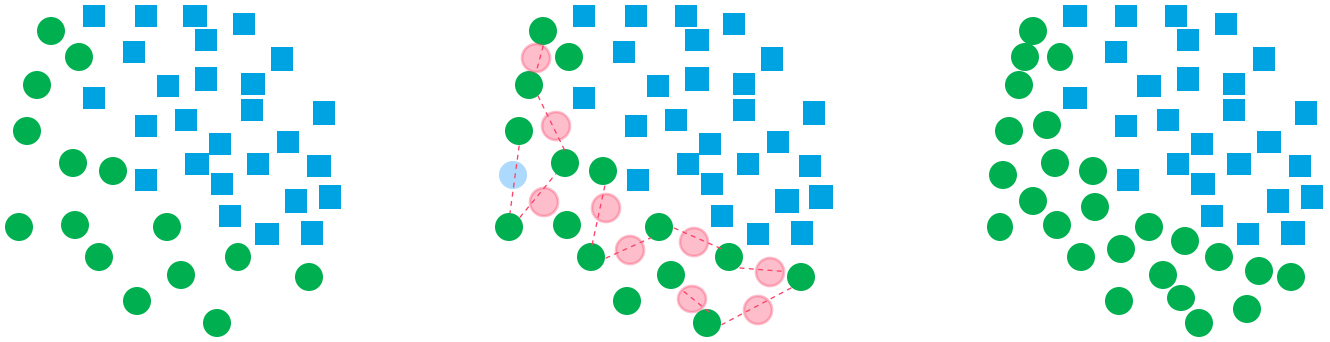
# Various techniques exist to regularize a neural network and avoid overfitting

## How to prevent Neural Network from overfitting ?

1. Data augmentation
2. Weights regularization
3. Early Stopping
4. Dropout

### Illustration

**SMOTE : Syntetic Minority oversampling technique**




- By increasing the size of data and diversity, data augmentation generalize better, hence preventing overfitting

# Various techniques exist to regularize a neural network and avoid overfitting

## How to prevent Neural Network from overfitting ?

Data augmentation

2 Weights regularization

Early Stopping

Dropout

### Illustration

**1 Lasso regularization**

New objective function:

$$\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 + \lambda_1 \underbrace{\sum_{j=0}^{n}|\beta_j|}_{\text{L1 norm}}$$

**2 Ridge regularization**

New objective function:

$$\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 + \lambda_2 \underbrace{\sum_{j=0}^{n}\beta_j^2}_{\text{L2 norm}}$$

ElasticNet data regularization is the combination of both Lasso and Ridge regularization

```
torch.optim.SGD(net.parameters(), lr=0.2, weight_decay=1e-2)
```

# Various techniques exist to regularize a neural network and avoid overfitting

## How to prevent Neural Network from overfitting ?

1 Data augmentation

2 Weights regularization

3 Early Stopping

4 Dropout

### Illustration

- Early stopping stops the training when the validation loss is increasing
- If the validation loss is higher than the current lowest validation lost for a certain number of epochs the training is stopped
- The number of epochs we wait is known as the patience

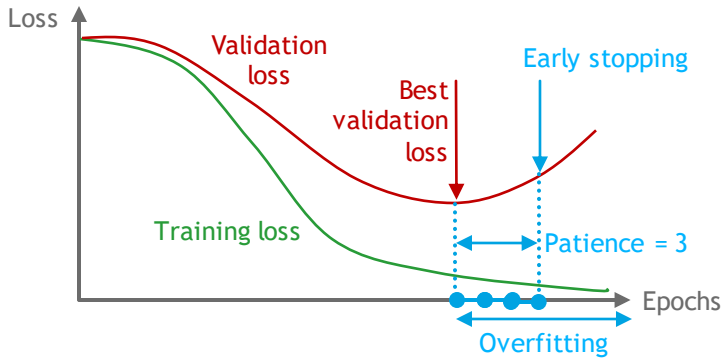



*Illustration of early stopping technique*

# Various techniques exist to regularize a neural network and avoid overfitting

## How to prevent Neural Network from overfitting ?

1 Data augmentation

2 Weights regularization

3 Early Stopping

4 Dropout

### Illustration

- Not training certain weights at a given update step, leading to greater generalization over the whole network
- The classic formulation of this is dropout where neuron activations are “dropped out” (set to zero) with probability p

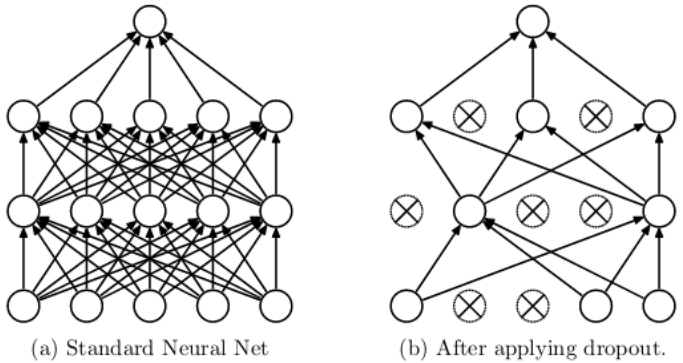



*Illustration of dropout technique*

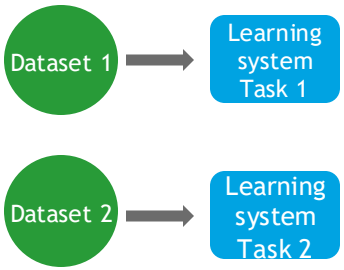
```
torch.nn.Dropout(p=0.5)
```

# Transfer learning uses knowledge acquired for one task to solve related ones and enables to reduce the data required for the related task

## General Overview

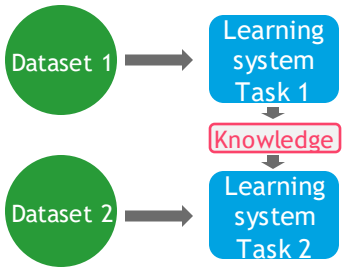
Isolated, single task learning :

- Knowledge is not retained or accumulated
- Learning is performed without considering past learned knowledge in other tasks




Learning of a new task relies on the previous learned tasks :

- Learning process can be faster, more accurate and / or need less training data




There are two variations of transfer learning:

- Same domain but different tasks
- Same task, but different domains

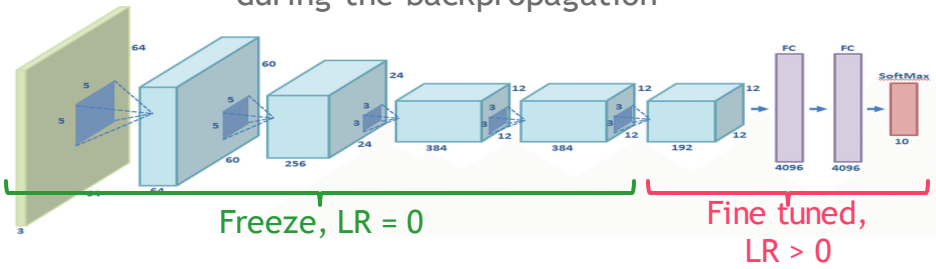
## Transfer learning in Deep Learning

Use pre-trained model and freeze most layers of the neural network

- Freeze layer imply that weights are not updated during the backpropagation, the learning rate (LR) is equal to 0

Apply fine tuning to lasts layers in order to adapt the model to the specific task

- The weights of fine-tuned are updated during the backpropagation



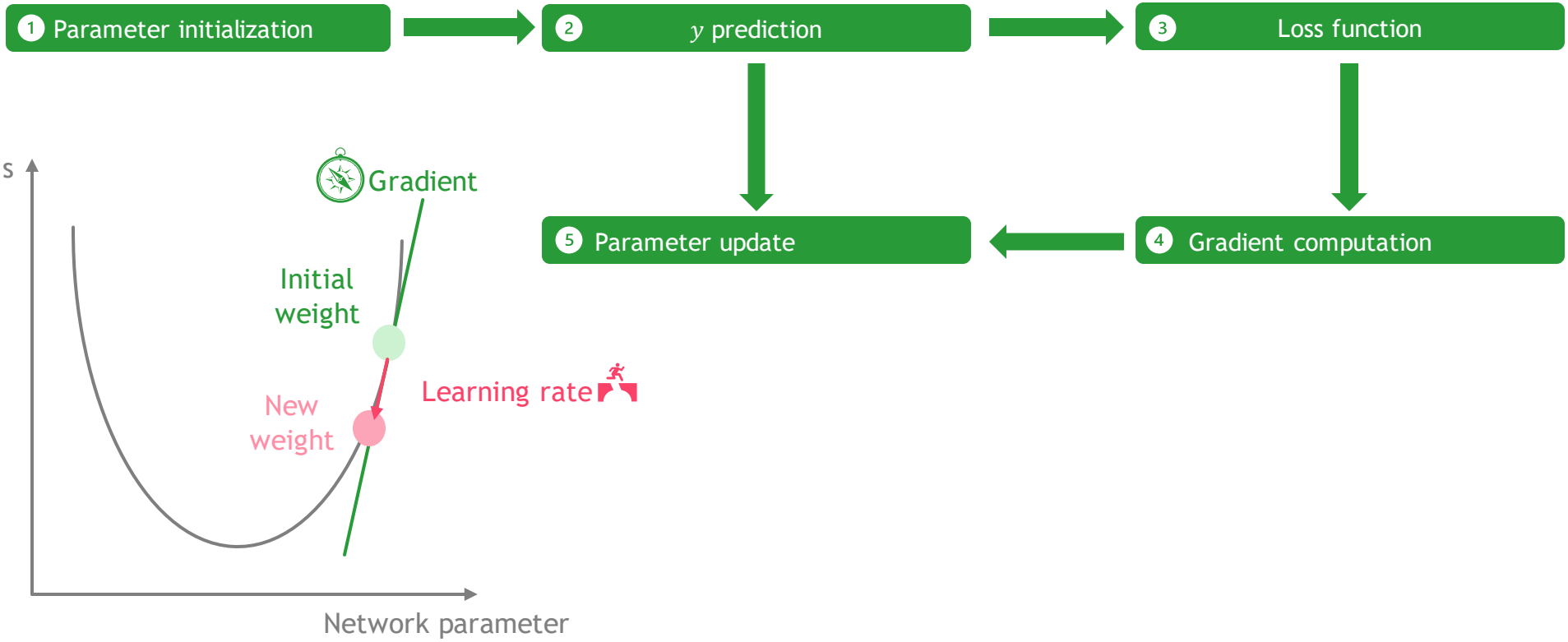

- Keep simple and general features from shallow layers
- Retrain complex features from deep layers

**Tips**

- Each of the PyTorch domain librarie (torchvision , torchtext) comes with pretrained models (torchvision.models)
- HuggingFaces has a series of pretrained models on many different domains (vision, text, audio) and plenty of different datasets (https://huggingface.co/models )

To conclude, Neural networks are based on human brain to create a powerful model that can learn complex functions

## Reminder of what we have learned

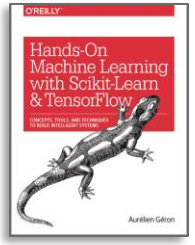


# Some examples of interesting resources to go further

- **In 1h: Machine Learning for everyone** https://vas3k.com/blog/machine_learning/


- **In 4h: Machine learning Toolbox ("learning by doing")**
  https://www.datacamp.com/courses/machine-learning-toolbox

- **In 9 Weeks : Hands-On ML with Scikit-Learn & TensorFlow**
  https://www.oreilly.com/library/view/hands-on-machine-learning/9781491962282/




- **In 11 Weeks: Machine Learning (by Andrew Ng)**
  https://www.coursera.org/learn/machine-learning/home/welcome

- And above all
- https://www.coursera.org/specializations/deep-learning

- **More than 30 Weeks: Deep Learning (Goodfellow et al.)**
  http://www.deeplearningbook.org/