Deep learning project

Multilayer perceptrons - convolutional neural networks
self organazing maps - reinforcement learning

Abdoulaye KOROKO

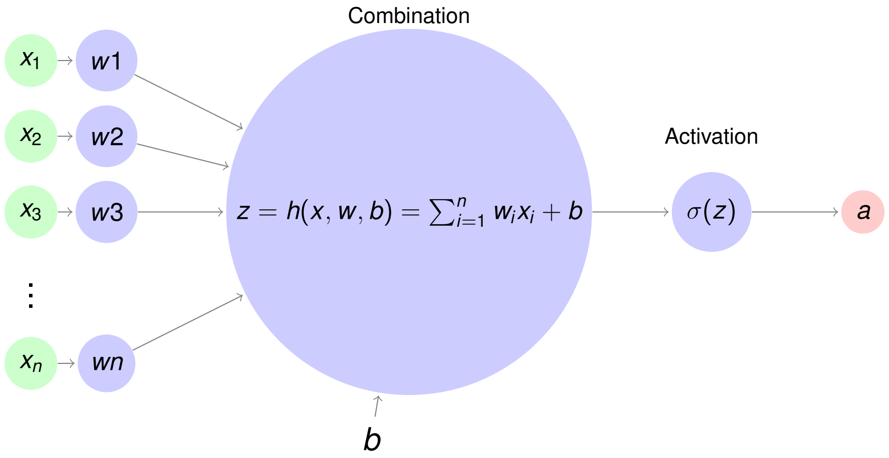
Professor: ANNICK VALIBOUZE

ISUP-SORBONNE UNIVERSITE

April, 2020

A perceptron or neuron

perceptron = binary classifier. It takes as input a vector $x_i \in \mathbb{R}^n$ and computes a binary output *a* (activation)



A perceptron or neuron

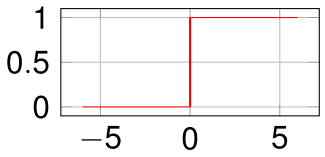
- $(x, w, b) \mapsto h(x, w, b)$ and $z \mapsto \sigma(z)$ are input and activation functions respectively
- $W = (W_1, W_2, \dots, W_n)$ and b are the weights and bias respectively.
- functioning of a neuron: first one applies the input function h to input data x to have z : $z = h(x, w, b)$ and the output a of the neuron is obtained by applying the activation function to z: $a = \sigma(z)$

Activation functions

Activation functions are differentiable. They have different forms that impact the training of the network:

- Heaviside step function

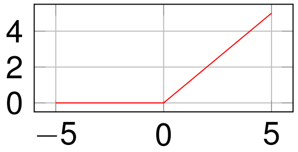
$$\sigma(z) = \begin{cases} 0 & \text{if} \quad z \leq 0 \\ 1 & \text{if} \quad z > 0 \end{cases}$$



Activation functions

- Rectified linear unit (ReLU)

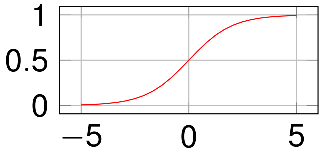
$$\sigma(z) = \begin{cases} 0 & \text{pour} \quad z \leq 0 \\ z & \text{pour} \quad z > 0 \end{cases}$$



Activation functions

- Sigmoid function

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

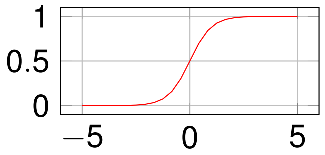


- Sofmax function = generalization of sigmoid and specifically used as activation function of output layer neurons in multi-class classification tasks. for a vector $z = (z_1, z_2, ..., z_K)$, $K \geq 2$, $f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$

Activation functions

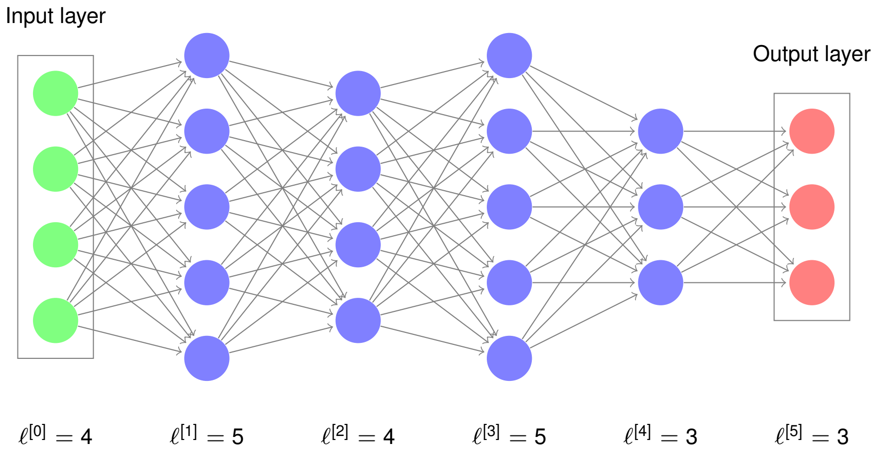
- Hyperbolic tangent function (*TanH*),

$$\sigma(z) = \tanh(z) = \frac{2}{1+e^{-2z}} - 1$$



Definition of MLP

MLP consists of at least three layers of neurons: an input layer, a hidden layer and an output layer.



Definition of MLP

Except for the input neurons, each neuron uses a nonlinear activation function. Neurons of a hidden layer receive the outputs of neurons of the layer which precedes this layer and send their outputs to the neurons of the next layer through the synaptic weights. Similarly, neurons of the output layer receive the outputs of the last hidden layer as input. For a neuron j of a layer ℓ, $\sigma_j^{[\ell]}$ denotes its activation function, $b_j^{[\ell]}$ its bias and $w_{j,k}^{[\ell]}$ connection weights between this neuron j and neurons of previous layer (layer $\ell - 1$). Its output is $a_j^{[\ell]} = \sigma_j^{[\ell]}(z_j^{[\ell]})$ with $z_j^{[\ell]} = \sum_k \omega_{j,k}^{[\ell]} a_k^{[\ell-1]} + b_j^{[\ell]}$.

Training of MLP

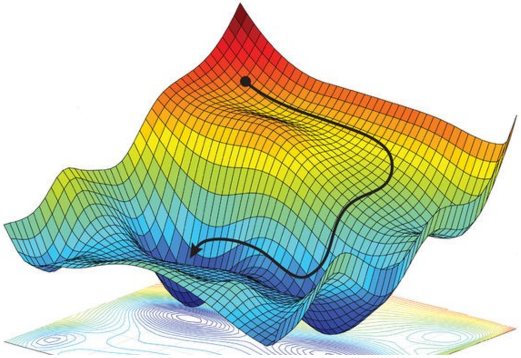
MLP = supervised learning algorithm.
Supervised learning algorithms consist of estimating a function through a corpus $((x^{(i)}, f(x^{(i)}))_{i\in[1,n]}$ where $\forall i, x_i \in \mathbb{R}^p, p \geq 1$.
The estimation principle is to compute an approximation function g of f which minimizes the mean error $\frac{1}{n}\sum_{i=1}^{n} L(g(x^{(i)}), f(x^{(i)}))$ where L denotes the cost function.
In the case of MLP, One has to estimate the weights. Thus the training of MLP consists of solving the following optimization problem:

$$\arg\min_{\omega} \frac{1}{n}\sum_{i} L(g_\omega(x^{(i)}), f(x^{(i)}))$$

MLP is trained with a technique called back-propagation through gradient descent algorithm.

Gradient descent

Gradient descent is an iterative method for minimizing an objective function f defined in a space $\mathbb{E}$ with a norm $\|.\|$. The algorithm computes in the favorable case the global minimum of f and in the unfavorable case one local minimum of f.



Descent gradient algorithm

One starts with an initial $x_0 \in \mathbb{E}$ and repeats until convergence (according to stop criterion) the following procedure :

1. Computation of $\nabla f(x_k)$ the gradient of f at x_k
2. Computation of next iterate $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$ where α_k denotes the learning rate.
3. Stop if $\|\nabla f(x_k)\| \leqslant \varepsilon$, otherwise re-start with 1 .

Back-propagation

Back-propagation consists of computing the gradient of the error function and updating the weights of all neurons of the MLP starting from the output layer until the first layer.
Let's consider MLP with L hidden layers. Let's denote by i and o input and output layers respectively. Let's suppose one wants to train the MLP with a corpus of example $(x^i, d^i)_{i \in \{1,..,n\}}$ where $(x^i, d^i) \in \mathbb{R}^p \times \mathbb{R}^{N_o} \;\; \forall i$. Weights are randomly initialized. One compute a prediction y^i of x^i $\forall i$ and weights are adapted in order to minimize the error between d^i and y^i .

Back-propagation algorithm

Let's consider a neuron j of a layer ℓ. For a training sample (x, d) of the corpus, weights are updated by:

$$w_{jk}^{[\ell]} = w_{jk}^{[\ell]} + \Delta w_{jk}^{[\ell]}$$

with

$$\Delta w_{jk}^{[\ell]} = -\gamma \frac{\partial E}{\partial w_{jk}^{[\ell]}}$$

k denotes the number of a neuron in the layer $\ell - 1$ and γ the learning rate.

$$E = \frac{1}{2}\sum_{j=1}^{N_o}(d_j - y_j)^2 = \frac{1}{2}\sum_{j=1}^{N_o}\left(d_j - a_j^{[o]}\right)^2$$

denotes the square error. N_o is the number of neurons in output layer.

Back-propagation algorithm

Let's denote by $a_j^{[\ell]} = \sigma_j^{[\ell]}\left(z_j^{[\ell]}\right)$ the activation where

$$z_j^{[\ell]} = \sum_k w_{jk}^{[\ell]} a_k^{[\ell-1]} + b_j^{[\ell]} \tag{1}$$

By applying the chain rule:

$$\frac{\partial E}{\partial w_{jk}^{[\ell]}} = \frac{\partial E}{\partial z_j^{[\ell]}} \frac{\partial z_j^{[\ell]}}{\partial w_{jk}^{[\ell]}} \tag{2}$$

According to equation (1):

$$\frac{\partial z_j^{[\ell]}}{\partial w_{jk}^{[\ell]}} = a_j^{[\ell-1]} \tag{3}$$

Back-propagation algorithm

Let's define

$$\delta_j^{[\ell]} = -\frac{\partial E}{\partial z_j^{[\ell]}} \tag{4}$$

Therefore one has:

$$\Delta w_{jk}^{[\ell]} = \gamma \delta_j^{[\ell]} a_j^{[\ell-1]} \tag{5}$$

According the chain rule :

$$\delta_j^{[\ell]} = -\frac{\partial E}{\partial z_j^{[\ell]}} = -\frac{\partial E}{\partial a_j^{[\ell]}} \frac{\partial a_j^{[\ell]}}{\partial z_j^{[\ell]}} \tag{6}$$

and

$$\frac{\partial a_j^{[\ell]}}{\partial z_j^{[\ell]}} = (\sigma_j^{[\ell]})' \left(z_j^{[\ell]} \right) \tag{7}$$

Back-propagation algorithm

if $\ell = o$ (output layer),

$$\frac{\partial E}{\partial a_j^{[o]}} = -\left(d_j - a_j^{[o]}\right) \tag{8}$$

and thus for all neuron j of output layer:

$$\delta_j^{[o]} = \left(d_j - a_j^{[o]}\right)(\sigma_j^{[o]})'\left(z_j^{[o]}\right) \tag{9}$$

Back-propagation algorithm

if ℓ is a hidden layer, thus by applying the chain rule:

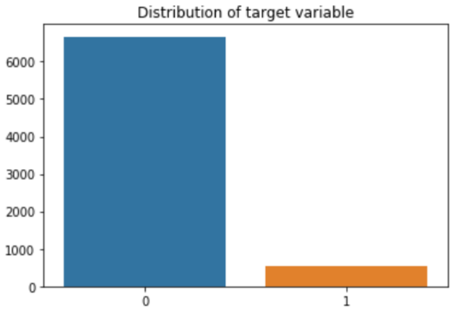
$$\begin{aligned}
\frac{\partial E}{\partial a_j^{[\ell]}} &= \sum_{k=1}^{N_{[\ell+1]}} \frac{\partial E}{\partial z_k^{[\ell+1]}} \frac{\partial z_k^{[\ell+1]}}{\partial a_j^{[\ell]}} \\
&= \sum_{k=1}^{N_{[\ell]+1}} \frac{\partial E}{\partial z_k^{[\ell+1]}} \frac{\partial}{\partial a_j^{[\ell]}} \sum_{p=1}^{N_{[\ell]}} w_{pk}^{[\ell+1]} a_p^{[\ell]} \\
&= \sum_{k=1}^{N_{[\ell+1]}} \frac{\partial E}{\partial z_k^{[\ell+1]}} w_{kj}^{[\ell+1]} a_j^{[\ell]} \\
&= -\sum_{k=1}^{N_{[\ell+1]}} \delta_k^{[\ell+1]} w_{kj}^{[\ell+1]} a_j^{[\ell]}
\end{aligned} \tag{10}$$

Back-propagation algorithm

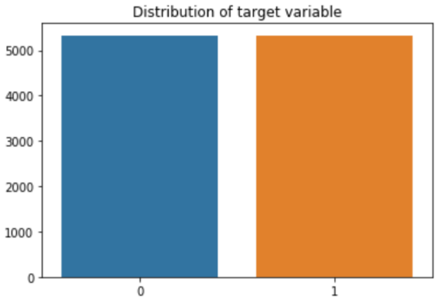
Equations (10) gives a recursive procedure for computing the δ for all units in the network which are then used to compute the weight updates according to descent gradient equation.

hypothyroidism detection : Context and dataset

We have medical database on patients. The dataset has 6 real attributes and one target y ($y = 0$ for a normal patient an $y = 1$ for hypothyroid one).
Goal : predict trough this dataset and MLP whether an upcoming patient referred to the clinic is hypothyroid.
Class Imbalance Problem so use of data augmentation.



(a) Original dataset



(b) data augmented

hypothyroidism detection : Network architecture and training

Deep learning framework : `pytorch`
We built a small network with 2 hidden layers and bacthnormalisation. The network is trained for 500 epochs.

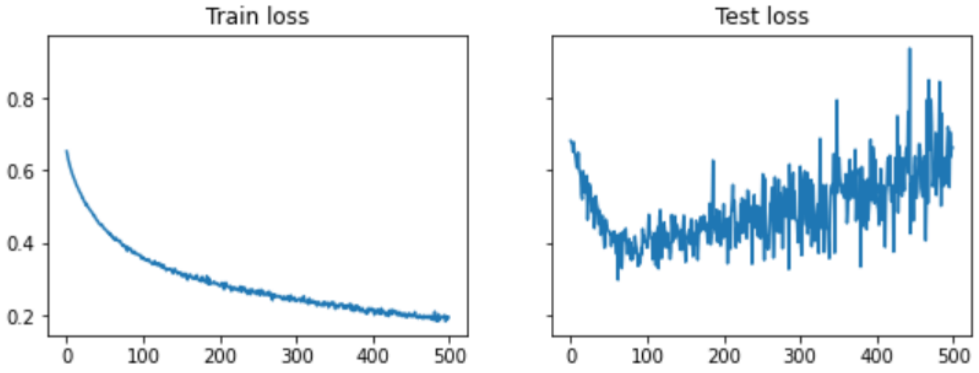
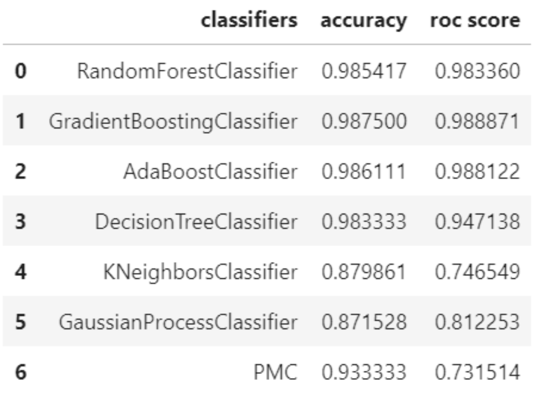


Figure 2: Training curves

hypothyroidism detection : inference

Metrics: accuracy and ROC score
93% of accuracy and 73% of ROC score on test set of size 1440.
We compared MLP with other machine learning algorithms on same dataset.



	classifiers	accuracy	roc score
0	RandomForestClassifier	0.985417	0.983360
1	GradientBoostingClassifier	0.987500	0.988871
2	AdaBoostClassifier	0.986111	0.988122
3	DecisionTreeClassifier	0.983333	0.947138
4	KNeighborsClassifier	0.879861	0.746549
5	GaussianProcessClassifier	0.871528	0.812253
6	PMC	0.933333	0.731514

Figure 3: Comparison of MLP with other machine learning algorithms

CNN : Introduction

Multilayer perceptrons are less accurate and time consuming when it comes to deal with unstructured data (images, videos). Their number of parameters depend on the dimension of input data.
For instance, for a simple image binary classification task (dog or cat) with MLP, one can have a model with 10^9 parameters. Unless we have an extremely large dataset, lots of GPUs, and an extraordinary amount of patience, learning the parameters of this network may turn out to be impossible.
CNNs are convenient for such data since the size of their parameters does not depend on input data dimension.

Mathematical convolution operations

- In mathematics, the convolution between two functions $f, g : \mathbb{R}^d \to \mathbb{R}$ is defined by :

$$[f \circledast g](x) = \int_{\mathbb{R}^d} f(z)g(x-z)dz$$

- In case of discrete measure, for example measure that has $\mathbb{Z}$ as support, convolution is defined as :

$$[f \circledast g](i) = \sum_{a\in\mathbb{Z}} f(a)g(i-a)$$

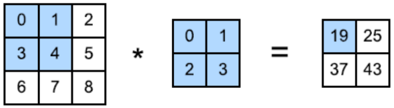
- Convolution between two matrix $X \in \mathbb{M}_{nx,px}$, $F \in \mathbb{M}_{nf,pf}$ is defined as :

$$\forall(i,j), R(i,j) = \sum_{n=0}^{nx-1}\sum_{p=0}^{px-1} X(n,p)F(i-n,j-p)$$

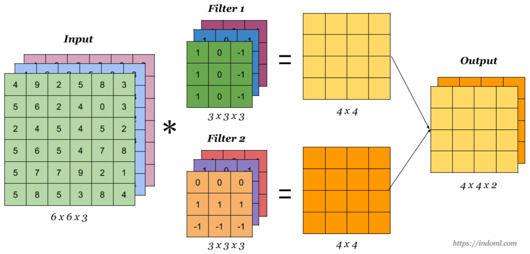
where $R \in \mathbb{M}_{nr,pr}$ with $nr = nx - nf + 1$ and $pr = px - pf + 1$.

Convolution between images and filters

A numerical image is represented by a matrix (2D array for gray scale and 3D for RGB). The mathematical convolution between two matrix is applied to images and filters in image processing (edge and contour detection, smoothing, thresholding, etc). One can apply several filters to same image to detect more features.



(a) 2D convolution with one filter



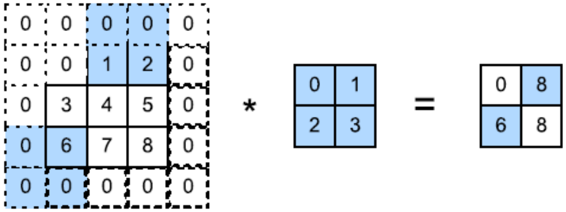
(b) 3D convolution with two filters

Figure 4

Padding and strided convolution

- **padding** convolution shrinks output and therefore throws away a lot of information that are in the edges. To solve these problems we can pad the input image before convolution by adding some rows and columns of 0 to it.
- **strided convolution** we used a stride s as the number of pixels we will jump when we are convolving filter.

if a matrix image $n \times n$ is convolved with $f \times f$ filter and padding p and stride s it gives us $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$ matrix.



Convolution layer

Given an input X:

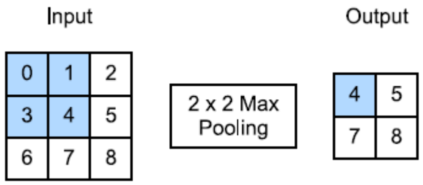
- Apply a padding *p* (optional)
- Choose a stride *s*
- Choose dimension and number of filters
- Convolve X with each of the filters and concatenate the results to have the output.
- Add a bias (optional)
- Apply an activation function (ReLU)

Pooling layer

CNNs sometimes use pooling layers to reduce the size of the inputs, speed up computation, and to make some of the features it detects more robust.
There exits two kind of pooling layer:

- **Max pooling** is saying, if the feature is detected anywhere in this filter then keep a high number.
- **Average pooling** is taking the averages of the values instead of taking the max values.

Pooling layers have no parameters to learn. They only have hyperparameters: filter size *f*,stride *s*, Max or average pooling.



Conv-Net

A Conv-Net is a succession of several convolution layers (eventually followed by layers of pooling) followed by fully connected layers. CNNs are trained with back-propagation through gradient descent.

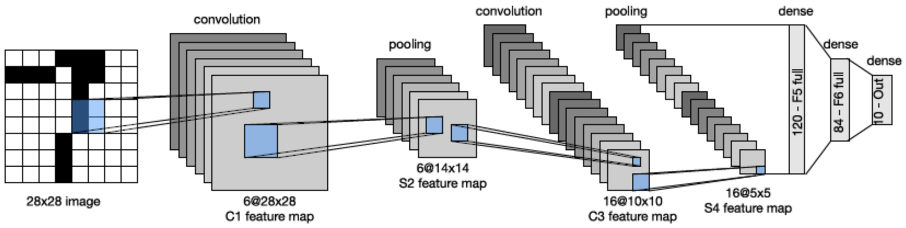


Figure 7: Example of Conv-Net: Le-Net5

covid-19 detection : Dataset and context

A team of researchers in collaboration with medical doctors have created a database of chest X-ray images for COVID-19 positive cases along with Normal and Viral Pneumonia images. There are 219 COVID-19 positive images, 1341 normal images and 1345 viral pneumonia images.
Goal: Build a CNN that will predict with upcoming x-ray images whether the patients are normal, have COVID-19 or viral pneumonia.

covid-19 detection : Dataset and context

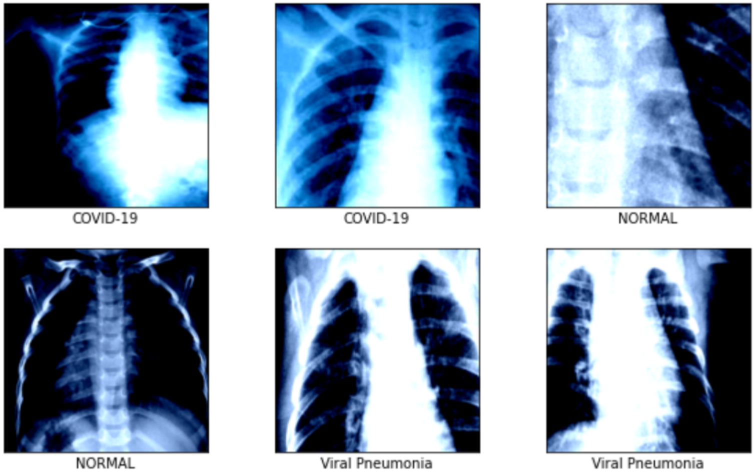


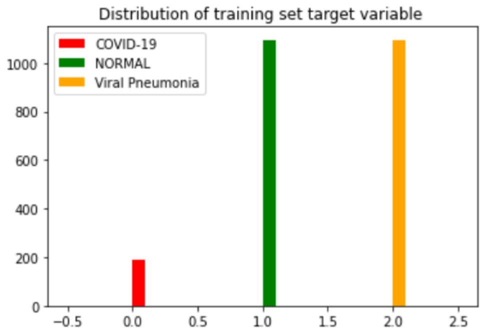
Figure 8: Some images of database

Network architecture and training

We built two networks with deep learning framework `pytorch` using technique of transfer learning:

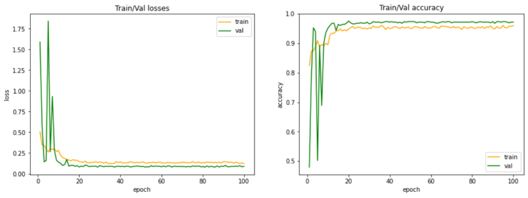
- **resnet18** a deep convolutional neural network with 18 layers.
- **darknet53** a very deep convolutional neural network with 53 layers.

We trained these two networks with training set of size 2500 (289 COVID-19 cases,1095 normal and the rest Viral pneumonia).

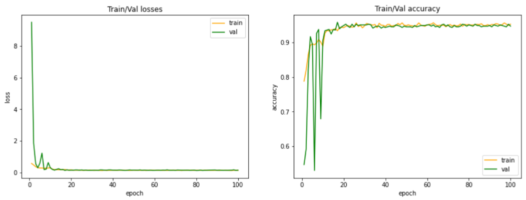


Network architecture and training

We trained these two networks on GPU for 100 epochs using data augmentation techniques (random size crop, random horizontal flip, normalization).
The training lasted 108 and 125 minutes for *resnet18* and *darknet53* respectively.



(a) resnet18



(b) darknet53

Figure 10: Training curves

Inference

We evaluated both of the networks with a test set containing 530 images.

- **resnet18**

```
                 precision    recall  f1-score   support

       COVID-19       0.97      1.00      0.98        30
         NORMAL       0.97      0.98      0.98       250
Viral Pneumonia       0.98      0.96      0.97       250

       accuracy                           0.98       530
      macro avg       0.97      0.98      0.98       530
   weighted avg       0.98      0.98      0.98       530
```

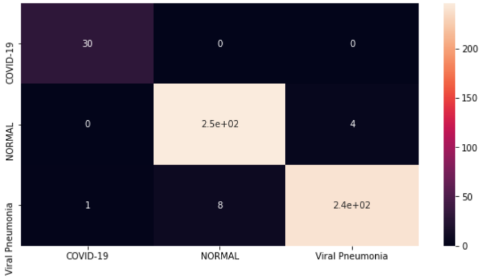


Figure 11: resnet18 evaluation

Inference

- **darknet53**

```
                 precision    recall  f1-score   support

        COVID-19       0.97      1.00      0.98        30
          NORMAL       0.97      0.98      0.98       250
 Viral Pneumonia       0.98      0.96      0.97       250

        accuracy                           0.98       530
       macro avg       0.97      0.98      0.98       530
    weighted avg       0.98      0.98      0.98       530
```

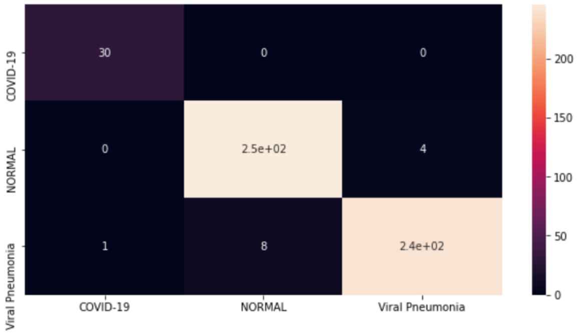


Figure 12: darknet53 evaluation

Inference

As shown above, **resnet18** achived 100% of recall score, 97% of precision on COVID-19 class and 98% of accuracy score on whole images while **darnket53** had 93% of recall and precision on COVID-19 and 96% of accuracy on whole images.
darknet53 perfomed less well because, it is very deep and thus requires large dataset to avoid over-fitting.

Inference

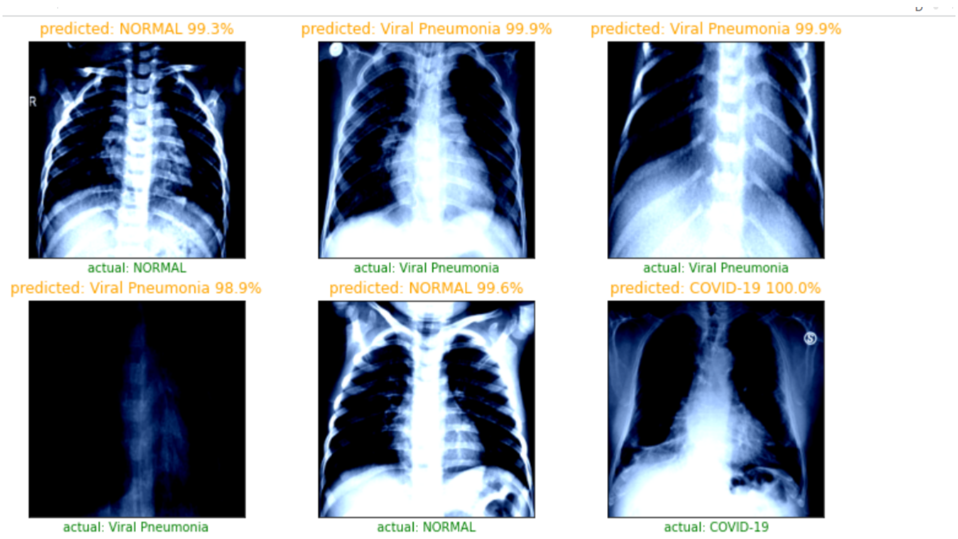


Figure 13: Some predictions with resnet18

Definition of SOM

SOM are introduced by Kohonen in 1980s and belong to deep unsupervised learning algorithms. SOM are artificial neural networks with no hidden layers and produce a direct mapping between the training set and the output network. They can be used in dimensionality reduction or clustering tasks.

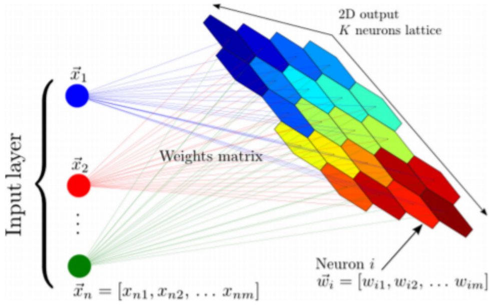

2D output
K neurons lattice
$\vec{x}_1$
$\vec{x}_2$
Input layer
Weights matrix
Neuron i
$\vec{w}_i = [w_{i1}, w_{i2}, \dots w_{im}]$
$\vec{x}_n = [x_{n1}, x_{n2}, \dots x_{nm}]$

Figure 14: A schematic representation of a SOM. The training set of n examples is mapped into a two-dimensional map of K neurons that are represented by vectors containing the weights for each input attribute [4].

Training algorithm of SOM

Let's consider a SOM with K output neurons and a training set of n examples $(x_1, x_2, .., x_n$ where $x_i = (x_{i1}, ..., x_{im}) \in \mathbb{R}^m \; \forall i \in \{1, .., n\})$.
Each output neuron k is associated with a weight vector $w_k = (w_{k1}, ..., w_{km}) \in \mathbb{R}^m$. These weights are randomly initialized.
For every iteration t, all examples of training set are individually processed and weights of output neurons are updated to optimally match each example. This procedure produces the self-organization of the maps and conserves the topology of the training space.

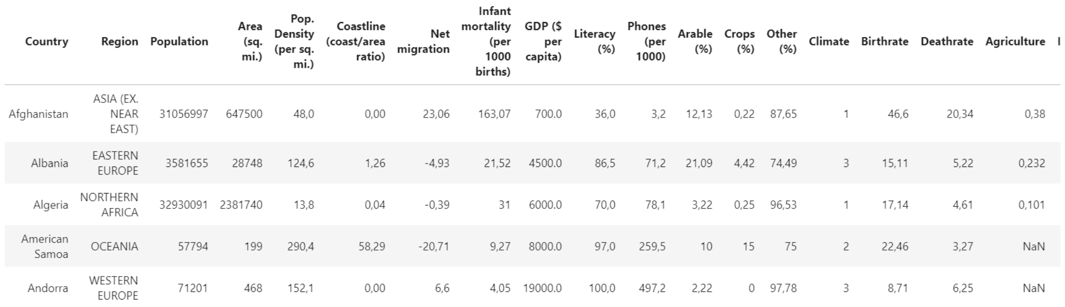
Training algorithm of SOM

Algorithm

```
For t=1,...,number of iteration:
.   For i=1,...,training set size:
      • Compute $d_k = d(x_i, w_k(t)) = \sqrt{\sum_{p=1}^{m}(x_{ip} - w_{kp}(t))^2} \quad \forall k \in \{1, ..., K\}$
      • Choose $b = \arg\min_{k\in\{1,...,K\}} d_k$. There is the number of best matching unit (bmu). It's
        the closest in term of Euclidean distance to $x_i$
      • Update weights according to the following formulas:
        $w_k(t+1) = w_k(t) + \alpha(t)H_{b,k}(t)[x_i - w_k(t)]\ \forall k \in \{1, ..., K\}$ where:
        $\alpha(t)$ denotes the learning rate (decreases at each iteration);
        $H_{b,k}(t)$ ) denotes the neighborhood function that also decreases with iteration and with
        the distance between the nodes b and k. These function implies that weights of closest
        neurons to the bmu are strongly updated and instead weights of furthest one are
        slightly modified.
  end For
end For
```

Application: database and context

We have a database containing information (information on population, region, area size, infant mortality and more) about 227 countries of the world. Each country has 20 features
Goal: Clustering of countries according to their quality of life.



Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climate	Birthrate	Deathrate	Agriculture	I
Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00	23,06	163,07	700.0	36,0	3,2	12,13	0,22	87,65	1	46,6	20,34	0,38	
Albania	EASTERN EUROPE	3581655	28748	124,6	1,26	-4,93	21,52	4500.0	86,5	71,2	21,09	4,42	74,49	3	15,11	5,22	0,232	
Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04	-0,39	31	6000.0	70,0	78,1	3,22	0,25	96,53	1	17,14	4,61	0,101	
American Samoa	OCEANIA	57794	199	290,4	58,29	-20,71	9,27	8000.0	97,0	259,5	10	15	75	2	22,46	3,27	NaN	
Andorra	WESTERN EUROPE	71201	468	152,1	0,00	6,6	4,05	19000.0	100,0	497,2	2,22	0	97,78	3	8,71	6,25	NaN	

Figure 15: Database of countries

Implementation of SOM and results

We used an open source python library `MiniSom`. We decided to form 20 groups of countries. Thus we created a 5×4 output map (20 outputs neurons). Countries that have the same bmu form a group. We trained our SOM for 100 iterations.

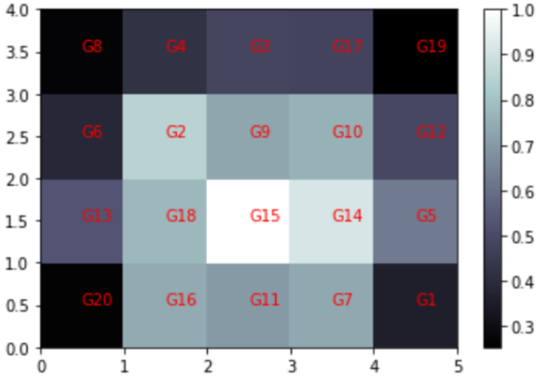


Figure 16: Output map

Implementation of SOM and results

We noticed that countries that belong to the same group have actually the same quality of life. Example of 2 groups:

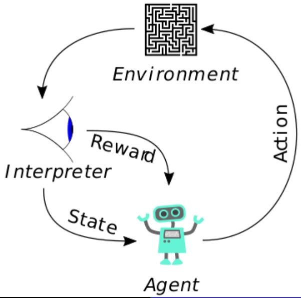
```
{'G1':  ['Afghanistan ','Benin ','Burkina Faso ','Burma ', 'Cambodia ',
              'Cameroon ','Central African Rep. ','Chad ','Congo, Dem. Rep. ',
"Cote d'Ivoire ",'Ethiopia ','Guinea-Bissau ','Laos ','Lesotho ','Liberia ',
              'Malawi ','Mali '],
```

```
'G8': ['Bulgaria ','Czech Republic ','Denmark ','France ','Germany ',
 'Italy ','Latvia ','Liechtenstein ','Malta ','Netherlands ','Portugal ',
 'Slovakia ','Spain ','United Kingdom ','United States '],
```

Please see report to see the remaining groups.

Reinforcement learning : introduction

RL is an area of artificial intelligence concerned with training an agent which interacts with an environment in order to maximize rewards. The agent interacts with the environment by performing actions. Each action leads to a new state of the environment associated with a reward that could be positive or negative. Training process consists of teaching the agent to learn a policy that allows it to maximize its total reward across an episode. (an episode is everything that happens between the firs state and the last state within the environment).



Applications of RL

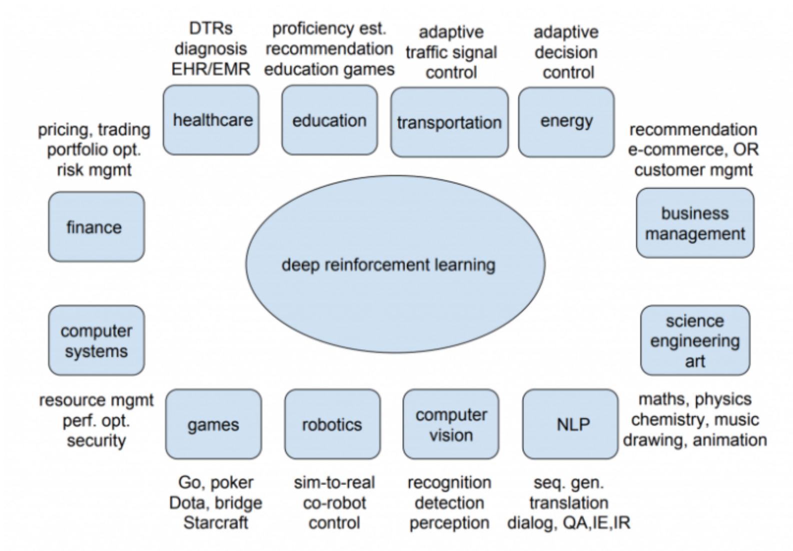


Figure 18: Reinforcement learning applications [17]

Markov Decision Process (MDP)

Each state within an environment is an outcome of its previous state which in turn is a consequence of its previous state. Dealing with all this information, even for environments with short states, is impossible.
To avoid this problem, one assumes that each state follows a Markov property, i.e., each state depends only on the previous state and the transition function from that state to the current state.
In figure[19] there are 2 scenarios with 2 different starting points and the agent takes different paths to reach the same state (red point).The markov property implies that to exit the labyrinth, we do not need to know the path taken by the agent to reach the red state point but only need the information on the red state.

Markov Decision Process

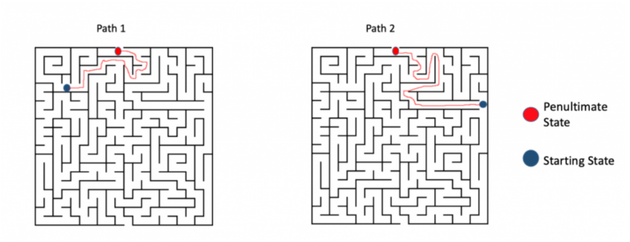


Figure 19: Illustration of MDP [17]

A MDP is a tuple (S, A, T, R) where S and A denote finite sets of states and actions respectively. $T : S \times A \times S \rightarrow [0; 1]$ is the transition function and denotes the probability of being at the state s' after performing the action a at the state s (noted as $T(s, a, s')$). $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function. The Markov property is defined as:
$\mathbb{P}\left(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \ldots\right) = \mathbb{P}\left(s_{t+1} | s_t, a_t\right) = T\left(s_t, a_t, s_{t+1}\right)$

Policy, Bellman equation and Q-learning

Given a MDP (S, A, T, R) , a policy or strategy is a function π that given a state s computes an optimal action a. Policy can be deterministic $\pi : S \to A$ or stochastic $\pi : S \times A \to [0, 1]$. In this presentation, we'll only consider the deterministic case. The policy is chosen in accordance with the reward function R. Let's denote by $r_t = R(s_t, \pi(s_t), s_{t+1})$ the reward obtained by the agent after performing the action $\pi(s_t)$ according to policy π. There are three optimality criteria:

- $E(\sum_{t=0}^{h} r_t)$ finite horizon;
- $\liminf_{h\to+\infty} E\left(\frac{1}{h}\sum_{t=0}^{h} r_t\right)$ or $\limsup_{h\to+\infty} E\left(\frac{1}{h}\sum_{t=0}^{h} r_t\right)$ average reward;
- $E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$ discounted, infinite horizon, $\gamma \in [0; 1]$

In this presentation, we'll only consider the last criteria.

Policy, Bellman equation and Q-learning

Once policy and criteria are defined, two functions can be defined:

- $V^\pi : S \rightarrow \mathbb{R}$, The value of a state s under policy π and is the expected reward when starting in s and performing π

  $V^\pi(s) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s\right\} = \sum_{s'\in S}\left[R\left(s,\pi(s),s'\right)+\gamma V^\pi\left(s'\right)\right]T\left(s,\pi(s),s'\right)$
- $Q^\pi : S \times A \rightarrow \mathbb{R}$ state-action value function, defined as the expected reward starting from state s, performing action a and following policy π

  $Q^\pi(s,a) = E_\pi\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k}|s_t = s, a_t = a\right\} = \sum_{s'\in S}\left[R\left(s,a,s'\right)+\gamma Q^\pi(s',\pi(s'))\right]T\left(s,a,s'\right)$

These two functions are equivalent i.e $V^\pi(s) = Q^\pi(s,\pi(s))$. The recursive expression of $V^\pi(s)$ or $Q^\pi(s,\pi(s))$ is called Bellman equation.
Q-learning is defined as training the agent to learn an optimal policy π^* (policy that receive the most positive rewards): $\forall s$, $\forall \pi$

$$Q^{\pi^*}(s,a) \geq Q^\pi(s,a)$$

Policy, Bellman equation and Q-learning

The optimal state-action value function satisfies the optimal Bellman equation :

$$Q^*(s,a) = \sum_{s' \in S} \left[R\left(s,a,s'\right) + \gamma \max_{a' \in A} Q^*\left(s',a'\right) \right] T\left(s,a,s'\right)$$

Q-learning consist of learning $Q^*(s,a) \quad \forall (s,a) \in S \times A$.
$Q^*(s,a)$ values are updated according to the following equation:

$$Q^*_t(s,a) = Q^*_{t-1}(s,a) + \alpha TD_t(s,a)$$

where

$$TD_t(s,a) = Q^*_t(s,a) - Q^*_{t-1}(s,a) = \sum_{s' \in S} \left[R\left(s,a,s'\right) + \gamma \max_{a' \in A} Q^*\left(s',a'\right) \right] T\left(s,a,s'\right) - Q^*_{t-1}(s,a)$$

temporal difference and $\alpha \in [0\,;\,1]$.

Policy, Bellman equation and Q-learning

If $\alpha = 0$, nothing is learnt since there's no update ($Q^*_t(s,a) = Q^*_{t-1}(s,a)\ \forall t$) and if $\alpha = 1$, update is performed without taking into account previous values. Thus α should be in the interval]0 ; 1[.

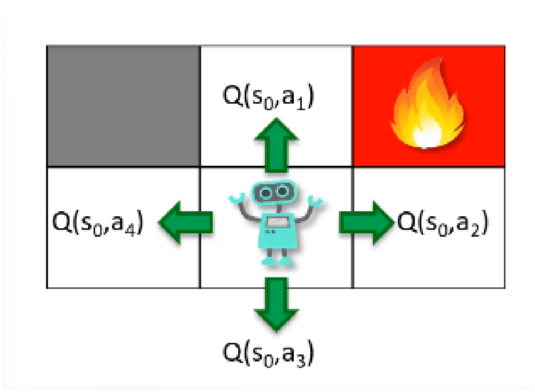


Figure 20: Q(s,a) values [19]

Deep Q-learning : principle

Deep Q-learning consists of computing an approximation of state-action value function using a neural network. The network takes as input the state of the agent and predicts possible Q-values associated to any action in that state.

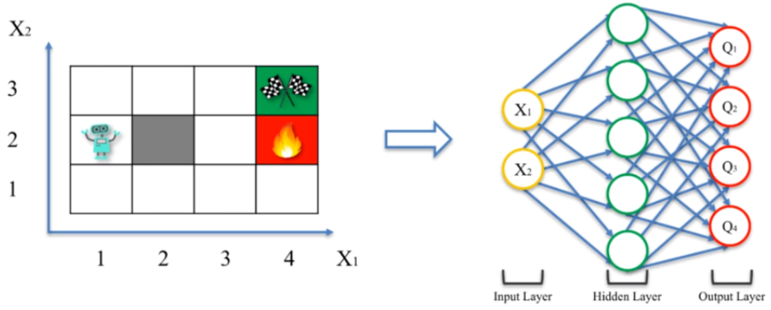


Figure 21: Deep Q-learning [19]

Deep Q-learning : principle

The cost function is the square error between predicted value $\hat{Q}$ and target value Q. However we do not know the target values.Thus target values $Q_t(s, a)$ are approximated by previous values $Q_{t-1}(s, a)$. Therefore at each iteration t, the cost function is defined as :

$$L = \sum_{a,s} (Q_{t-1}(s, a) - \hat{Q}_t(s, a))^2$$

where $\hat{Q}_t(s, a)$ are predicted values by the network.

Deep Q-learning: Experience Replay

Experience Replay is a technique used to avoid over-fitting. In fact some states are very rare in an environment and this leads to imbalanced data leading the network to over-fit on frequent states. To solve this problem, the system saves the discovered data (state,action,reward,next state) in a table and during training, examples are randomly chosen from the table to train the network.

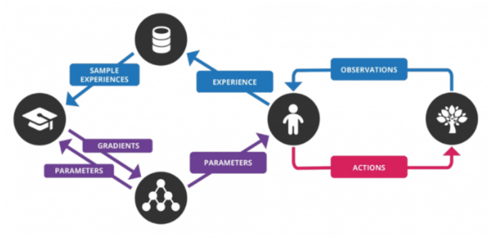


Figure 22: Training with Experience Replay [17]

References

[1] Y. Li, *REINFORCEMENT LEARNING APPLICATIONS*, arXiv:1908.06973

[2] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*

[3] T. Schaul, J. Quan, I. Antonoglou and D. Silver, *PRIORITIZED EXPERIENCE REPLAY*

[4] M. Carrasco Kind and Robert J. Brunner, *SOMz: photometric redshift PDFs with self organizing maps and random atlas*

[5] , https://www.kaggle.com/fernandol/countries-of-the-world

[6] , B. Krose, P. van der Smagt, *An introduction to Neural Networks*

[7] http://odds.cs.stonybrook.edu/annthyroid-dataset/

[8] https://pytorch.org/

[9] Andrew P.Bradley, *The use of the area under the ROC curve in the evaluation of machine learning algorithms*

[10] https://test.pypi.org/project/MiniSom/1.0/

[11] https://docs.gimp.org/2.6/fr/plug-in-convmatrix.html

[12] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola,*Dive into Deep Learning,Release 0.7.0*

[13] https://www.mathworks.com

[14] Cours de spécialisation deep learning d'Andrew Ng sur Coursera, https://www.coursera.org/specializations/deep-learning

[15] M.E.H. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al-Emadi, M.B.I. Reaz, "Can AI help in screening Viral and COVID-19 pneumonia?" arXiv preprint, 29 March 2020, https://arxiv.org/abs/2003.13145. https://www.kaggle.com/tawsifurrahman/covid19-radiography-database

[16] Wouter M. Kouw, M. Loog , *An introduction to domain adaptation and transfer learning*

[17] https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/

[18] https://www.superdatascience.com/blogs/the-ultimate-guide-to-self-organizing-maps-soms

[19] https://www.udemy.com/course/intelligence-artificielle-az/learn/lecture/10858968?start=0overview

[20] https://en.wikipedia.org/wiki/Reinforcement$_l earning/media/File: Reinforcement_l earning_d iagram.svg$