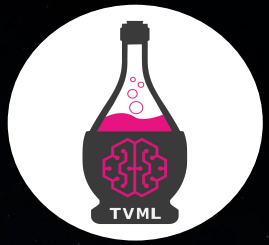
Introduction to Machine Learning with PyTorch



TARALLUCCI, VINO E MACHINE LEARNING
STUDY GROUPS

Cesare Montresor
cesare.montresor@gmail.com

School of AI Verona

Disclaimer

I’m a student, not an expert

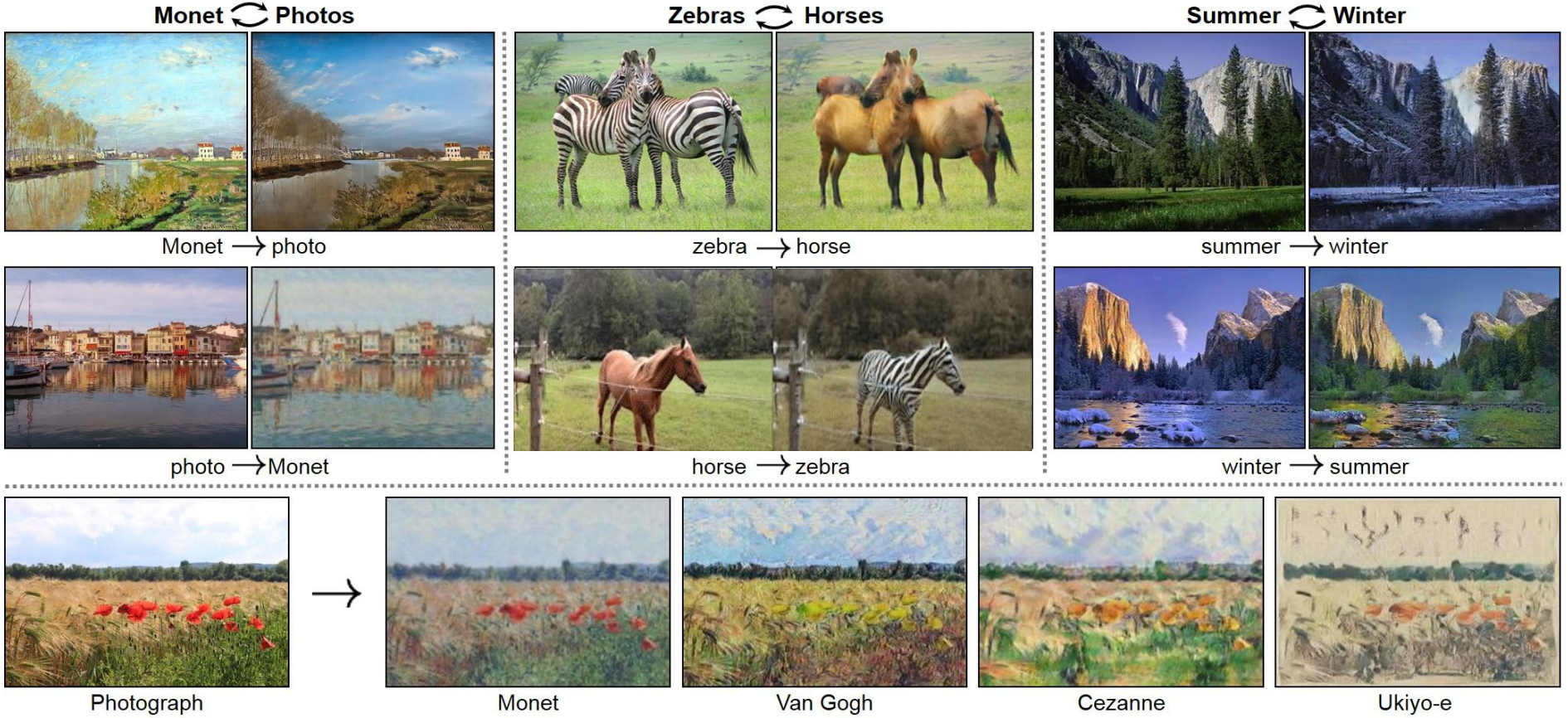
...there will be some errors.

Why ?

- it’s hype.
- Yields interesting results.
- It’s here to stay.
- Solve problems otherwise impossible.
- Open new opportunities.
- Easy to implement O_o.

Idea

Give input and desired output e let the logic write itself (supervised learning).


Monet ⇄ Photos
Zebras ⇄ Horses
Summer ⇄ Winter
Monet → photo
zebra → horse
summer → winter
photo → Monet
horse → zebra
winter → summer
Photograph
Monet
Van Gogh
Cezanne
Ukiyo-e

OpenAI Dexterity

OpenAI Five


900 × 176
D'OH!

Steps

1. Processing the input.
2. Error compared to the desired output.
3. Correct slightly.

Repeat

1. Processing the input: a function

Input (x)	Output (y)
3	9
2	4
7	49
9	81

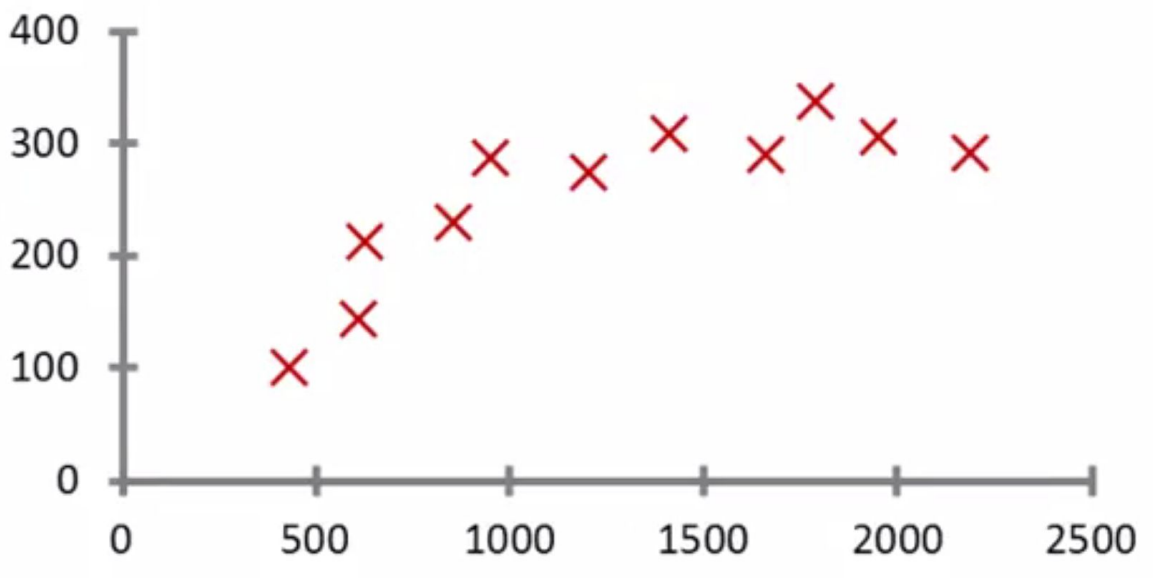
int y = myFunction(x)

y = f(x)

myFunction(x) == f(x) == x^2

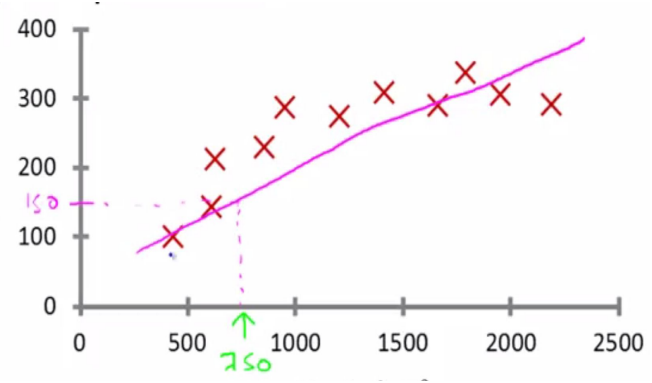
1. Example: real-estate price quotation

Y Price (€)	**X** Area (dm^2)
100.000	450
150.000	600
220.000	620
300.000	1000
...	...
290.000	2250



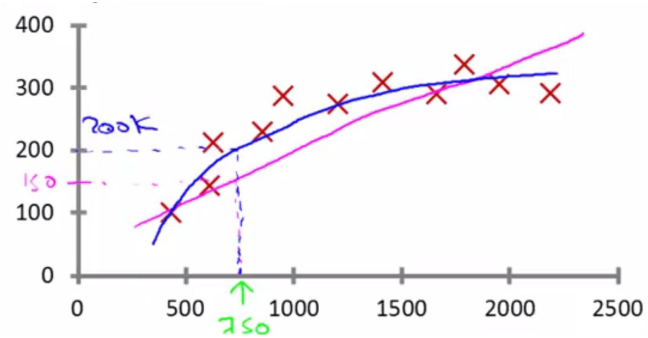
1. Approximation

Line



$y = (x * a) + b$

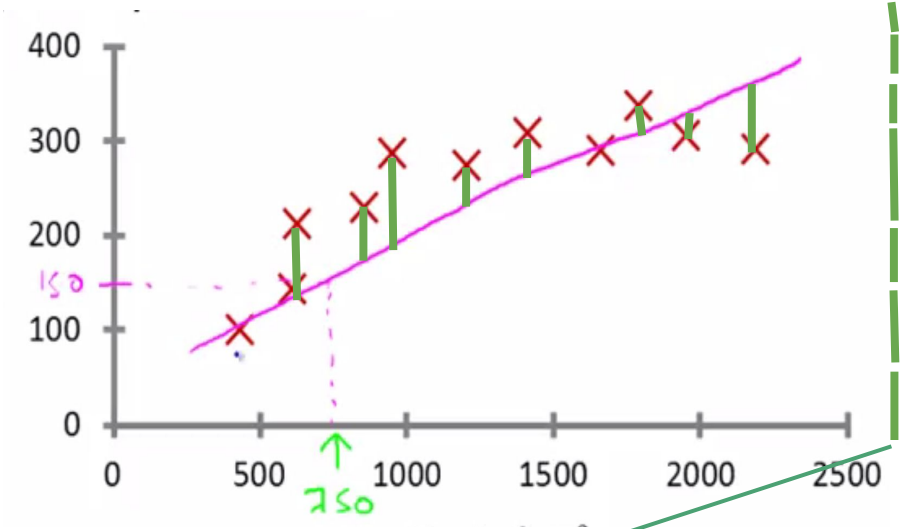
Parabola



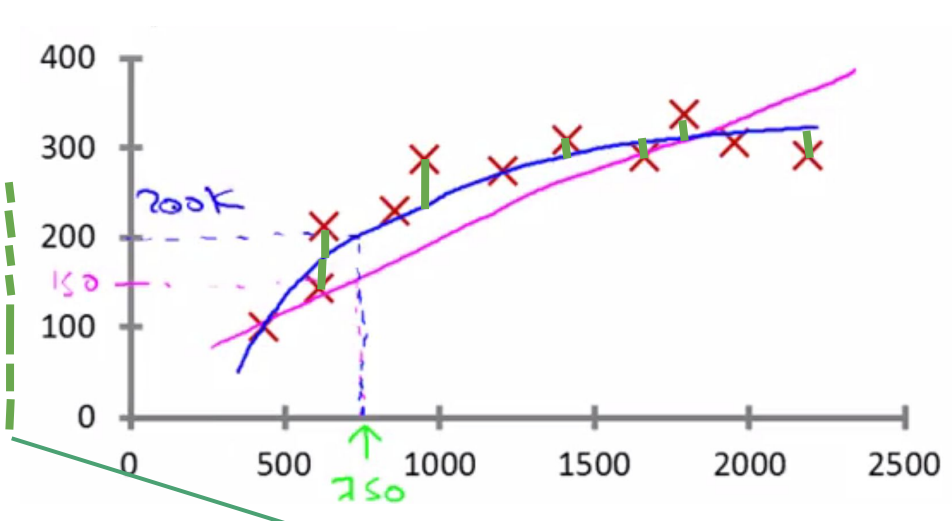
$y = (x^2 * a) + (x * b) + c$

2. Error (loss)

Line



Parabola



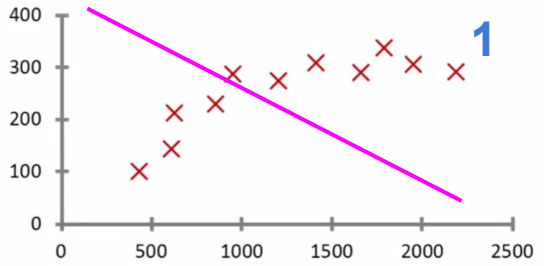
~500

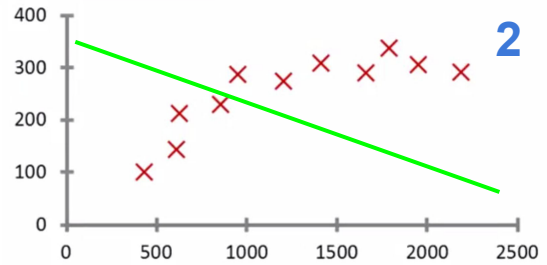
~250

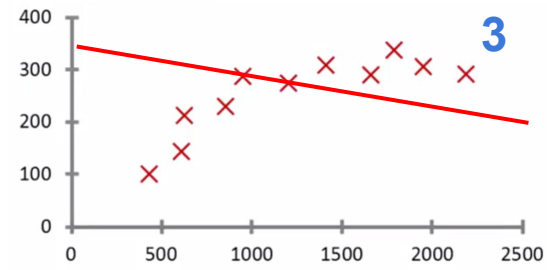
Nella pratica conviene usare MSE

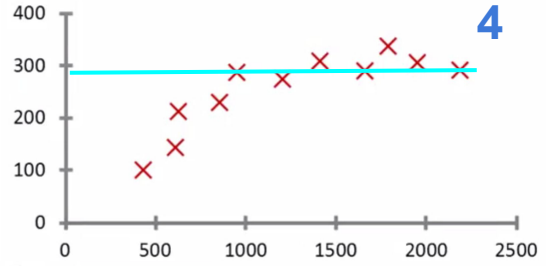
$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

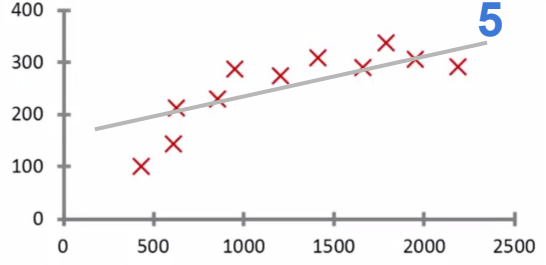
3. Correction (optimize)

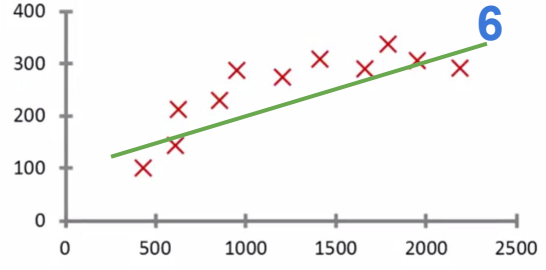




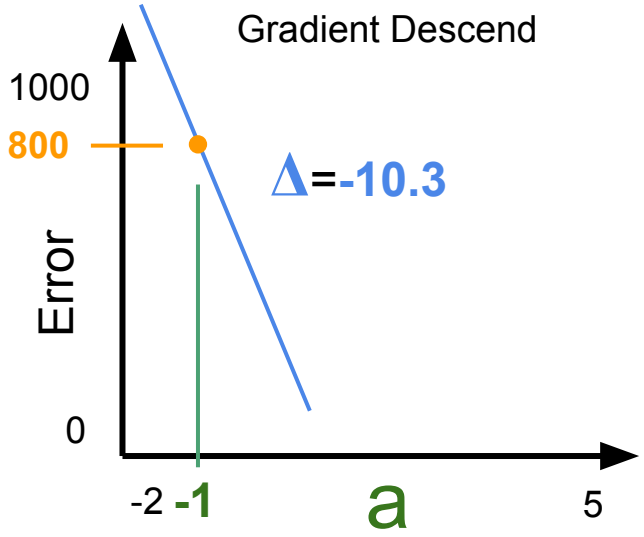








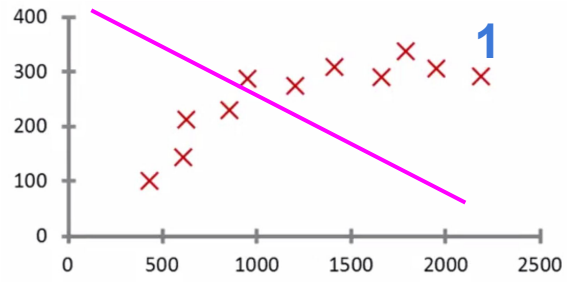
3. Correction (optimize)



Learning Rate (α) = **0.01**

$a = a - \Delta * \alpha$

$a = -1 - (-10.3) * 0.01 = \underline{-0.897}$



$y = (x * a) + b$

Errore = $\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$

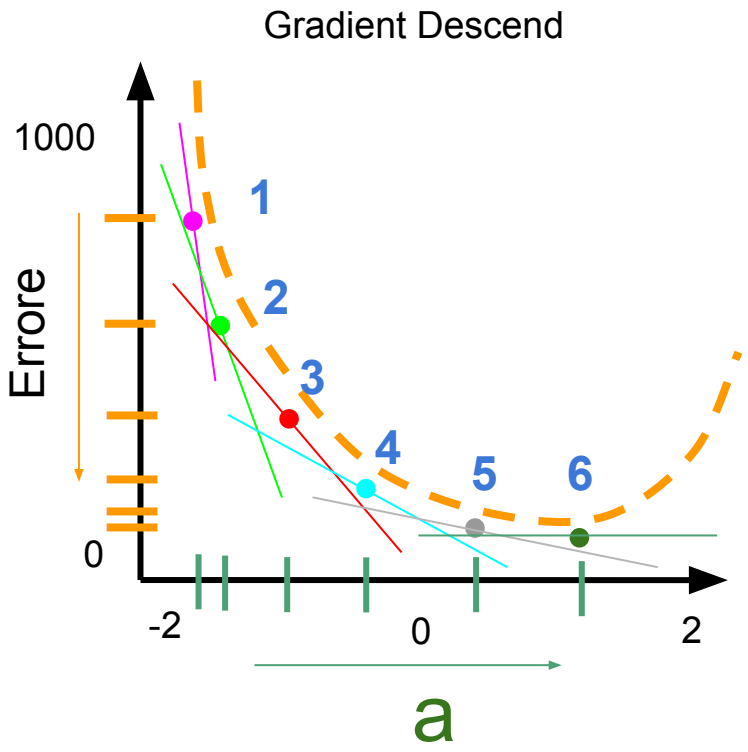
Error (loss):

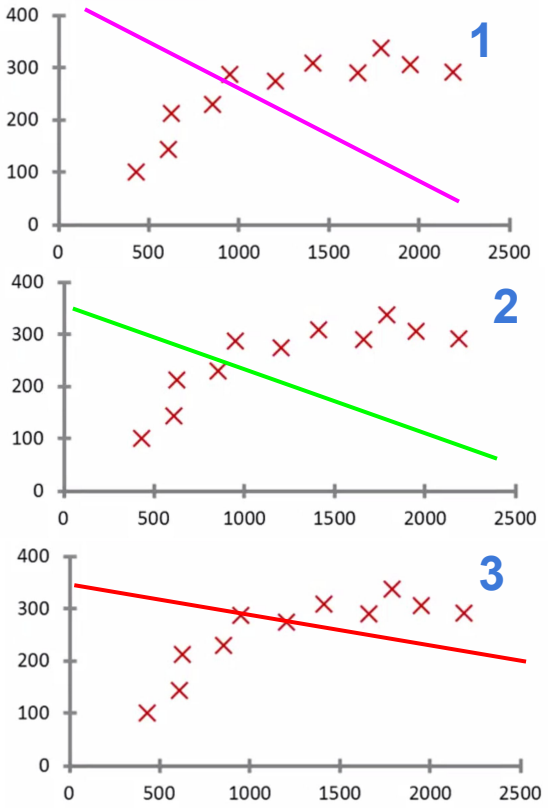
$((x_1*a+b - \hat{y}_1)^2 + \ldots + (x_n*a+b - \hat{y}_n)^2) / n$

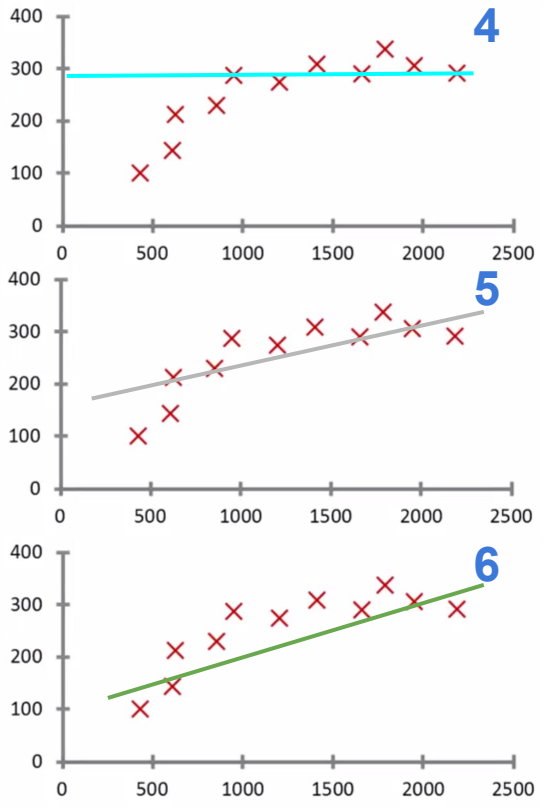
Derivative of the error with respect to a (Δ) "gradient"

$(2*x_1*(x_1*a+b - \hat{y}_1) + \ldots + 2*x_n*(x_n*a+b - \hat{y}_n)) / n$

3. Correction (optimize)







```python
from torch.utils.data import Dataset
import pandas as pd

class CsvDataset(Dataset):
    def __init__(self, file_path, transform=None):
        self.data = pd.read_csv(file_path)
        self.transform = transform

def __getitem__(self, index):
    house_price = self.data.iloc[index, 0]  # y
    house_info = self.data.iloc[index, 1:]  # X
    if self.transform is not None:
        house_info = self.transform(house_info)
    return house_info, house_price

def __len__(self):
    return len(self.data)
```

```python
from torchvision import transforms
from torch.utils.data import DataLoader

my_dataset = CsvDataset( 'data.csv' )

data_loader = DataLoader(
    my_dataset,
    batch_size = 8,
    shuffle = True
)

x,y = next(iter(data_loader))
```

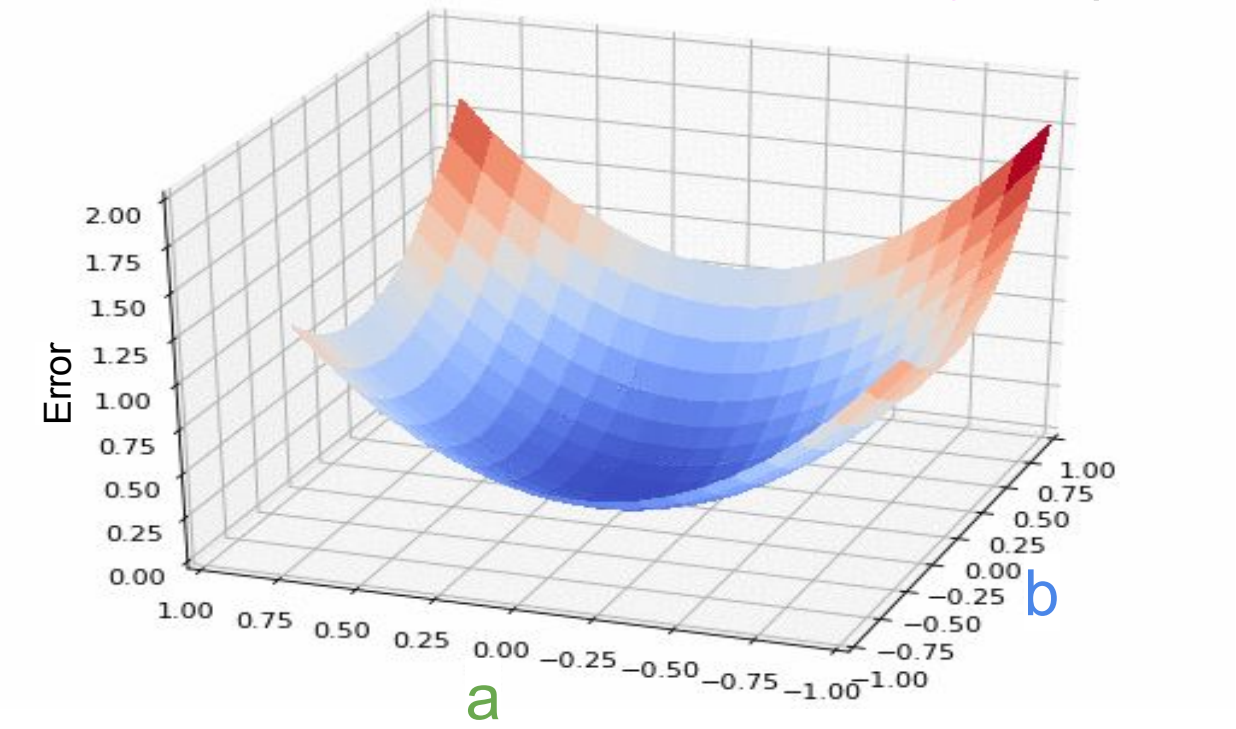
```
import torch
import torch.nn.functional as F

x = torch.from_numpy(  np.random.randn(10)  )
y = torch.from_numpy(  np.random.randn(10)  )

a = torch.randn(1, requires_grad=True)
b = torch.randn(1, requires_grad=True)

y_pred = x*a+b

error = ((y_pred - y) ** 2).mean()
# Oppure, più semplicemente
error = F.mse_loss(y, y_pred)
```

```
learning_rate = 1e-3

# x: house_info, y: house_price
for x, y in train_loader:
    y_pred = x*a+b
    # calcolo dell'errore
    error = ((y_pred - y) ** 2).mean()
    # gradient usando la derivata dell'errore
    error.backward()
    # ottimizzo a e b
    a.data -= learning_rate * a.grad.data
    b.data -= learning_rate * b.grad.data
    # reset gradient a 0
    a.grad.data.zero_()
    b.grad.data.zero_()
```

```
import torch.nn.functional as F
from torch.optim import SGD
from torch import nn

learning_rate = 1e-3
optimizer = SGD( [a, b], lr=learning_rate)
model = nn.Linear(1,1)
for x, y in train_loader:
    y_pred = model(x)
    # calcolo dell'errore
    error = F.mse_loss(y_pred, y)
    # gradient usando la derivata dell'errore
    error.backward()
    # ottimizzo a e b
    optimizer.step()
    # reset gradient a 0
    optimizer.zero_grad()
```

For 2 parameters

$$y = (x * a) + b$$



More parameters

Price (y)	Year (x_1)	Area (x_2)	Var N (x_N)
300.000	1995	120	...
120.000	2001	83	...
49.000	1973	50	...
400.000	2011	200	...
...	...	...	...

int y = myFunction(x_1,x_2);

$y = f(x_1,x_2)$

Artificial parameters:

$x_3 = sqrt(x_1^2+x_2^2)$

$y = f(x_1,x_2,...,x_N)$

How many paramenters ?

Size: ~200 Kb
Width: 1600 px
Height: 900 px
1 px = 255, 255, 255

1600 × 900 x 3 =

4.320.000

(more than four milions)

⬇

GPU

$$y = f(x_1, x_2, \ldots, x_{4.320.000})$$

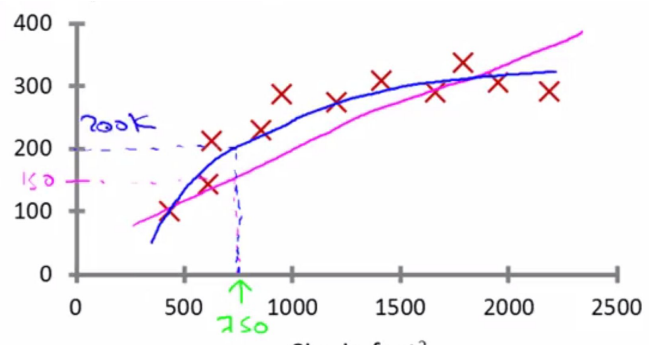
```
import torch

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

a = torch.Tensor((2,3), device=device)
model = torch.nn.Linear(10,10).to(device)
```

```
gpu_conut = torch.cuda.device_count()

gpu0 = torch.device("cuda:0")
gpu1 = torch.device("cuda:1")
…
cpu0 = torch.device("cpu:0")
cpu1 = torch.device("cpu:1")
```

Categories of algorithms

- Regression / Classification
- Supervised / Unsupervised

Regression

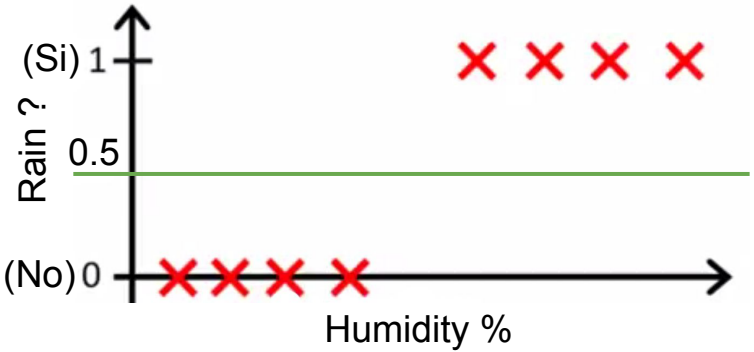


Obtain a continuous output.

Examples:

- Value of a good
- Probability of success.
- Control of an actuator.
- Age from a picture.

Classification

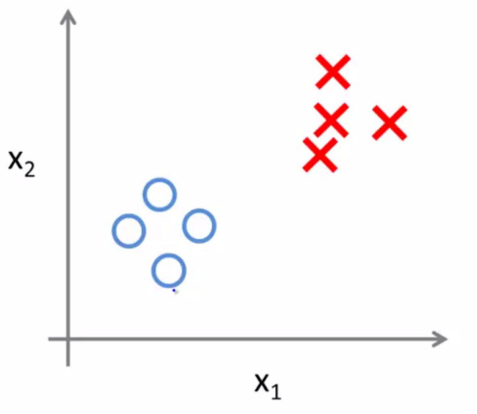


To determine a class/category

Examples:

- Kind of object, from picture
- Weather forecast.
- Gender from picture.
- Sentimente from text.

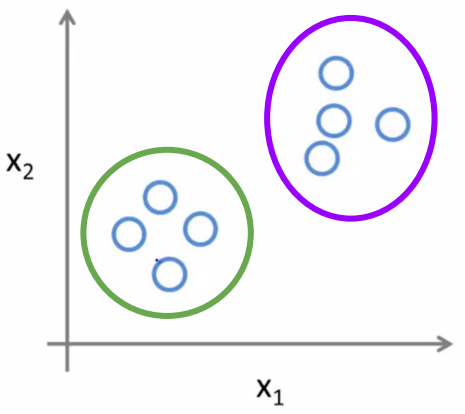
Supervised



Provied input and desired output.

- Linear Regression
- SVM
- Decision tree
- Neural Networks

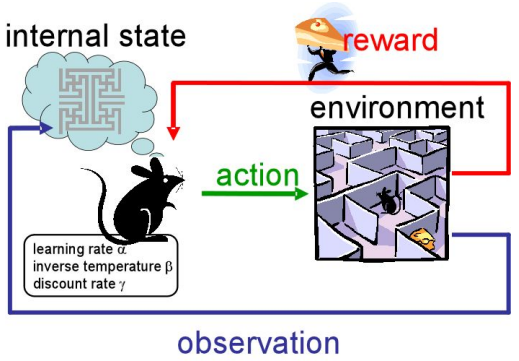
Unsupervised



Providing input and identify group/structure in data.

- K-means, dbscan
- PCA, SVD
- Anomaly detection
- Neural Networks

Reinforcement



Developing strategies based on actions and rewards.

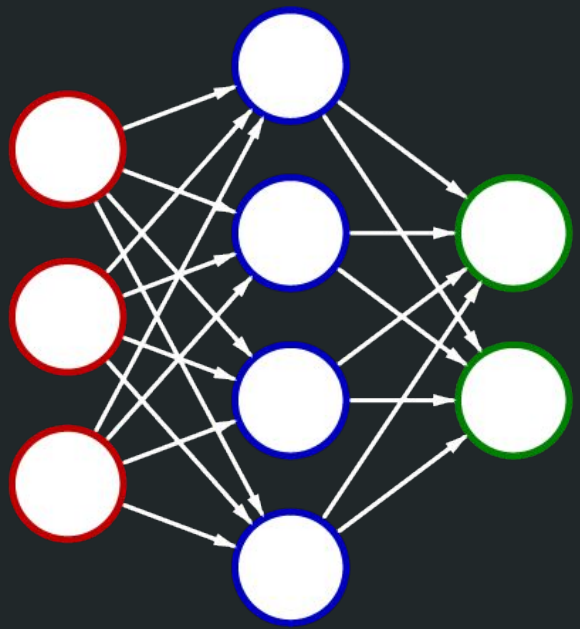
- Monte-Carlo method
- Q-Learning
- A3C, A2C
- Proximal Policy Optimization

Neural Networks

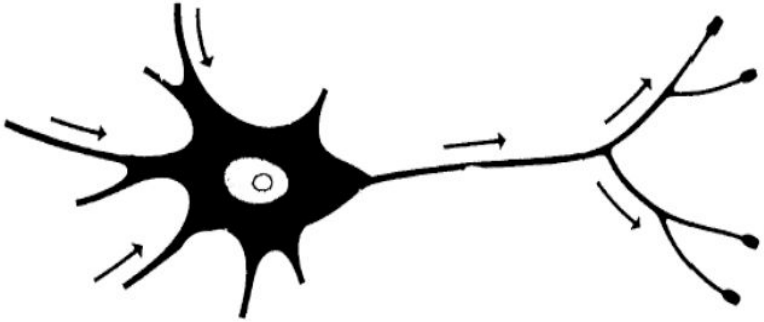
(Differentiable Universal Function Approximators)

(Differentiable Computational Graphs)

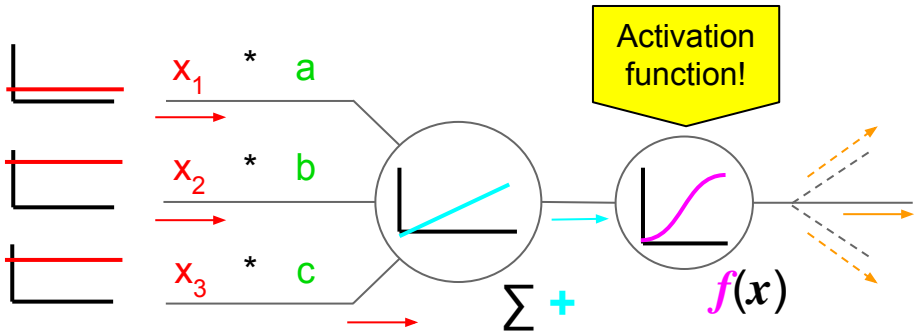
(≈deep learning)



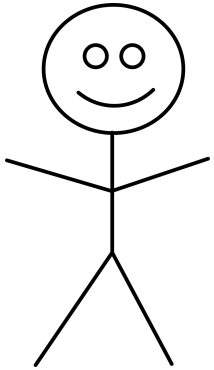
“Neuron”



=



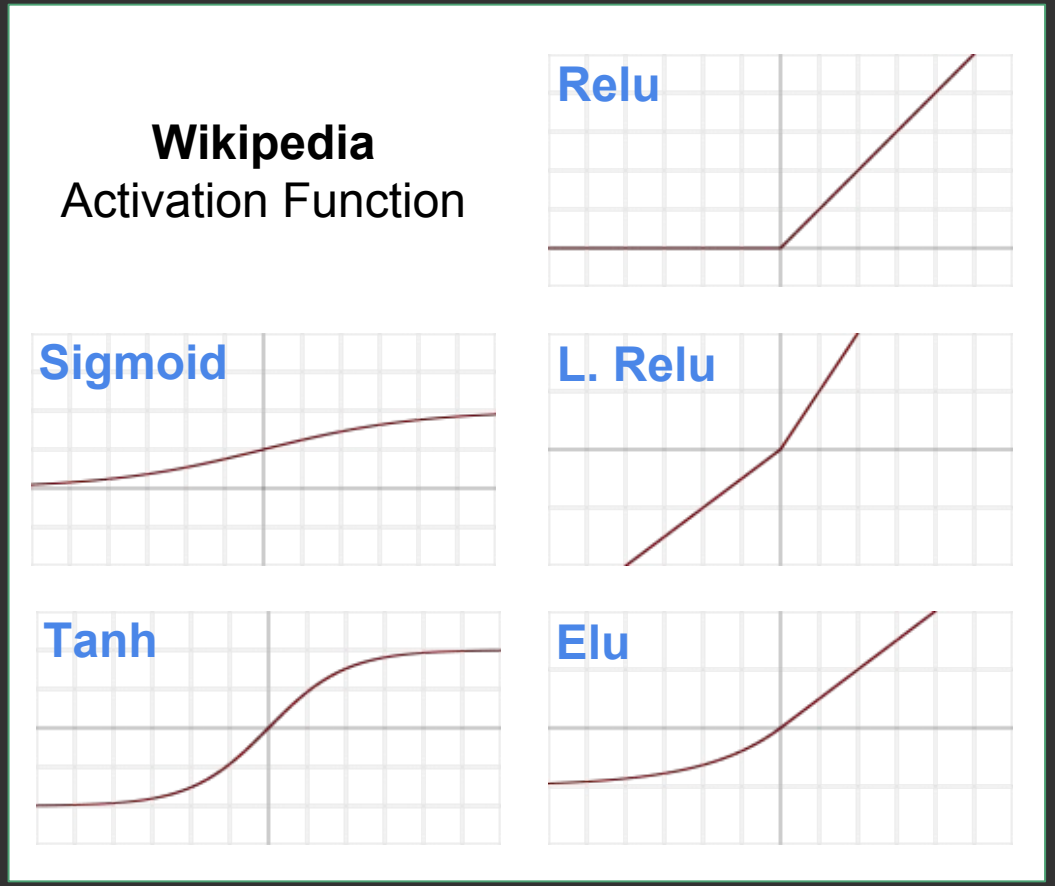
=



$$y = f(x_1 * a + x_2 * b + x_3 * c)$$

```
import torch.nn.functional as F

a = torch.rand(4, requires_grad=True)

F.relu(a)          # [0, ∞)
F.sigmoid(a)       # (0,1)
F.tanh(a)          # (-1,1)
F.softmax(a)       # exp probability
F.leaky_relu(a)
F.elu(a)
```

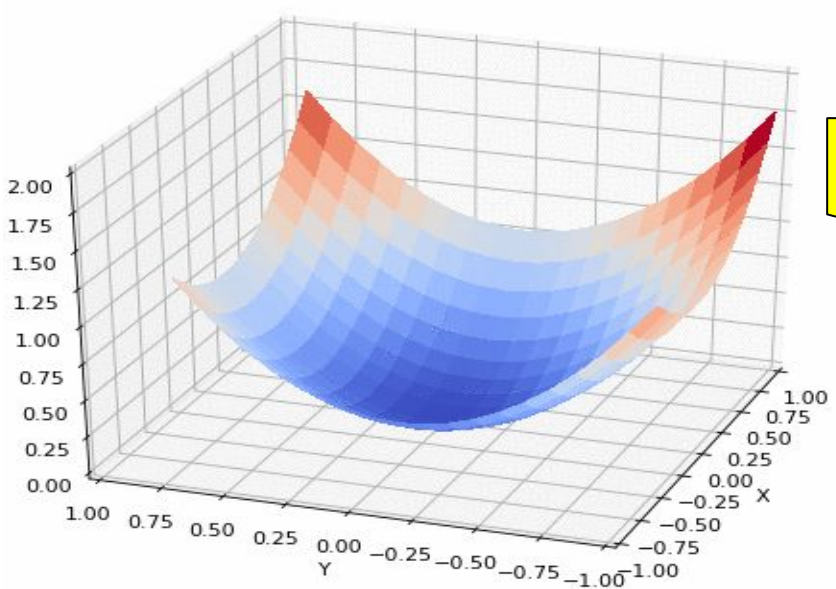
Wikipedia
Activation Function



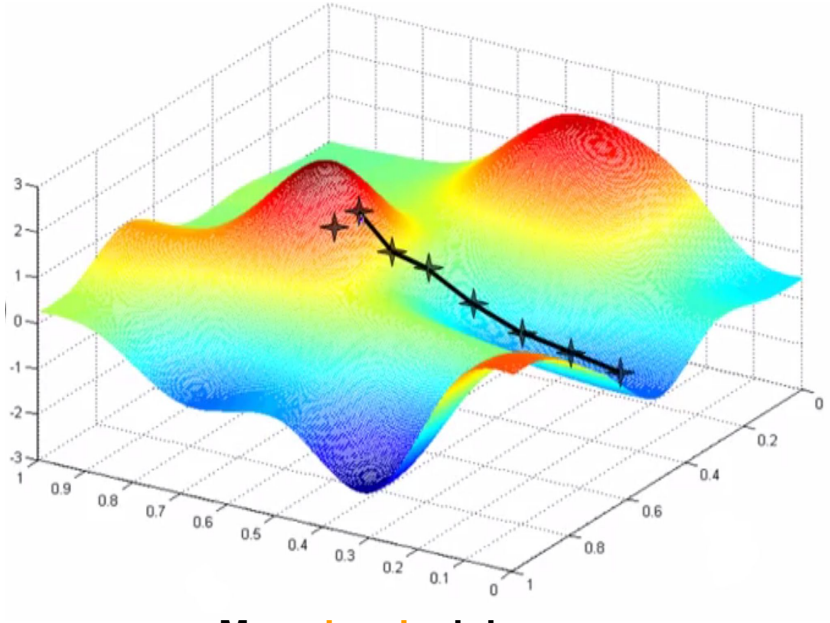
https://pytorch.org/docs/stable/nn.html#non-linear-activation-functions

Optimizing non-linear functions

Improves the representation capabilities of the model, but makes non-deterministic the optimization process.



Activation function!



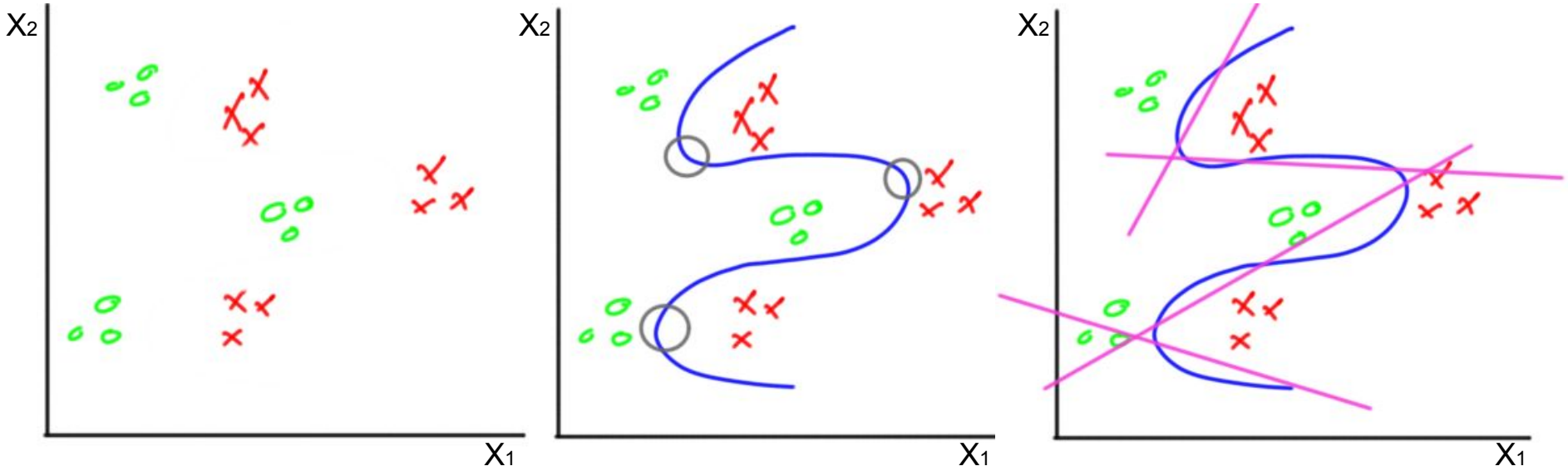
One global minima
Guaranteed to obtain the optimal solution

Many local minima
No guaranteed that any solution is the most optimal

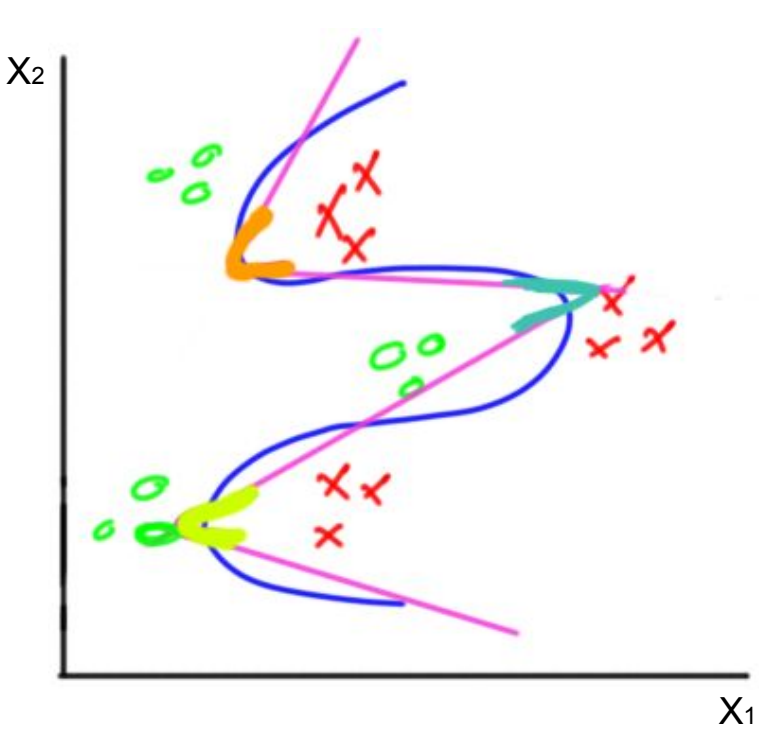
http://neuralnetworksanddeeplearning.com/chap4.html

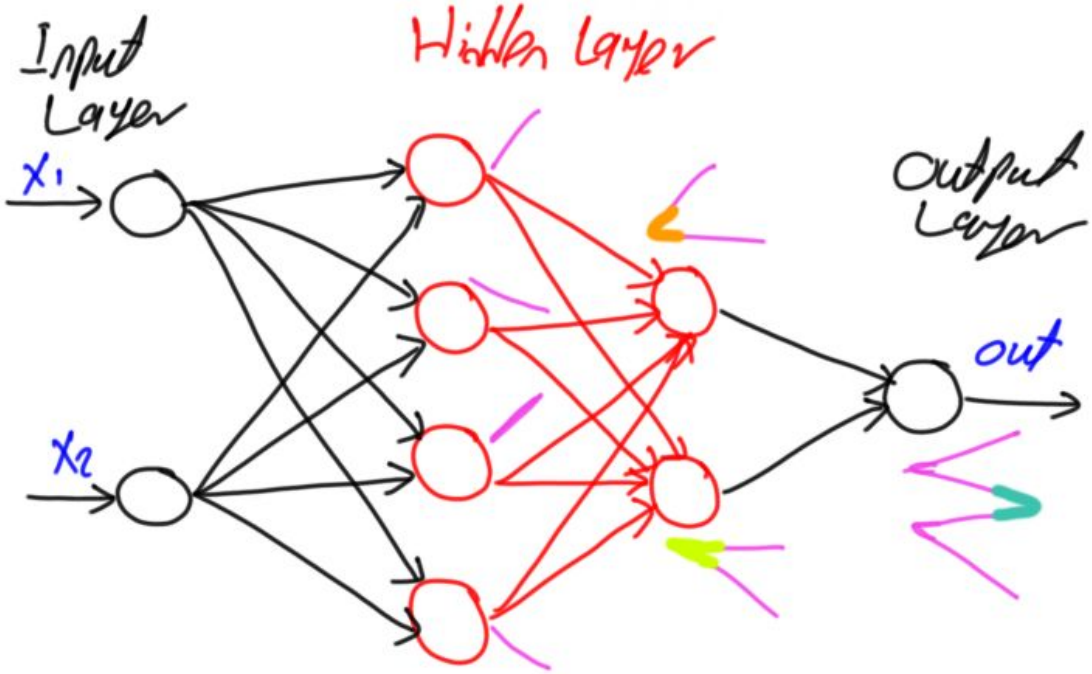
Hierarchy of neurons

Problem: Identify issues in a engine.
X_1 = Vibrations
X_2 = Temperature



Hierarchy of neurons

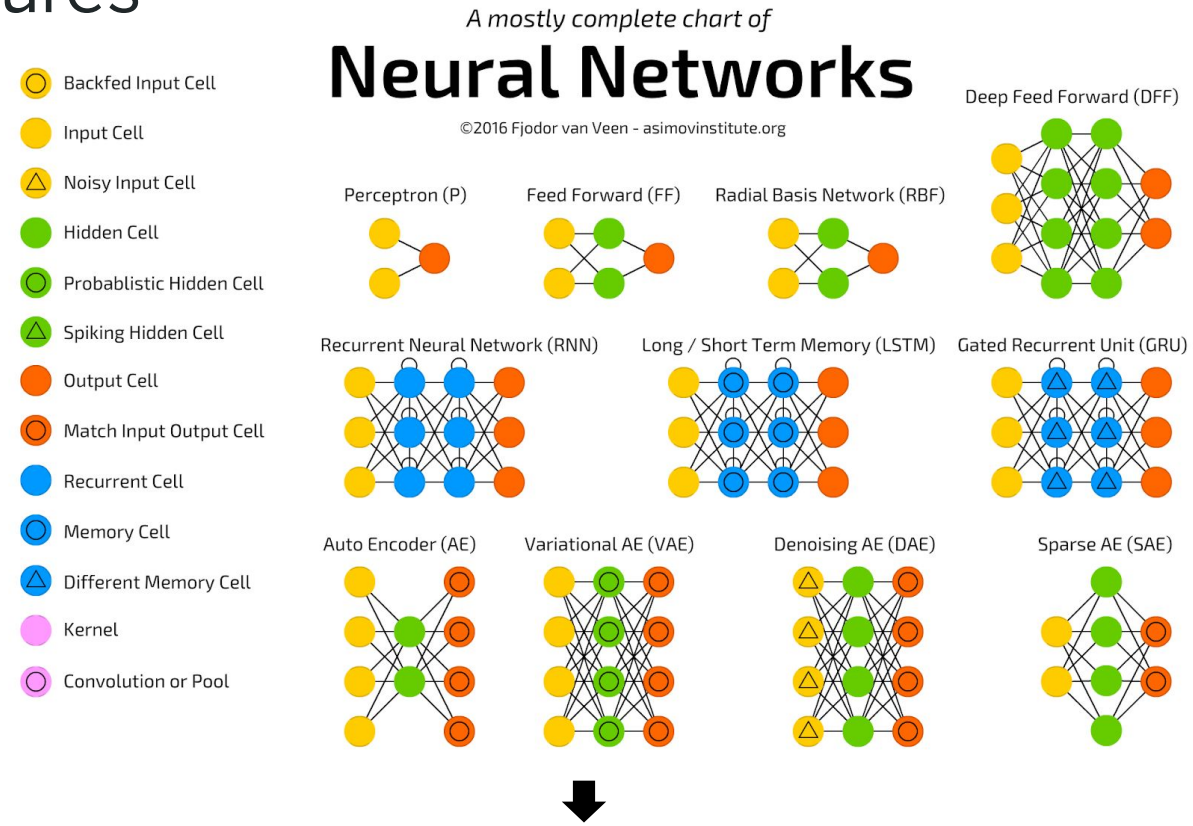


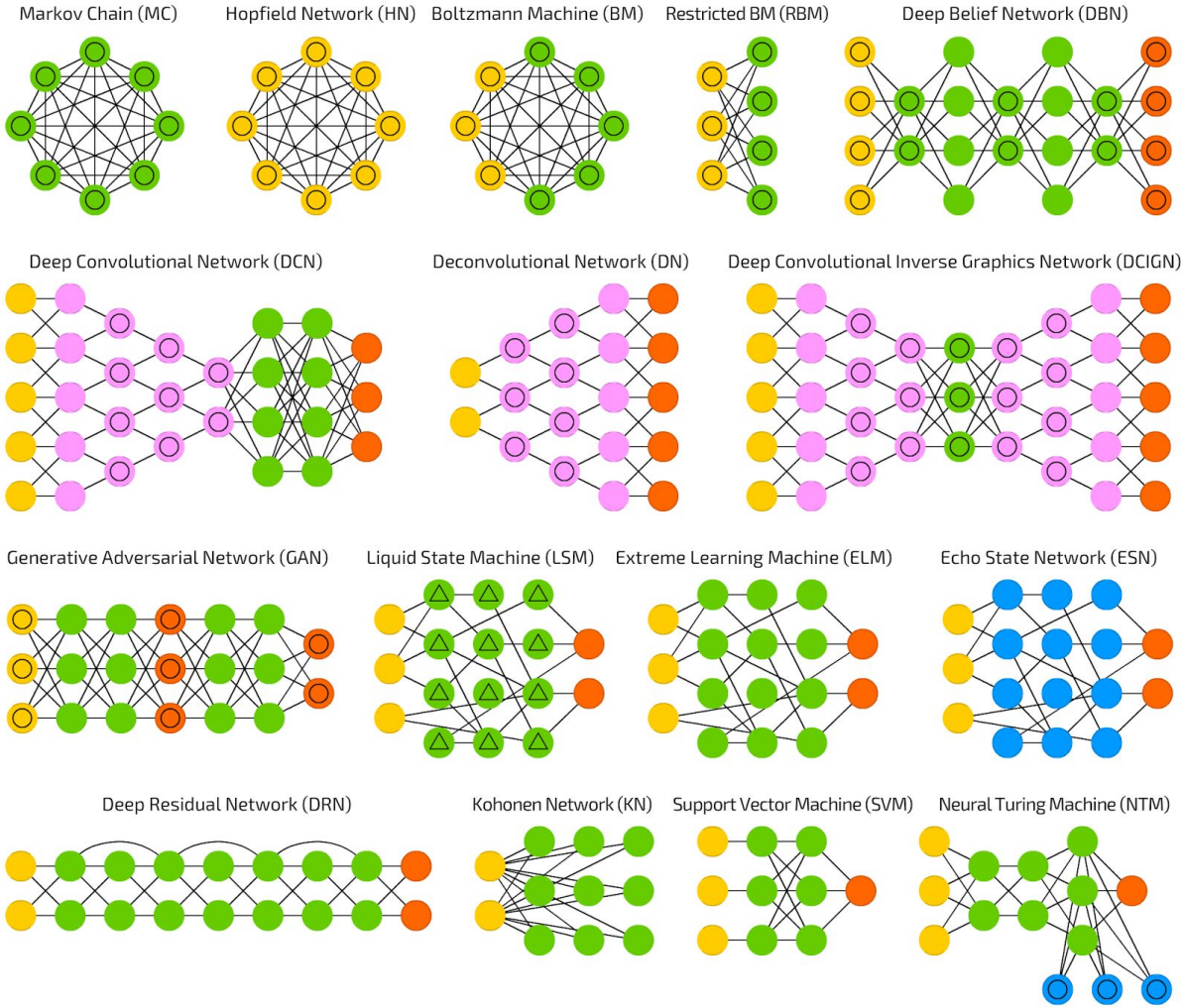


```
from torch import nn

model = nn.Sequential(
   nn.Linear(2, 4),
   nn.ReLU(),
   nn.Linear(4, 2),
   nn.ReLU(),
   nn.Linear(2, 1),
   nn.Sigmoid()
)
```

```
from torch import nn

class NetworkFC(nn.Module):
   def __init__(self):
      super().__init__()
      self.input = nn.Linear(2, 4)
      self.hidden = nn.Linear(4, 2)
      self.output = nn.Linear(2, 1)
   def forward(self, x):
      x1 = F.relu( self.input(x) )
      x2 = F.relu( self.hidden(x1) )
      x3 = F.sigmoid( self.output(x2) )
      return x3

model = NetworkFC()
```

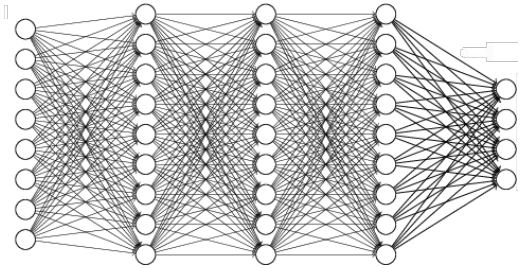
Architectures




Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

NN

“Fully connected”

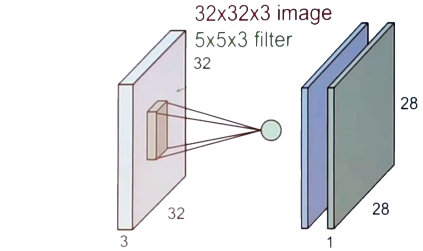


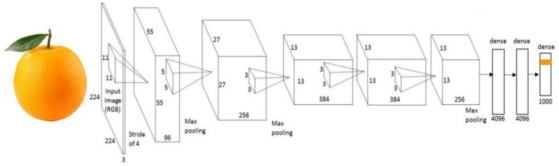
Every neuron is connected to every other neuron of the layers before and after

Also know as Linear, Dense, “fully connected” or “fc”.

CNN

“Convolutional Neural Network”

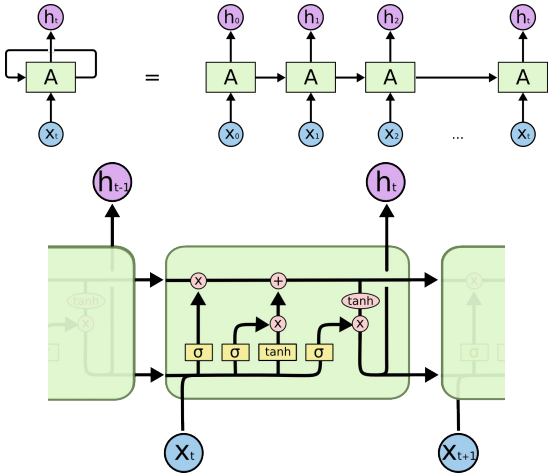




Preserves the spatial information. Useful in parsine 2D/3D images, but also text.

RNN

“Recurrent Neural Networks”



Store an internal state while parsine sequential information. Useful in parsing time series, music, text, etc

```
from torch import nn
```

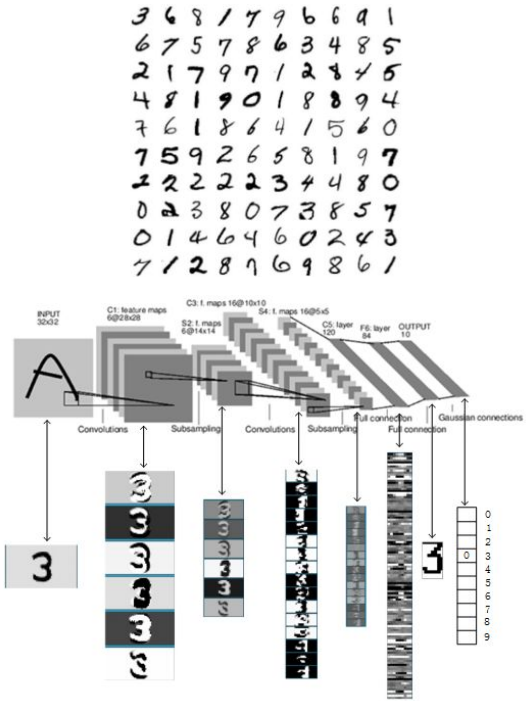
```
class torch.nn.Linear( in_features, out_features, bias=True)
model = nn.Linear( 2, 4)
```

```
class torch.nn.Conv1d( in_channels, out_channels, kernel_size, ... )
class torch.nn.Conv2d( in_channels, out_channels, kernel_size, ... )
class torch.nn.Conv3d( in_channels, out_channels, kernel_size, ... )
model = nn.Conv2d( 3, 20, 3)
```

```
class torch.nn.RNNCell(input_size, hidden_size, … )
class torch.nn.LSTMCell(input_size, hidden_size, … )
class torch.nn.GRUCell(input_size, hidden_size, … )
class torch.nn.RNN(input_size, hidden_size, num_layers, ... )

model = nn.RNN(10, 20, 2)
```

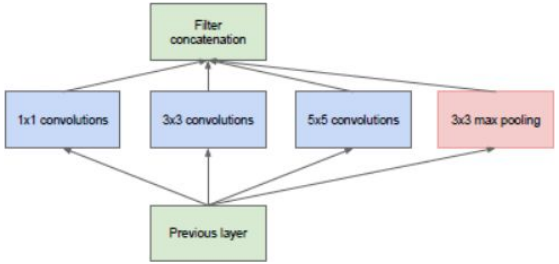
LeNet

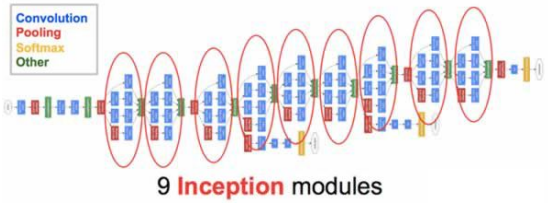
“Funziona!”



GoogLeNet

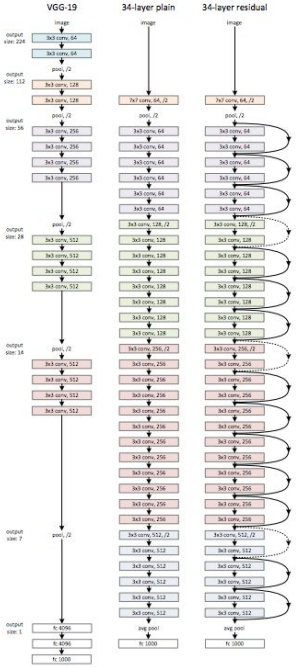
“Network nel network”





ResNet

“Up to 101!”



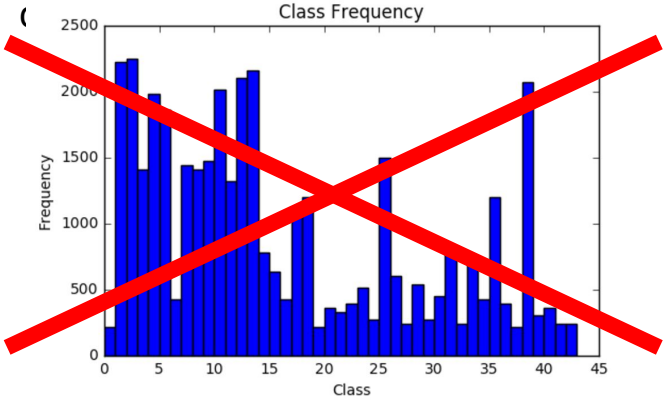
```
import torchvision.models as models

# AlexNet
model = models.alexnet()
# returns a model pre-trained on ImageNet
models.alexnet(pretrained=True)

# Inception v3 / GoogleNet
models.inception_v3( ... )

# SqueezeNet
models.squeezenet1_0( ... )
models.squeezenet1_1( ... )

# DenseNet
models.densenet121( ... )
models.densenet169( ... )
models.densenet161( ... )
models.densenet201( ... )

# VGG
models.vgg11( ... )
models.vgg11_bn( ... )
models.vgg13( ... )
models.vgg13_bn( ... )
models.vgg16( ... )
models.vgg16_bn( ... )
models.vgg19( ... )
models.vgg19_bn( ... )

# ResNet
models.resnet18( ... )
models.resnet34( ... )
models.resnet50( ... )
models.resnet101( ... )
models.resnet152( ... )
```

... Dataset !!!

Often forgotten (1 slide ^_^), in favour of the architectures, the Dataset is of **vital importance**.
It **profoundly** impact the quality of the learning and the final result.

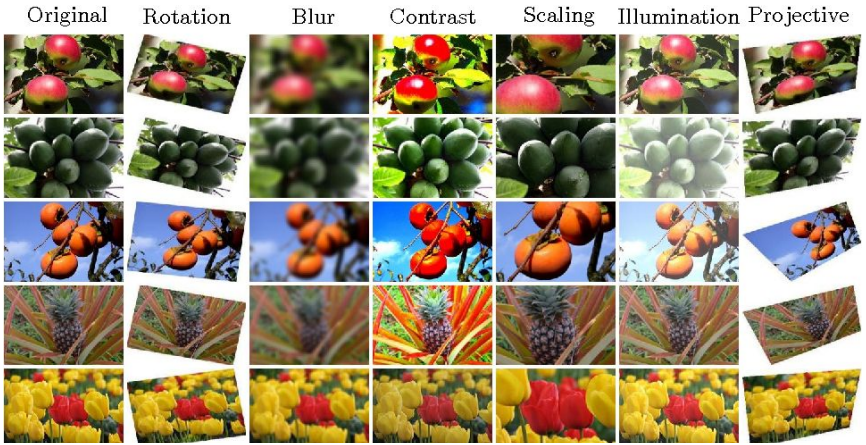
Wide, Varied and Balanced

- hundreds, thousents, millions ? it depends...
- Cats of every size, positions, colors, ages, dimentions, distance, etc...
- Equal numbers of elements. (N gatti ~= N (



Augmenting

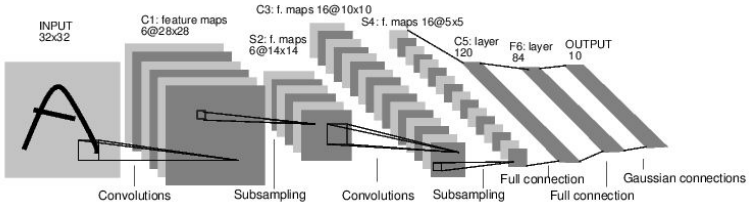
To extend the dataset with copies of the original data with added noise and distortions.



```
import torchvision.datasets as datasets
from torch.utils.data import DataLoader

mnist_ds = datasets.MNIST(root, train=True, download=True)
mnist_loader = DataLoader( mnist_ds, batch_size = 8, shuffle = True )
Images, labels = next(iter(data_loader))

# Other datasets
datasets.FashionMNIST( … )
datasets.MNIST( ... )
datasets.CocoCaptions( ... )
datasets.LSUN( ... )
datasets.CIFAR10( ... )
datasets.STL10( ... )

# Generic wrappers
datasets.ImageFolder(root, ... )
datasets.DatasetFolder(root, ... )
```

```
from torchvision import transforms
from torch.utils.data import DataLoader

tranform_list = transforms.Compose([
   transforms.ToTensor(),
   transforms.Normalize( mean=(0.5,), std=(0.5,) ),
   transforms.RandomHorizontalFlip(),
   transforms.RandomVerticalFlip(),
   transforms.RandomRotation( (-15, 15) ),
   transforms.RandomApply([
      transforms.ColorJitter(brightness=0.3)
   ])
])

mnist_ds = datasets.MNIST(root='.', train=True, transform=tranform_list, download=True)
mnist_loader = DataLoader( mnist_ds, batch_size = 64, shuffle = True )
images, labels = next(iter(data_loader))
```

Questions ?




```
import torch
from torch import nn
from torch import optim
import torch.nn.functional as F
from torchvision import datasets, transforms

trainset = datasets.MNIST('.', download=True, transform=transforms.ToTensor())
trainloader = torch.utils.data.DataLoader(trainset, batch_size=256, shuffle=True)

model = nn.Sequential(
  nn.Conv2d(1, 20, kernel_size=5 ), nn.ReLU(),
  nn.Conv2d(20, 40, kernel_size=3, stride=2), nn.ReLU(),
  nn.Conv2d(40, 80, kernel_size=3, stride=2), nn.ReLU(),
  nn.Conv2d(80, 160, kernel_size=3, stride=2), nn.ReLU(),
  nn.AdaptiveMaxPool2d(1),
  nn.Linear(160, 10 ),
  nn.Softmax(dim=1)
)

optimizer = torch.optim.Adam(model.parameters(),lr=0.001)

for epoch in range(10):
  for images, labels in trainloader:
    optimizer.zero_grad()
    output = model.forward(images)
    loss = F.mse_loss(output, labels)
    loss.backward()
    optimizer.step()

    loss = loss.item()
    print(f"\r Loss: {loss}                    ", end="")
```

About

Cesare Montresor

- cesare.montresor@gmail.com
- https://twitter.com/CesareMontresor
- https://linkedin.com/in/cmontresor

School of AI

- https://www.theschool.ai/

School of AI Verona

Tarallucci, Vino e Machine Learning

- https://www.facebook.com/groups/TarallucciVinoMachineLearningVerona/
- https://www.meetup.com/Tarallucci-Vino-Machine-Learning/

TARALLUCCI, VINO E MACHINE LEARNING
GRUPPI DI STUDIO

Italian Association for Machine Learning

- https://www.iaml.it/
- https://www.facebook.com/machinelearningitalia/

iaml
ITALIAN ASSOCIATION FOR MACHINE LEARNING

Credits

images:

- https://www.coursera.org/learn/machine-learning
- http://cs231n.stanford.edu
- https://github.com/junyanz/CycleGAN
- http://colah.github.io/posts/2015-08-Understanding-LSTMs
- https://www.slideshare.net/xavigiro/deep-convnets-for-global-recognition
- https://www.quora.com/How-does-the-Inception-module-work-in-GoogLeNet-deep-architecture
- https://www.quora.com/How-does-deep-residual-learning-work
- https://www.google.it/imghp
- http://www.asimovinstitute.org/neural-network-zoo
- https://www.linkedin.com/pulse/beginners-ask-how-many-hidden-layersneurons-use-artificial-ahmed-gad/