Neural Network Introduction

Author: Ankit Gupta

1. Basics of neural network
2. Development of different models over time
3. Vectorize calculations
4. Construct a feed forward neural network

1) Different Human models

1) McCulloch & Pitts (MCP) Human Model (1943)

$x_i \in \{0,1\}$

$y_i \in \{0,1\}$

$g(x) = \sum_{i=1}^{n} x_i$ — sum aggregation of inputs

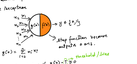
$f(g(x)) = \begin{cases} 1, & g(x) \geq \theta \text{ and no inhibition} \\ 0, & \text{otherwise} \end{cases}$ — activation function

$y_i = \text{have should not power}$

x_1	x_2	y_1	y_2
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

2) Rosenblatt's Perceptron (Improved of MCP Model)

Step function



$g(x) = \sum_{i=1}^{n} w_i x_i$

$f(g(x)) = \begin{cases} 1, & g(x) \geq \theta \\ 0, & \text{otherwise} \end{cases}$ — step function because output $\in \{0,1\}$ — threshold/bias

3) will bring inputs & outputs i.e. $x_i \to y_i \in \{0,1\}$

3) Removal of absolute weighing inputs

General Framework

$g(x) = \sum_{i=1}^{n} w_i x_i + b$

$f(g(x)) = f(\sum_{i=1}^{n} w_i x_i + b)$

For classification: $\hat{y} = \begin{cases} 1, & \text{if } f(g(x)) \geq \theta \\ 0, & \text{otherwise} \end{cases}$ — threshold

For Regression: $\hat{y} = f(g(x))$

* Now we can write computation as dot product

$\sum_{i=1}^{n} w_i x_i \Rightarrow \vec{w} \cdot \vec{x}$ = dot product of $w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}_{n \times 1}$ and $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n \times 1}$

* Matrix formulation for feed forward neural network

Looking at real life perceptron

We know the general framework for classification

$g(x) = \sum_{i=1}^{n} w_i x_i + b$

Note:- Here n represents the number of explanatory variables.

Don't confuse this with n = number of observations

$\therefore g(x) = \sum_{i=1}^{n} w_i x_i + b$

$f(g(x)) = f(\sum_{i=1}^{n} w_i x_i + b) = f(z)$

$f(z) \Rightarrow$ Activation function ; why is this important ; to introduce non-linearity

* Importance of activation function

1) Non-linearity → to introduce non linearity into the network.

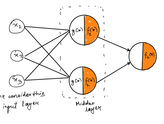
2) Control of output range → particularly useful when we want output in certain range

Example: Sigmoid, $f(z) = \frac{1}{1+e^{-z}}$; tanh $f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$; ReLU $f(z) = \max(0, z)$; Leaky ReLU $f(z) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x \leq 0 \end{cases}$; α above to 0

* Consider a 2 hidden layer neural network

(also called 2 layers if you consider output as the second layer)

(also called 3 layers if you consider input layer as first layer)



Here consider this as input layer

hidden layer

Consider a real world problem of determining whether a person will get heart attack.

Data might look like this

Age	Cholesterol level	Body fat (%)	Heart attack
37 (x1)	120 (x2)	1 (x3)	1
⋮	⋮	⋮	⋮

m rows, n+1 columns

Here we have 3 explanatory variables age, cholesterol level & body fat

Process

⇒ Each observation will be passed as input to all the neurons. This means each neuron will produce sumproduct of weights and inputs.

Bias will be added to each sumproduct in each neuron

⇒ The sumproduct will be passed through activation function

Hidden layers can have activation function such as ReLU, Leaky ReLU

Output layer can have following activation functions :-

Sigmoid function for binary classification

Identity function for Regression

Softmax for multilabel classification

Coming to the procedure,

after getting the output from hidden layer activation functions, this output is then as input to the last neuron

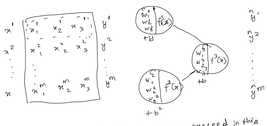
⇒ Again the output neuron will perform sumproduct of its input & weight. Bias will be added

⇒ This sumproduct will be passed through activation function of output layer

Final answer will be between 0 & 1.

⇒ We will assign the category as 1 if the final answer is more than 0.5 and 0 if final answer is less than 0.5

We have to do this for all the observations in the dataset



Looking at the figure, you can proceed in this

$$\begin{bmatrix} x_1^1 & x_2^1 & x_3^1 \\ x_1^2 & x_2^2 & x_3^2 \\ \vdots & & \\ x_1^m & x_2^m & x_3^m \end{bmatrix}_{m \times 3} \cdot \begin{bmatrix} w_1^1 & w_1^2 \\ w_2^1 & w_2^2 \\ w_3^1 & w_3^2 \end{bmatrix}_{3 \times 2} = \begin{bmatrix} \underbrace{x_1^1 w_1^1 + x_2^1 w_2^1 + x_3^1 w_3^1}_{\text{sumproduct of 1st neuron}} & \underbrace{x_1^1 w_1^2 + x_2^1 w_2^2 + x_3^1 w_3^2}_{\text{sumproduct of 2nd neuron}} \\ \vdots & \vdots \\ x_1^m w_1^1 + x_2^m w_2^1 + x_3^m w_3^1 & x_1^m w_1^2 + x_2^m w_2^2 + x_3^m w_3^2 \end{bmatrix}_{m \times 2} + \underbrace{\begin{bmatrix} b^1 & b^2 \\ b^1 & b^2 \\ \vdots & \vdots \\ b^1 & b^2 \end{bmatrix}}_{\text{Adding bias to each sumproduct}}$$

Here m = number of observations

n = 3 = number of inputs/features

$$\Rightarrow \underbrace{\begin{bmatrix} x_1^1 w_1^1 + x_2^1 w_2^1 + x_3^1 w_3^1 + b^1 & x_1^1 w_1^2 + x_2^1 w_2^2 + x_3^1 w_3^2 + b^2 \\ \vdots & \vdots \\ x_1^m w_1^1 + x_2^m w_2^1 + x_3^m w_3^1 + b^1 & x_1^m w_1^2 + x_2^m w_2^2 + x_3^m w_3^2 + b^2 \end{bmatrix}}_{\text{one bias for each neuron}}$$

⇒ this is fed to activation function for each neuron.

$$\begin{bmatrix} (S.P.)_1^1 & (S.P.)_2^2 \\ \vdots & \vdots \\ (S.P.)_m^1 & (S.P.)_m^2 \end{bmatrix}_{m \times 2} \rightarrow \begin{bmatrix} ReLU(S.P.)_1^1 & ReLU(S.P.)_1^2 \\ \vdots & \vdots \\ ReLU(S.P.)_m^1 & ReLU(S.P.)_m^2 \end{bmatrix}_{m \times 2} \begin{bmatrix} w_1^3 \\ w_2^3 \end{bmatrix}_{2 \times 1}$$

Continued.

Thus if we try to generalize, then the matrix formulation can be written as

Dataset dimension :-

Here

m → number of observations

n → no. of features

$$\begin{bmatrix} x_{11} & x_{12} \cdots x_{1n} \\ x_{21} & x_{22} \cdots x_{2n} \\ \vdots & \vdots \\ x_{m1} & x_{m2} \cdots x_{mn} \end{bmatrix}_{m \times n} ; \begin{bmatrix} w_1^1 & \cdots & w_1^D \\ w_2^1 & & w_2^D \\ \vdots & & \vdots \\ w_n^1 & & w_n^D \end{bmatrix}_{n \times D} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m \times 1}$$

* However, in field of machine learning, widely used convention is

Inputs $X \in \mathbb{R}^{n \times m}$, output $Y \in \mathbb{R}^{1 \times m}$; Predictions $\hat{Y} \in \mathbb{R}^{1 \times m}$