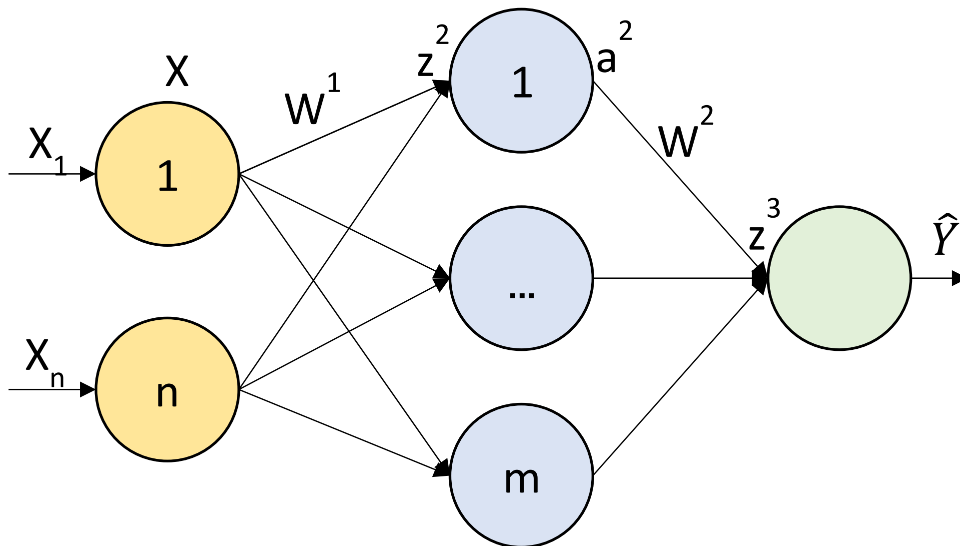
Neural Networks – First contact

1. Introduction

In the next illustration, an Artificial Neural Network is displayed. We can see how we can forward our input data (X) into the NN and get a prediction for our output $\hat{Y}$.



As we have the actual value that the prediction should have, we can compute the error using what is called a 'Cost Function'. With these, we can see how responsible are each neuron for that error, and update their values in order to decrease that error with the time (actually, with the number of iterations over our input vector).

Just for easier visualization before writing down the equations, let's move that NN to matrix notation:

X_1	…	X_n

z_1^2	…	z_m^2

a_1^2	…	a_m^2

$\hat{Y}$

$W_{1,1}^1$	…	$W_{1,m}^1$
…	…	…
$W_{n,1}^1$	…	$W_{n,m}^1$

$W_{1,1}^2$
…
$W_{1,m}^2$

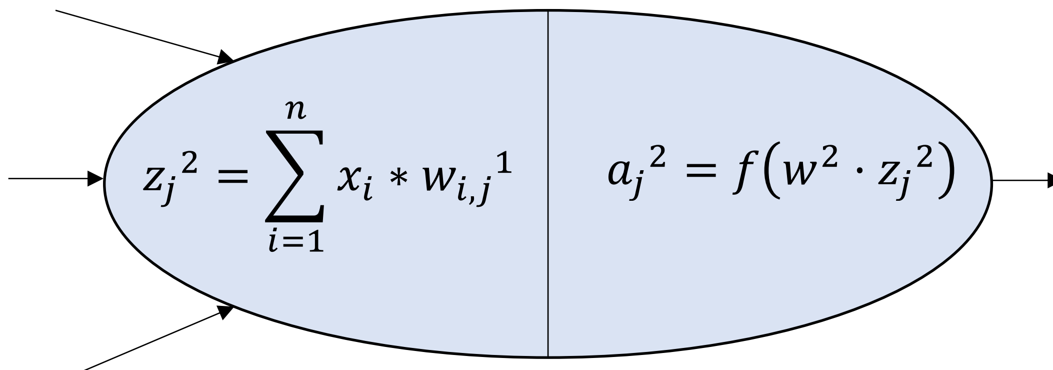
Note that we are using n and m to refer that it could be any number, as well as it can also be more hidden layers. Just for simplification, we will go over the NN illustrated, which means 1 hidden layer, $n = 2, m = 3$.

In the next table, we can see also a table with the notation we are using in this document:

Symbol	Variable	Range
X_i	Input i	$i \in (1, 2, \dots, n)$
${W_i}^1$	Input neuron i	$i \in (1, 2, \dots, n)$
${W_j}^2$	Hidden neuron j	$j \in (1, 2, \dots, m)$
$w_{i,j}$	Connection between ${W_i}^1 \rightarrow {W_J}^2$	$i, j \in \{(1, 1), \dots, (n, m)\}$
z^2	Net input to hidden neuron j	$j \in (1, 2, \dots, m)$
a^2	Net output from hidden neuron j	$j \in (1, 2, \dots, m)$
z^3	Net input to output neuron	$j \in (1, 2, \dots, m)$
$\hat{Y}$	Output of the NN (Prediction)	Integer

2. Feed Forward

Now we have all this information, we can develop the mathematical approach over our NN. Just before it, also for easing the visualization, let's see what is going on inside a neuron, for example our 1st hidden neuron.



According to the illustration, the neuron is adding all the inputs multiplied by their corresponding weight, and then applied an activation function to give the output to the next step.

The whole set of equation describing the NN is therefore:

$$\mathrm{z}^2 = X * W^1 \qquad \text{(Eq. 1)}$$

$$\mathrm{a}^2 = f(\mathrm{z}^2) \qquad \text{(Eq. 2)}$$

$$\mathrm{z}^3 = \mathrm{a}^2 * W^2 \qquad \text{(Eq. 3)}$$

$$\hat{Y} = f(\mathrm{z}^3) \qquad \text{(Eq. 4)}$$

With these equations, we complete the forward method that we will define in our Neural_Network class (we will start introducing Python notation also to be more familiar when looking at the code).

The Cost Function then, will be computed using that output of the NN and the real value (note this is supervised learning). There are several ways of computing this function, depending on your problem. This time, we will try to minimize the RMSE (Root Mean Squared Error), which correspond with the next formula:

$$J = \frac{1}{n}\sum_{t=1}^{n}\left(\hat{Y}_t - Y_t\right)^2 \qquad \text{(Eq. 5)}$$

Note than this could be expressed depending on the rest of our parameters:

$$J = \frac{1}{n}\sum_{t=1}^{n}(f(f(X * W^1) * W^2) - Y_t)^2 \qquad \text{(Eq. 6)}$$

3. Backward Propagation

Now, it is the moment to propagate backward in the NN the error made, and tell the weights how much responsible are them for that error, so they can update themselves in order to continuously decrease that error.

Thus, we want to know how much is the value of $\frac{dJ}{dW^1}$ and $\frac{dJ}{dW^2}$.

If we first develop the gradient correspond to the hidden layer weights:

$$\frac{dJ}{dW^2} = \frac{d}{dW^2}\left\{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{Y}_t - Y_t\right)^2\right\} = C^1 \cdot \frac{d\left(\hat{Y}_t - Y_t\right)^2}{dW^2} = C^2 - \left(\hat{Y}_t - Y_t\right) \cdot \frac{d\hat{Y}_t}{dW^2}$$

Now, if we forget for a second about the constant term, by applying the chain rule to the remaining derivative term:

$$\frac{dJ}{dW^2} = -\left(\hat{Y}_t - Y_t\right)\frac{d\hat{Y}_t}{dz^3} \cdot \frac{dz^3}{dW^2} =^1 - \left(\hat{Y}_t - Y_t\right) \cdot f'(z^3) \cdot \frac{dz^3}{dW^2} =^2 (a^2)^T \cdot \delta^3 \qquad \text{(Eq. 7)}$$

1–The prediction $\hat{Y}_t$ is a function of z^3, so it can be directly derived.
2–The prediction $\hat{Y}_t$ can be plot against W^2 as a straight line with slope a^2. Mathematically, this is represented as the transpose of that vector. Also, we have grouped the remaining terms into the δ^3, which is called the 'Back Propagating Error'.

And, now following again the chain rule to next equation, we can compute the remaining gradient:

$$\frac{dJ}{dW^1} = \frac{d}{dW^1}\left\{\frac{1}{n}\sum_{t=1}^{n}\left(\hat{Y}_t - Y_t\right)^2\right\} = C^1 \cdot \frac{d\left(\hat{Y}_t - Y_t\right)^2}{dW^1} = C^1 - \left(\hat{Y}_t - Y_t\right) \cdot \frac{d\hat{Y}_t}{dW^2}$$

$$\begin{aligned}\frac{dJ}{dW^1} &= -\left(\hat{Y}_t - Y_t\right)\frac{d\hat{Y}_t}{dz^3} \cdot \frac{dz^3}{dW^1} = -\left(\hat{Y}_t - Y_t\right) \cdot f'(z^3) \cdot \frac{dz^3}{dW^1} = \delta^3 \cdot \frac{dz^3}{dW^1} \\ &= \delta^3 \cdot \frac{dz^3}{da^2} \cdot \frac{da^2}{dW^1} =^1 \delta^3 \cdot (W^2)^T \cdot \frac{da^2}{dW^1} \\ &= \delta^3 \cdot (W^2)^T \cdot \frac{da^2}{dz^2} \cdot \frac{dz^2}{dW^1} =^2 \delta^3 \cdot (W^2)^T \cdot f'(z^2) \\ &\cdot \frac{dz^2}{dW^2} =^3 X^T \cdot \delta^3 \cdot f'(z^2) =^4 X^T \cdot \delta^2\end{aligned} \qquad \text{(Eq. 8)}$$

1–The input z^3 can be plot against a^2 as a straight line with slope $W^2 \rightarrow (W^2)^T$
2–The input a^2 is a function of z^2, so it can be directly derived
2–The input z^3 can be plot against a^2 as a straight line with slope $\mathrm{X} \rightarrow X^T$

4. Learning

So far, we have explained how the forward step done to get a prediction, then how to use the real values to calculate the value of the Cost Function, and how using backpropagation (partial derivatives + chain rule) we can tell each neuron how responsible they are for that error. But now, how does our NN learn? Well, the same thing as we do: try, fail, learn. These steps are what we call: to train.

Thus, we have covered the try (forward) and the fail (Cost) steps. Now, the step left, learn, is accomplished by updating the value of the weights. Each node (or neuron) is going to update its last value, following the next equation:

$$W^i = W^i - \lambda_i \cdot \frac{dJ}{dW^i} \qquad \text{(Eq. 9)}$$

That parameter λ_i is called the 'Learning Rate'. Thus, as dJ/dW^i is the error committed because of W^i, we are exactly telling our neuron to correct its own value given the mistake committed by their fault multiplied by this rate.

This rate is of course, a value between (0, 1), where 0 would mean something like, don't learn. If the value of the learning rate is too high (usually start with 0.01 or 0.001), the processes of learning could not converge, and if the value is too low, it can take forever. Just like in real life! We want to learn as fast as possible, but we know that we need to go slow in the beginning to assimilate all the information we didn't know, until we feel confident to go on and try with new information. This information, is what our NN call data.

(*) Eq. 9 is the easiest way to explain the training process. There are many ways to optimize (actually minimize) the value of the cost function. LINK is a good place to get more info!

5. Testing

The process of learning is repeated over and over until we consider we have learnt. But, what does 'have learnt mean'? Just like studying for an exam, we usually do all the problems in the book collection, until we ensure that we are able to solve all of them. But then the exam comes. In the exam, there is (usually) data that you have not worked with, you don't know what you are going to find!

But, if you have studied enough, you are pretty sure that if you apply you forward( ) function to the exam data, you will get an A+ with the outputs you'll get (your answers).

The behavior of NN are exactly the same. It trains until it is confident to forward new data. (Be careful with overfitting, training too much, which is really well explained LINK). The training can be defined then 2 ways:

- You can stablish a threshold for the Cost, and let the NN train until the value of the Cost is below that threshold.
    - Advantage: you ensure your desired final Cost.
    - Disadvantage: play carefully with the learning rate, you don't know how much time it can take to train.

- You can stablish the number of iterations you want your NN to train with the whole dataset. These iterations are usually called 'epochs'. Another term you should familiarize with is 'batch size'. This is used to update the weights every several inputs (the value of the batch size). So for example, is the batch size is 5, you will calculate the error with the same NN structure of 5 input observation, the calculate the mean (for instance), and then apply backward propagation with that error to update the weights.
    - Advantage: if you run 1 epoch with a defined batch size, you can calculate the time it takes and then calculate how much it will take the number of epochs you would like your NN to iterate over.
    - Disadvantage: you don't know beforehand which value of the Cost you will reach.

6. Conclusion

I hope this has been fun! This is just the start of the Neural Networks world! Now you can go to GH LINK to see how to code this from scratch, or go to GH LINK to see how it can be implemented using the classic frameworks like TensorFlow, Keras, PyTorch…

I highly recommend you start programming from scratch and make sure you understand everything! Then you will be ready to save time using the libraries, but with the certain that you understand what is under the hood.

Anything you need, just leave the comment, and if you liked it, please recommend it! Until next time, enjoy Neural Networks!!