Lecture 24 - 09-06-2020

1.1 Neural Networks

1.1.1 Feedforward NN

$$f : \mathbb{R}^d \to \mathbb{R}^n$$

Done with a combination of preditors.

$$g(x) = \sigma(w^T x) \qquad \sigma \textbf{ is a nonòinear activation function}$$

Entire structure is a DAG:

$$G = (V, E) \quad \textit{DAG directed and acyclic graph}$$

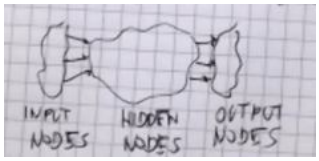

Figure 1.1:

Multilayer NN (special case of Feedforward)

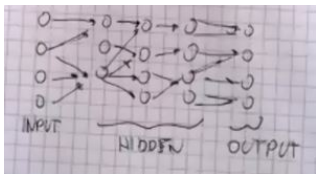

Figure 1.2:

Computing a function:

$$g(\cdot) = \sigma(w^T \cdot)$$

where $\cdot$ is the input of the previous layer.
We are going to split the nodes:

$$V = V_m \cup V_{hid} \cup V_{out} \qquad |V_{in}| = d$$

If (i, j) is an edge:

$$(i, j) \in E \Rightarrow w_{ij} \in \mathbb{R} \qquad \textit{where } i \textit{ and } j \textit{ are nodes}$$

W weighted matrix:

$$w_{ij} = 0 \quad if \quad (i,j) \notin E$$

We have then G, w, σ parameters that define a function compute by the network:

$$f_{G,w,\sigma} : \mathbb{R}^d \to \mathbb{R}^n$$

Let's pick any node j that $j \in V \setminus V_{in}$:

$$w(j) : \{w_{ij} : (i,j)$$

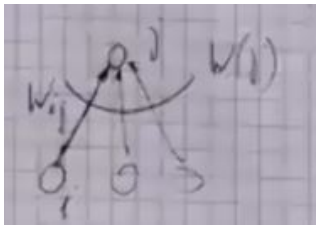

Figure 1.3:

MANCAAAA

Each node j has a state value which is evaluated during the computation of the function.

$$V_j = \sigma(w(j)^T v(j))$$

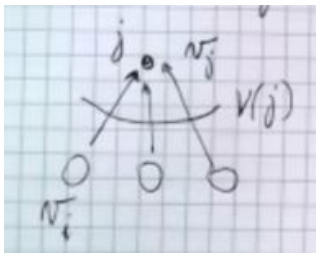

Figure 1.4:

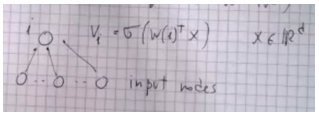

Figure 1.5:

$$V_i = \sigma(w(i)^T x) \qquad x \in \mathbb{R}^d$$
$$f_{G,w,\sigma}(x) \qquad v_i = \sigma(w(j)^T v(j))$$

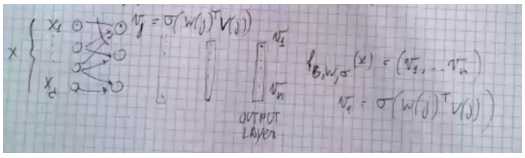

Figure 1.6:

$$f_{G,w,\sigma}(x) = (v_1, ..., v_n) \qquad v1 = \sigma(w(j)^T v(j))$$

What are the parameters I learn?
G, σ fixed and w is trained.
I can define a class F:

$$F_{G,\sigma}\textbf{class of all predictors } f_{G,w,\sigma} \textbf{ for } w \textbf{ variable}$$

σ is the identity and we can do other model like linear regression etc.

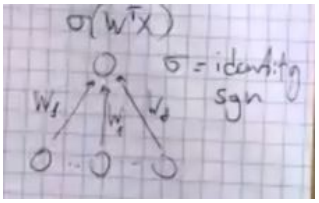

Figure 1.7:

Case of single output node:

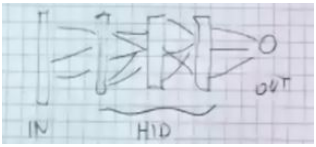

Figure 1.8:

In classification:

$$v_0 = f_{G,w,\sigma}(x) = sgn(w(0)^T v(0))$$

In regression:

$$w(0)^T v(0)$$

This depends on the problem. How do I choose the activation function σ? Could be a Sigmoid or Relu (0 in negative and 1 positive part) or other variant of this like Leaky Relu.
The first one is bounded the other one are not.

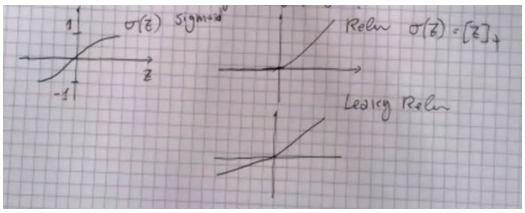

Figure 1.9:

How many layers?
How many nodes in each layer?
Patter connectivity between layers?
Activation function?
In order to define our class $F_{G,\sigma}$: we are gonna ask ourself some structural question. How many layers do I need in general? Is there a minimum range of layers?

Theorem

$\forall\, d\, \exists\, G = (V, E)$ *with* $d+1$ *input nodes, one hidden layer, one output node s.t* $F_{G,sgn}$ *contains all function of the form* $f : \{-1,1\}^d \to \{-1,1\}$

Proof

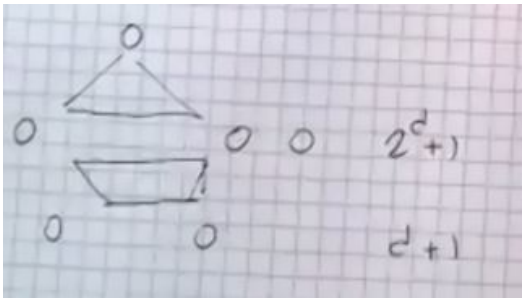

Figure 1.10:

$$f : \{-1,1\}^d \to \{-1,1\}$$
$$\bar{x}_1, ..., \bar{x}_N \qquad f(\bar{x}_i) = 1 \qquad N \leq 2^d$$

The *i-th* hidden node computes the function:

$$g_i(x) = sgn(\bar{x}^T \bar{x}_i - d + 1) \qquad \bar{x} \rightsquigarrow \hat{x}(\bar{x}, 1)$$
$$w(i) = (\bar{x}_i, -d+1) \qquad w(i)^T \hat{x} = \bar{x}^T \bar{x}_i - d + 1$$

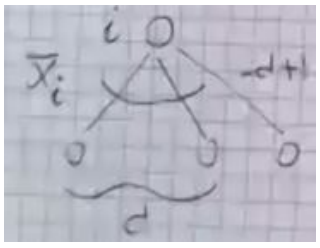

Figure 1.11:

$$x^T x_i = \begin{cases} d & if\ x = x_i \\ \leq d-2 & if\ x \neq x_i \end{cases}$$

Let's see the output layer

$$f(x) = g_1(x) \vee ... \vee g_N(x)$$

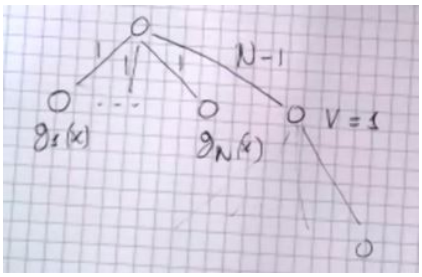

Figure 1.12:

The function compute by the output node is :

$$sgn(w(0)^T v(0)) = \sum_{i=1}^{N} g_i(x) w_{i,0} + N - 1$$

where $\sum_{i=1}^{N} g_i(x) w_{i,0}$ is equal to $-N+1$ to obtain 1.

$$x \; s.t \; f(x) = -1 \qquad g(x) = -1 \qquad i = 1, ..., N$$

$$x \; s.t. \; f(x) = 1 \qquad \exists i : g_i(x) = 1$$

I can definetely use more than one layer since exponential is not good. Maybe you can use fewer node, still potentially able to generate a classifier.

Theorem
$\forall d \in \mathbb{N}$ let $s(d)$ be the smaller integer
s.t. $\exists \; G = (V, E)$ with $|V| = s(d)$
s.t. $F_{G,sgn}$ contains all functions $f : \{-1, 1\}^d \to \{-1, 1\}$ then $|V| = \Omega(2^{\frac{d}{3}})$
If inputs are not binary we will use a sigmoid function. So 1 hidden layer is enought to approximate all function of the form:

$$f : [-1, 1]^d \to [-1, 1]$$

If you want to learn everything you need an exponential number of nodes.

1.2 Deep Learning

I have many hidden layers, not too many nodes in each layer.
I am choosing G with a specific profile (tall and skinny network).

$$F_{G_{dept},\sigma} \quad compared\ with \quad F_{G_{fat},\sigma}$$

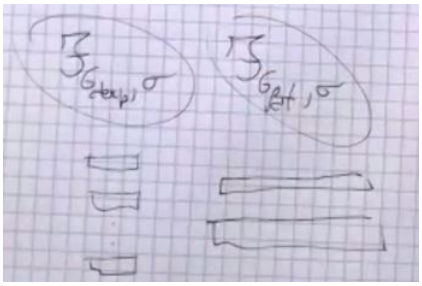

Figure 1.13:

Usually, the first one should be the better one.

$$|V'| > |V| \qquad G \to G' \qquad F_{G,\sigma}$$

Either increase size of 2 layer or add more layers:

$$|F_{G',\sigma}| \gg |F_{G",\sigma}|$$

where G' are new layers and G" are the fatter layers.

1.2.1 Convolution Neural Network

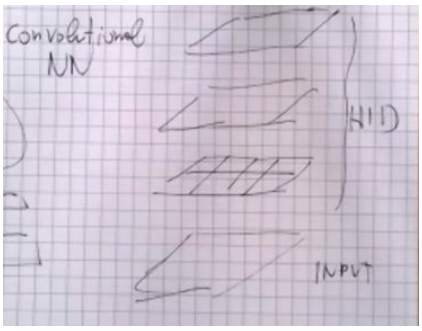

Figure 1.14:

== MANCAAAA FOTO GiUSTA ====

Using convolutional NN. We will recognise some shape (like presence of angles or simple geometric form obtained by combine the edge detector in the previous layers. If you have several layers you can have more more specific shapes combine the previous simpler layers.

1.2.2 General infos about Neural Networks

NN are trained using stochastic gradient descent.

$$W \qquad w_{ij} \leftarrow w_{ij} - \eta_t \frac{\eth \ell_{z_t}(w)}{\eth w_{ij}}$$

Z_t is randomly drawn from training set.

$$\ell_t(w) = \ell(f_{G,w,\sigma}(x_t), y_t) \quad \forall (i,j) \in E$$

We have $\nabla \ell_t(w)$
The gradient component $\frac{\eth \ell_{z_t}(w)}{\eth w_{ij}}$ $(i,j) \in \sqrt{s}$ is computed using a technique called "error backpropagation"
$\ell_t(w)$ is not convex! Gradient descent will not terminate in:

$$w^* = arg\min_w \hat{\ell}_s(w)$$

AutoML to determine hyperparameter!

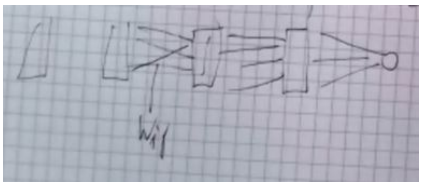

Figure 1.15: