# EXAM 21/02/2020

## EXERCISE N°1

### *POINT1*

An hypothesis class $H$ is PAC learnable with respect to $Z$ and a loss function $l : HxZ \rightarrow R^+$, if there exists a function $m_H: (0,1)^2 \rightarrow N$ (sample complexity) and a learning algorithm such that for every $\delta \in (0,1)$, $\varepsilon \in (0,1)$, for every distribution $D$ over $Z$ and for every true labeling function $f : X \rightarrow \{0,1\}$, if the realizability assumption holds with respect to $H, D, f$ ; when running the learning algorithm on $m \geq m_H(\varepsilon, \delta)$ iid samples generated by $D$ and labeled by $f$ the algorithm returns an hypothesis $h$ ,such that, with probability $\geq 1 - \delta :\ L_{D,f}(h) \leq \varepsilon$.

- $l : HxZ \rightarrow R^+$ where $Z = XxY$ be the loss function namely a function that given an hypothesis provides a measure of how much we lose by predicting the value $h(\vec{x})$ for $\vec{x}$ instead of the.
- A learning algorithm is a type of algorithm that given in input data it learns a function that is map the input into output also for data it has not seen yet.
- $L_{D,f}(h) = E_{z \sim D}[l(h,z)]$ is the so called risk or generalization error.

### *POINT2*

The realizability assumption is the assumption that consists of having an hypothesis $h^* \in H$ such that $L_{D,f}(h^*) = 0$ . If this assumption doesn't hold, then the formulation of PAC learning changes into agnostic PAC learning.

An hypothesis class $H$ is agnostic PAC learnable with respect to $Z$ and a loss function $l : HxZ \rightarrow R^+$, if there exists a function $m_H: (0,1)^2 \rightarrow N$ (sample complexity) and a learning algorithm such that for every $\delta \in (0,1)$, $\varepsilon \in (0,1)$, for every distribution $D$ over $Z$, when running the learning algorithm on $m \geq m_H(\varepsilon, \delta)$ iid samples generated by $D$ the algorithm returns an hypothesis such that, with probability $\geq 1 - \delta :\ L_{D,f}(h) \leq min_{h' \in H} L_{D,f}(h') + \varepsilon$ where $L_{D,f}(h) = E_{z \sim D}[l(h,z)]$.

### *POINT3*

Let $H$ be a finite hypothesis class. Let $\delta \in (0,1)$, $\varepsilon \in (0,1), m \in N$ such that : $m \geq \frac{\log\left(\frac{|H|}{\delta}\right)}{\varepsilon}$, then for any labeling function $f$, for any distribution $D$ for which the realizability assumption holds with probability $\geq 1 - \delta$ we have that for every ERM hypothesis $h_S$ it holds that : $L_{D,f}(h_S) \leq \varepsilon$.

## EXERCISE N°2

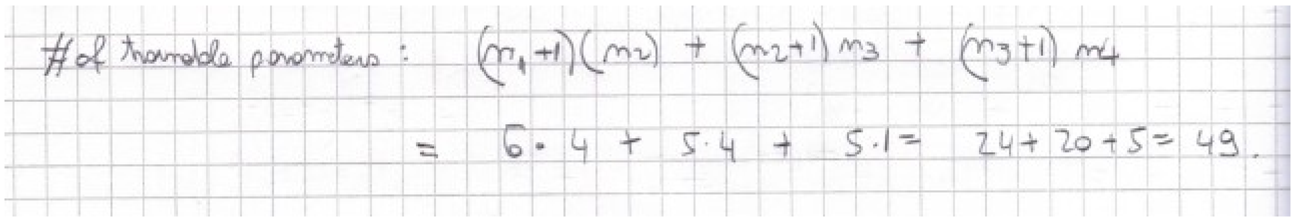
### *POINT1*

Neural networks models are the most complex models that we can use. To use them we need to define the architecture of a neural network i.e. the graph specifics or practically speaking the number of edges and neurons of a neural network and the activation function.
Therefore the hypothesis class of neural networks becomes :
$H_{V,E,\sigma} = \{h_{V,E,\sigma,w} : R^{|V_0|-1} \rightarrow R^{|V_T|}\ where\ w\ is\ a\ mapping\ from\ E\ to\ R\}$. So, the learning process of a neural network is aimed to learn the weights in order to define a function that maps the set of edges to the set of real numbers.

In particular, about the graph specific, one important thing is that the output layer of the neural network must be composed by only two neurons for binary classification one used to highlighting the probability that the label of the input will be +1 while the other to highlighting the probability that the label of the input will be -1 . This is due to the fact that, binary classification is a problem that has only two possible output values i.e. the values +1 or -1. For what concerns the other layers and the activation function, a good strategy to choose them, is to copy the architecture of neural networks already used in literature that works well with the problem of binary classification.

One neuron?

### *POINT2*



# of trainable parameters : $(n_1+1)(n_2) + (n_2+1)n_3 + (n_3+1)n_4$

$= 6 \cdot 4 + 5 \cdot 4 + 5 \cdot 1 = 24 + 20 + 5 = 49.$

### *POINT3*

The main provision done by CNN to reduce the number of parameters are the followings :

- Layers are not fully connected one's like for MLP. That means given a non fully connected layer of a CNN we have that the neurons of the next layer are connected with not all the neurons of the current one
- Parameters are shared in a layer. In addition to what we have said before, namely that the layers are not fully connected, the weights on such edges are the same for a certain subset of edges therefore, the numbers of parameter to learn decreases

## EXERCISE N°3

### *POINT1*

To have decision boundaries that are polynomial functions of degree M we can use two strategies :

- We can use SVM with polynomial kernels with Q=M
- We can use different representation of the data namely feature expansion/transformation where we map a vector $\vec{x}$ to a polynomial function of degree M in the features $x_i$ *'s of* $\vec{x}$ and then use linear models with the new feature space.

  Ex $\vec{x} \in R \rightarrow \varphi(\vec{x}) = \begin{bmatrix} x_1 \\ . \\ . \\ . \\ x_M \end{bmatrix} \rightarrow linear\ model\ for\ the\ transoformed\ \varphi(\vec{x}) = w_0 + w_1x_1 + \cdots +$

  $w_Mx_M\ and\ so\ for\ binary\ classification\ sign(w_0 + w_1x_1 + \cdots + w_Mx_M)$

### *POINT2*

What we have introduced in the previous point is related to Kernels SVM because to have polynomial functions as boundaries we use polynomial kernels with Q=M. In addition to that the main advantage is that we do not have to apply any transformation of the input because everything is done by the kernel.

## $POINT3$

The strategy to select the best value for parameter $M$ is to use cross-validation and once $M$ is chosen we use all the data to learn the best model with such parameter.

To be more precise, cross validation consists in a way to select a value of a parameter ϑ by understanding what is the best value that such parameter can assume with the data we have.
To do so, we split the dataset into k different folds of size m/k (this quantity is supposed to be integer).
Then for each value of the parameter ϑ we find the best model for all the possible k-1 folds that we can select. In addition to that, we compute the average error of each parameter as the average of the errors on the folds left out, and when we have repeated such procedure for all the values of the parameters, then we select the optimal one by choosing the one that minimizes the error computed for each parameter. To conclude, we train our model on all the dataset with the parameter found.
The pseudo code is the following :

```
Input : $S = \left((\overrightarrow{x_1}, y_1) \ldots (\overrightarrow{x_m}, y_m)\right)$; set of parameters θ; integer k; learning algorithm A

Split $S$ into $S_1, \ldots, S_k$

foreach ϑ ∈ θ
   for i = 1 ... k
      $h_{i,\vartheta} = A(S \backslash S_i; \vartheta)$
   $error(\vartheta) = \frac{1}{k}\sum_{i=1}^{k} L_{S_i}(h_{i,\vartheta})$
Outuput : $\vartheta^* = argmin_{\vartheta}(error(\vartheta))$
          $h_{\vartheta^*} = A(S; \vartheta^*)$
```