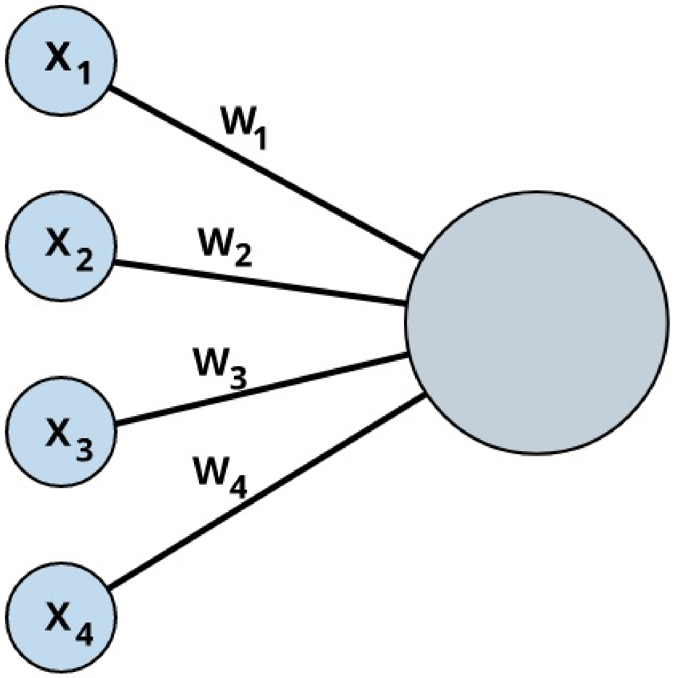
Table of Contents *generated with DocToc*

HOW IT WORKS - THEORY

Neurons

Neuraln netw ork consists, as the name indicates, of neurons. Each neuron takes an input form neurons connected to it. All inputs have their **weights** . Neuron sums inputs multiplied by their w eights, and applies activation function to this sum.



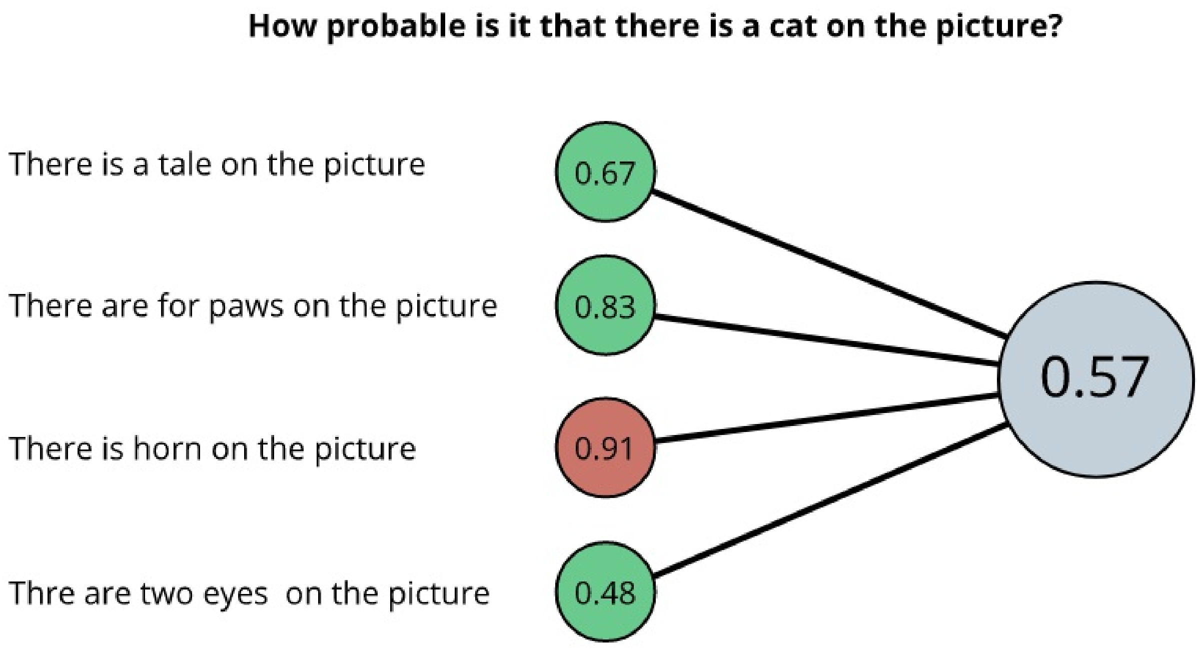
All **inputs** typically have a range 0.00 to 1.00 . Zero means that there is little to no chance of something and one meaning w e are almost sure that there is something

Weights have a range -1.00 to 1.00. -1 means that the inputs w ork against of the the thing w e are trying to predict and 1 means it acts in favour of it.

The goal of neral netw ork is to adjust its w eights so that it, w ith some certainty, can predict soulution form the inputs. If some input is more important and it acts in favour of some action it should have positive w eight close to 1. If it is not that important the w eight should be around 0. and if the input w orks against an action it should have negative w eight close to -1.

Example

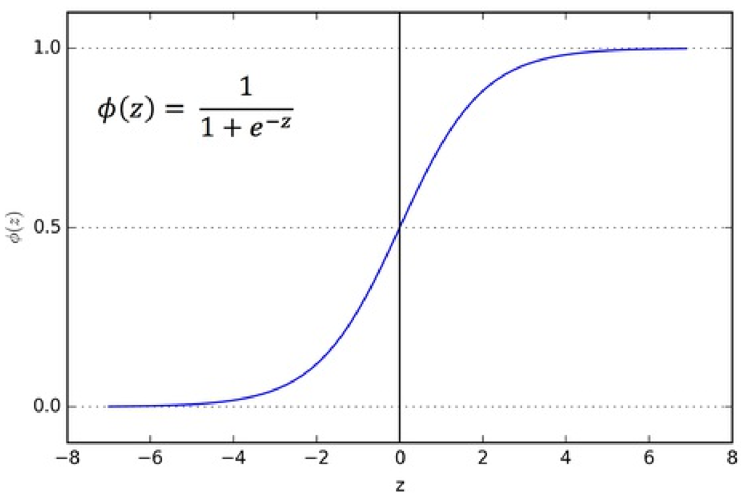
Let's consider an example below. We are trying to predict if there is a cat on a given picture. Previous layer of neurons have stated there is 67% probability that there is a tale on the image, 83% probalility that there are four paw s on the picture, 91% probalility that there is a horn and 48 % chance there are tw o eyes on the picture.



Of course w e know that cats have tails, four paw s and tw o eyes, but they don't have horns. That's w hy inputs **1**, **2** and **4** should have **positive** w eights. If w e know there is a tail, this should increase the chance that there is a cat. On the other hand input 3 should have negative w eight, because if w e know there is a horn, w e can be sceptical about image presenting a cat. If w e have show n our neural netw ork thousand of images of cats w e can hope that it w ill learn that cats don't have horns and put negative w eight in the input w hose role w as to deduce if there is a horn. How ever w e can't know for sure w hat neural netw ork picked as important and w hat it didn't, all inputs and outputs are just numers that somehow lead to the solution

Activation Function

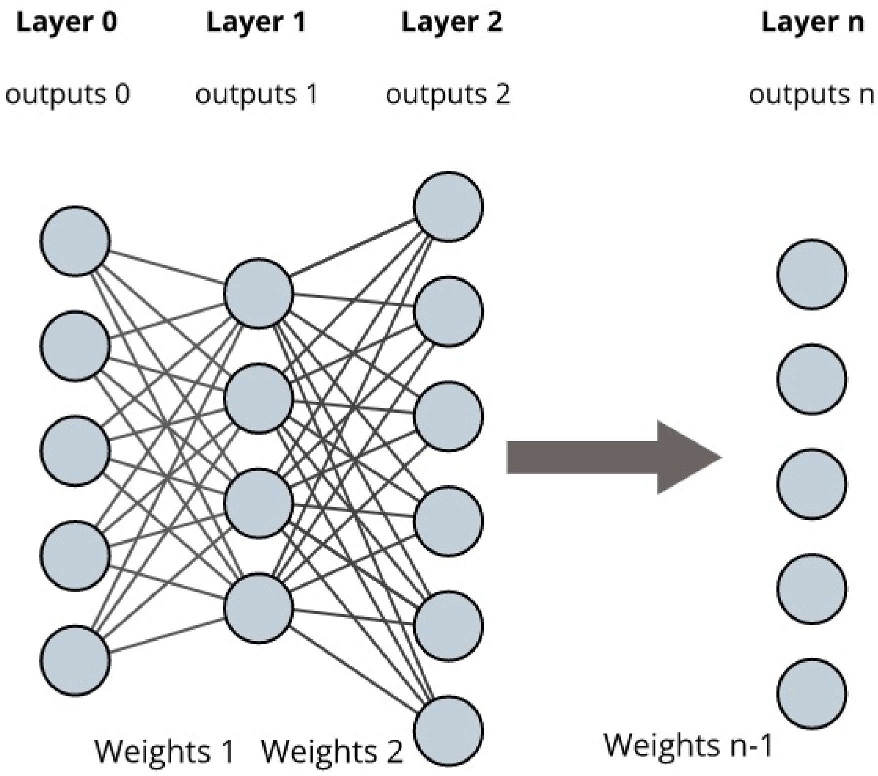
The sum of inputs multiplied by their w eights can often be larger than 1 or less than 0, that's w hy w e need something called **activation function**. Activation fuction applied to the output limits it to the range (0.00,1.00). An example of such funtion is **sigmoid **.



As the argumenst approch infinity the function aproches 1, as they approach minus nfinity it approches 0 w ith the value of 0.5 at w hen the argument is equal to 0.

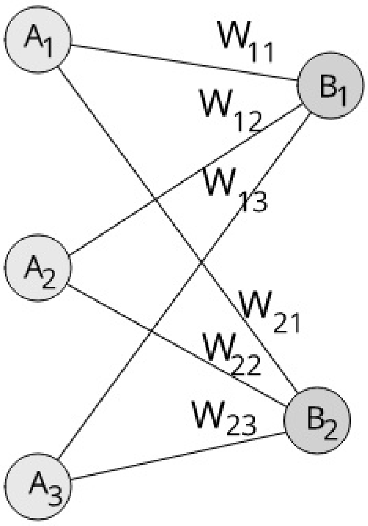
Layers

Full neural netw ork consists of not one but many neurons arragned in **layers**, each layer takes the output of the previous one and threats it as input



Weights

Let's consider an neural netw ork w ith 2 layers, firs one has 3 neurons and the second one has 2 neurons.



In this layer inputs to layer B should look like this:

$$InputB_1 = (A_1 \times W_{11}) + (A_2 \times W_{12}) + (A_3 \times W_{13})$$

$$InputB_2 = (A_1 \times W_{21}) + (A_2 \times W_{22}) + (A_3 \times W_{23})$$

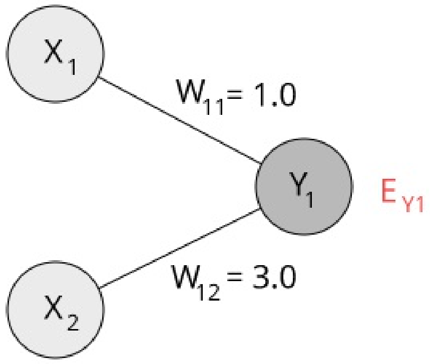
We can use **Matrix multiplication** to simplify this process

$$\begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \end{bmatrix} \bigcirc \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} (A_1 \times W_{11}) + (A_2 \times W_{12}) + (A_3 \times W_{13}) \\ (A_1 \times W_{21}) + (A_2 \times W_{22}) + (A_3 \times W_{23}) \end{bmatrix}$$

HOW DOES NEURAL NETWORK LEARN?

Errors

In order for neural netw ork to learn it first has to know about the mistakes it's making. Let's concider follow ing example:

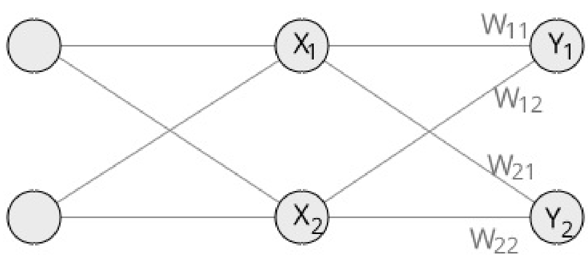


Lest's say w e guessed the value of Y1 but it's w rong and w e know the error. Let's call this error EY1. What should be the error of X1 or X2? One might say w e should split the errror of Y1 evenly becouse there are tw o nodes connected to Y1, so error of X1 should be 1/2 of error of Y1 and error of X2 should be 1/2 of error of Y1. Well it's generaly not a good w ey of doung this, beause w e can see that w eight comming from X2 is 3 times larger than the of of X1. Because of that X2 contributes to the mistake of Y1 3 times as much as X1. Therefore it w ould be better to describe the error of X1 as 1/4 the error of Y1 and error of X2 as 3/4 the error of Y1. Whis can lead to general principle:

$$ErrorX_1 = ErrorY_1 \times \frac{W_{11}}{W_{11} + W_{12}}$$

$$ErrorX_2 = ErrorY_1 \times \frac{W_{12}}{W_{11} + W_{12}}$$

Errors on layers



If w e consider the follow ing example then w e see that X1 and X2 contribute to not only the error of Y1 but also Y2. Therefore their error should come for both Y1 and Y2. If w e w rite it in equation it should look like this:

$$ErrorX_1 = ErrorY_1 \times \frac{W_{11}}{W_{11} + W_{21}} + ErrorY_2 \times \frac{W_{21}}{W_{11} + W_{21}}$$

$$ErrorX_2 = ErrorY_1 \times \frac{W_{12}}{W_{12} + W_{22}} + ErrorY_2 \times \frac{W_{22}}{W_{12} + W_{22}}$$

If we again write it in marices we would get something like this:

$$ErrorX = \begin{bmatrix} \frac{W_{11}}{W_{11}+W_{21}} & \frac{W_{21}}{W_{11}+W_{21}} \\ \frac{W_{12}}{W_{12}+W_{22}} & \frac{W_{22}}{W_{12}+W_{22}} \end{bmatrix} \bigcirc \begin{bmatrix} ErrorY1 \\ ErrorY2 \end{bmatrix}$$

We can then ommint the values in the denominator. If we do this we only loose the information about the normalization of the errors not the information about how much each weight contributs to it.

$$ErrorX = \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{bmatrix} \bigcirc \begin{bmatrix} ErrorY1 \\ ErrorY2 \end{bmatrix}$$

We can now see then the weight matrix is the same one what we have been using for to feed forward values between the layers, exept now it's transposed what means it is flipped along the diagonal line. And if you think about it, it makes perfect sence, we are now going backawrds instead of forward what means we need to invert the matrix:

$$\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}^T = \begin{bmatrix} W_{11} & W_{21} \\ W_{12} & W_{22} \end{bmatrix}$$

We can now write the general priciple:

$$Errors_{n-1} = (Weights_{n-1})^T \bigcirc Errors_n$$

With this alghoritm we can desribe errors from the end to the beggigng of the neural network

Learning

Ok we now know the erros but how can we adjust the weight to correct them?

HOW IT WORKS - PYTHON

works in progress

INSTALATION

Requirements

Simple neuron requires libraries :
Numpy
Scipy

to install them you can simply do it with pip:

```
pip install numpy
```

```
pip install scipy
```

Installing SimpleNueron

To install SimpleNeuron you all you have to do is copy the SimpleNeuron.py file to your directory

USAGE

First you need to imporst NeuralNetw ork object

```
from SimpleNeuron import NeuralNetwork
```

then you need to initialize neural netw ork object, it takes 2 parameters as input table consisting of numbers of neurons in each layer and learning rate, for example:

```
nn = NeuralNetwork([3,6,8],0.3)
```

w ill create a neural netw ork w ith 3 layears consisting, as follow s 3,6,8 neurons w ith learning rate of 0.3
Then you w ill have to train your netw ork on some data for this you use:

```
nn.learn(input,output)
```

finally w hen your netw ork is trained you can use predict function to predict output based on input:

```
nn.predict(input)
```