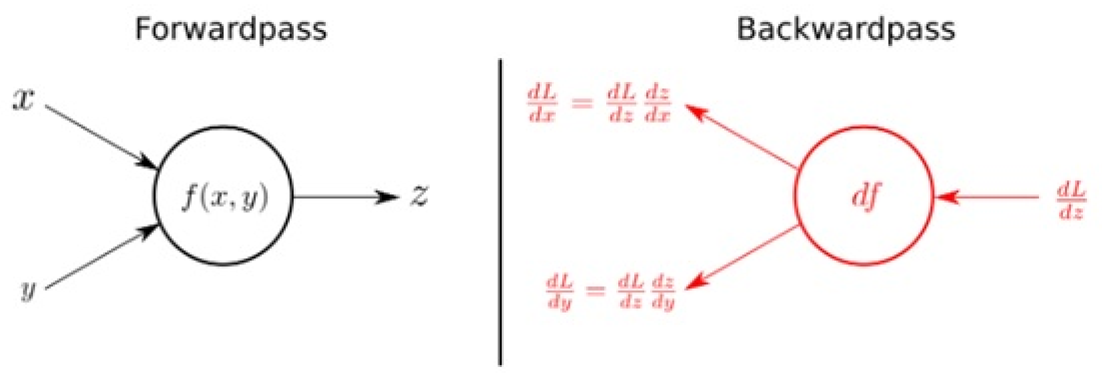
Forward Propagation and Backward Propagation


Forwardpass
Backwardpass
x
y
$f(x,y)$
z
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
df
$\frac{dL}{dz}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$

Let's have a quick recap about the previous assignment:

So in the previous assignment you saw about all the building blocks of neural network like Neurons, Layers, Synapses, Weights, biases, activation functions and loss functions.

These are the basic components which are very much important for building any kind of neural network . So make sure you have understood everything about those components. If you have not, then go through that assignments once again before moving further in this assignment.

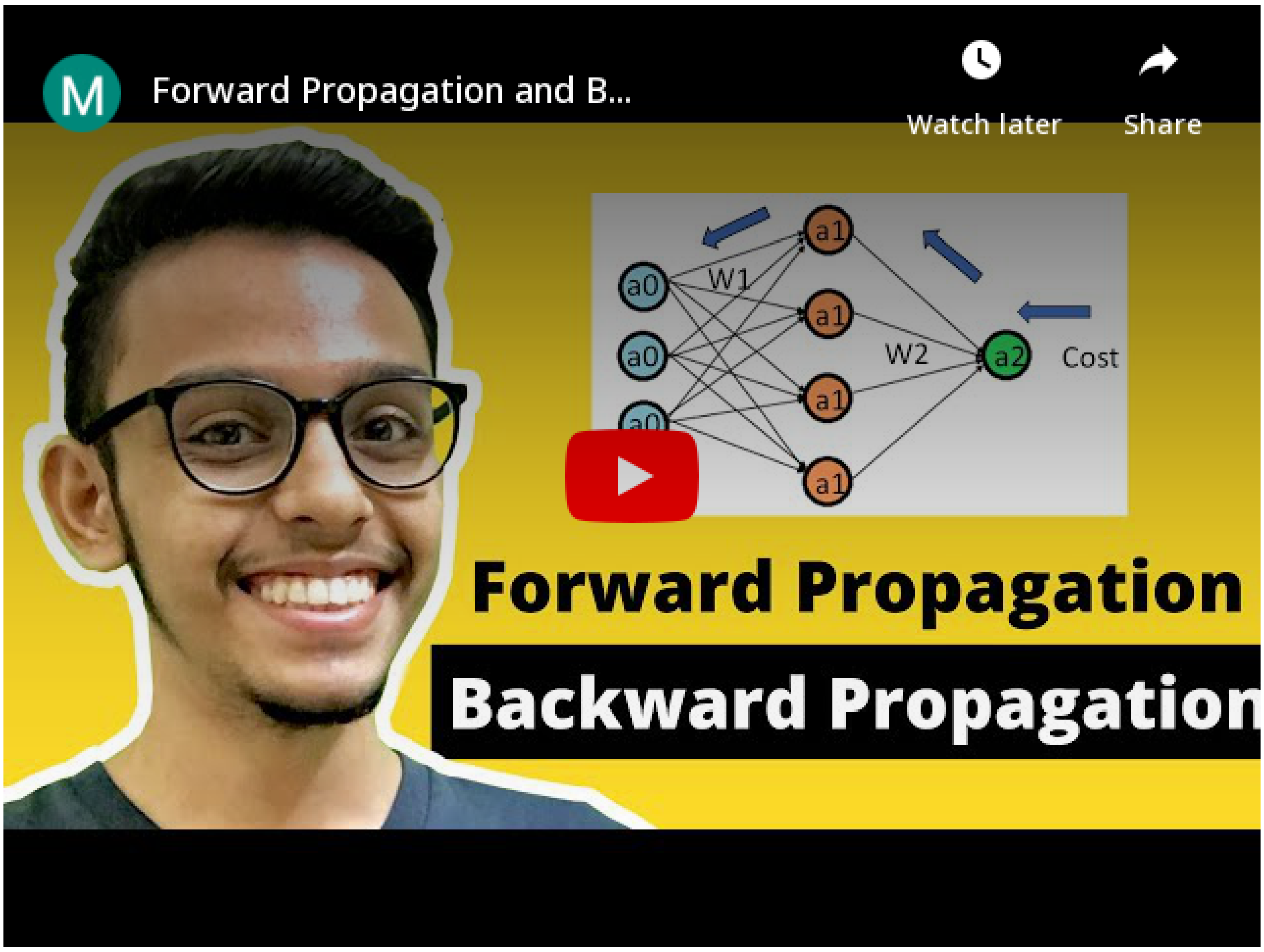
So now in this assignement we are going to see what exactly **Forward Propagations** and **Backward Propagations** are?

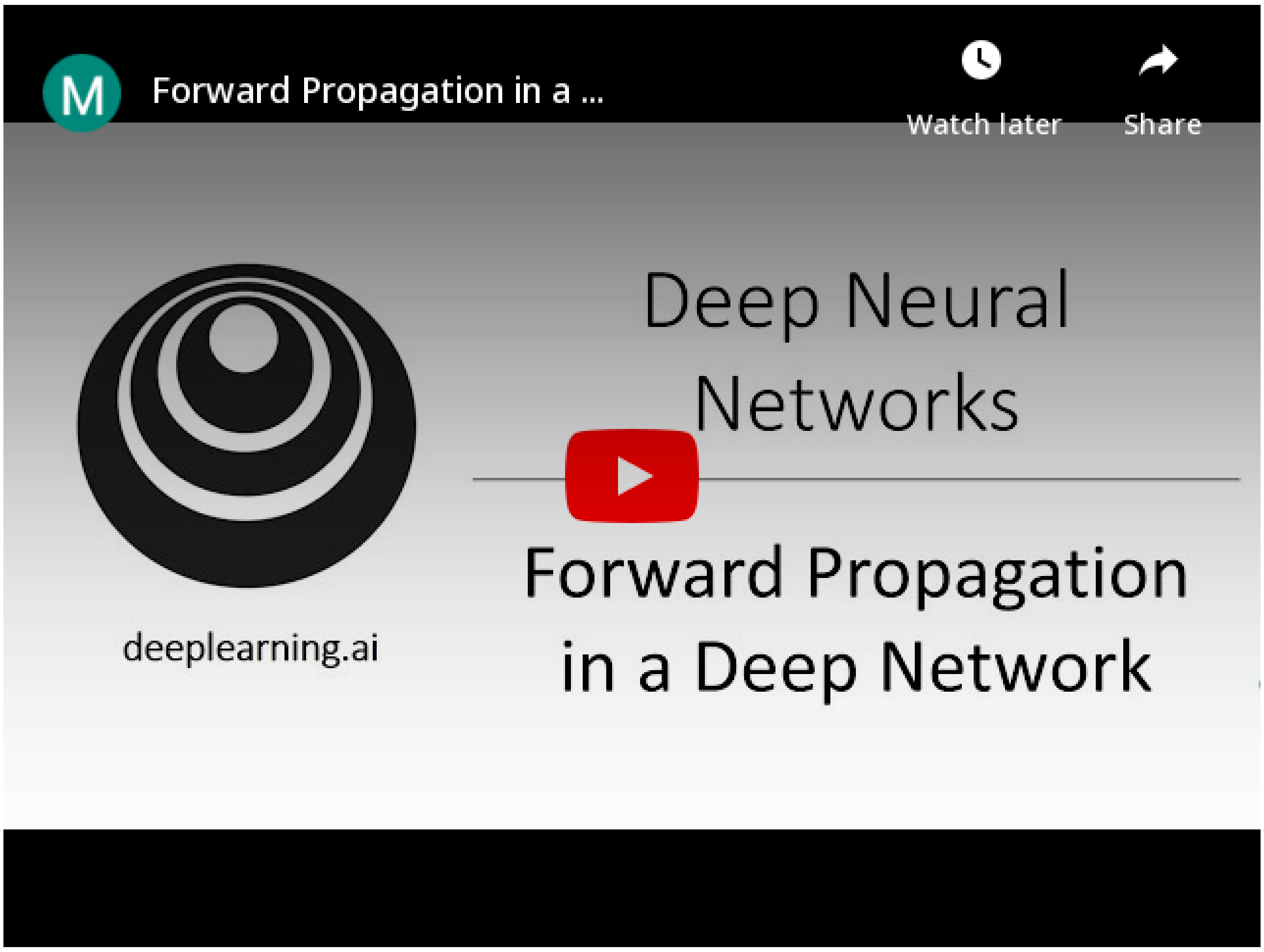
So firstly what is this term **Forward Propagtion**?

Well, if you break down the words, forward implies moving ahead and propagation is a term for saying spreading of anything. forward propagation means we are moving in only one direction, from input to the output, in a neural network. Think of it as moving across time, where we have no option but to forge ahead, and just hope our mistakes don't come back to haunt us.

Also as the name suggests, the input data is basically fed in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer.

Watch the below mentioned videos inorder to understand what exactly forward propagation is?


Forward Propagation and B...
Watch later
Share
a0
a0
a0
W1
a1
a1
a1
a1
W2
a2
Cost
Forward Propagation
Backward Propagation


Forward Propagation in a ...
Watch later
Share
deeplearning.ai
Deep Neural Networks
Forward Propagation in a Deep Network

One of the first neural networks used the concept of forward propagation.

Let's see the basic intution behind forward propagation with the help of a simple equation of line.

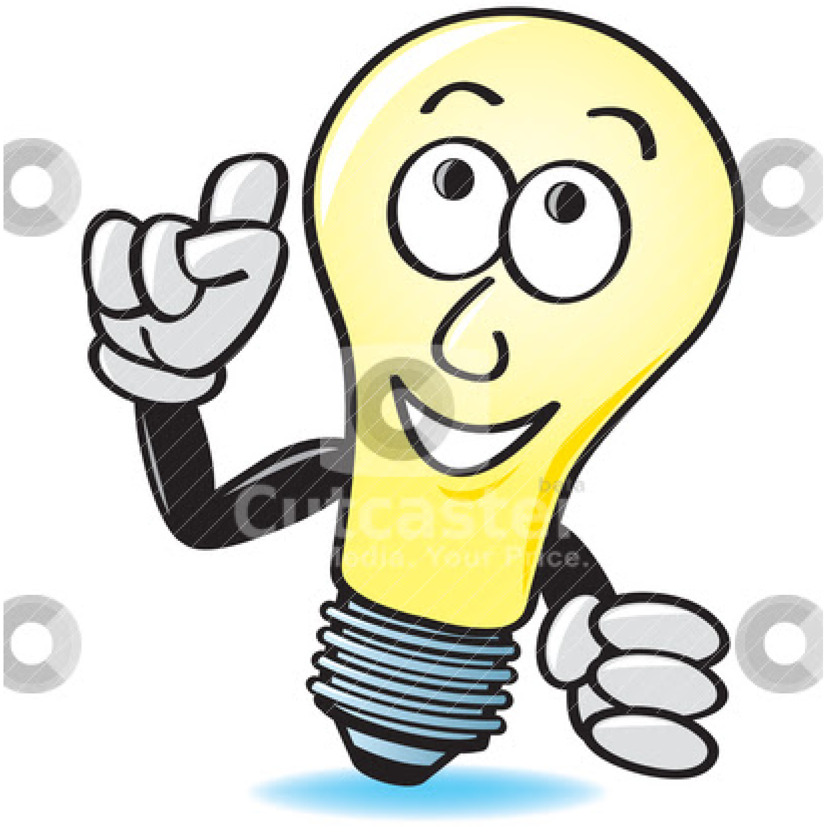
Now, as we all know that a line can be represented with the help of the equation:

y = mx + c

Where **y** is basically the y coordinate of the point, m is the slope, **x** is the x coordinate and **c** is the y-intercept i.e., the point at which the line crosses the y-axis.

But why are we getting into the line equation here? This will basically help us, understand components of a neural network in detail.

So do you remember how we said neural networks are supposed to mimic the thinking process of humans. Well, let’s just assume that we do not know the equation of a line, but we do have a graph paper and draw a line randomly on it.



For the sake of this example, you drew a line through the origin and when you saw the x and y coordinates, they looked like this:

X coordinate	Y coordinate
1	3
2	6
3	9
4	12
5	15

This looks familiar. If I asked you to find the relation between x and y, you would directly say it is y = 3x. But let us go through the process of how forward propagation works.

We will basically assume here x is the input and y is the output.

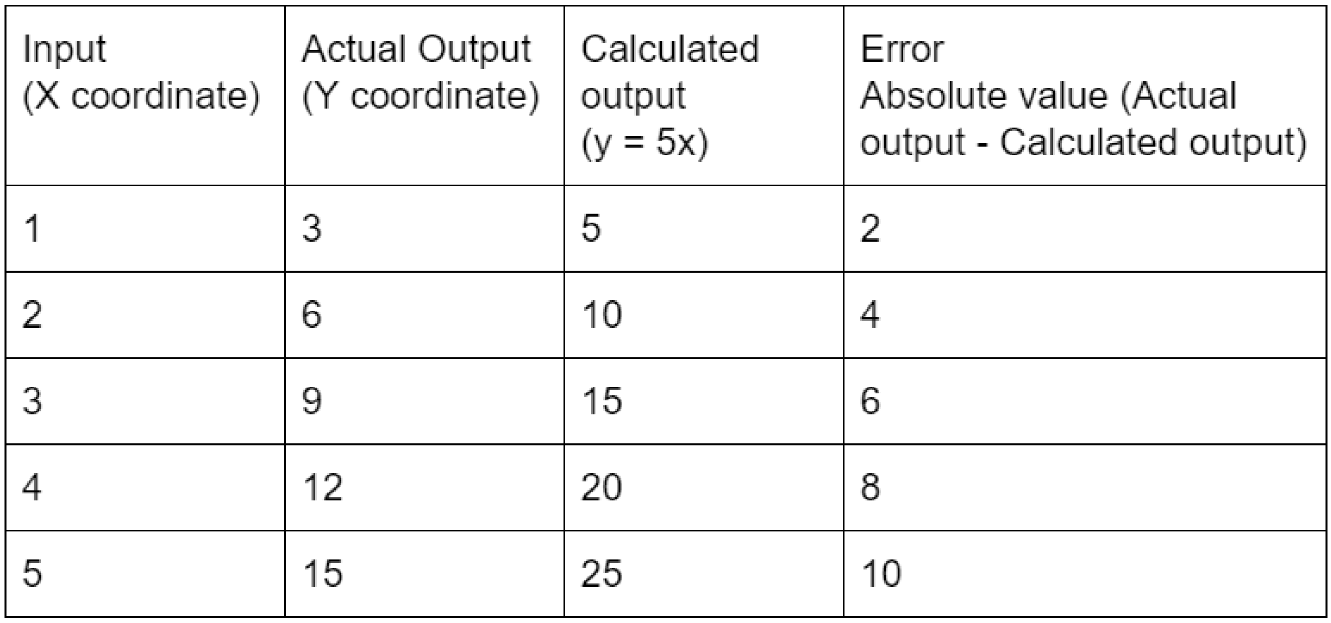
The first step here is the initialisation of the parameters. We will guess that y must be a multiplication factor of x. So we will assume that y = 5x and see the results then. Let us add this to the table and see how far we are from the answer.

Input (X coordinate)	Output (Y coordinate)	Random guess (y = 5x)
1	3	5
2	6	10
3	9	15
4	12	20
5	15	25

Note that taking the number 5 is just a random guess and nothing else. We could have taken any other number here. I should point out that here we can term 5 as the weight of the model.

All right, this was our first attempt, now we will see how close (or far) we are from the actual output.

One way to do that is to use the difference of the actual output and the output we calculated. We will call this the error. Here, we aren't concerned with the positive or negative sign and hence we

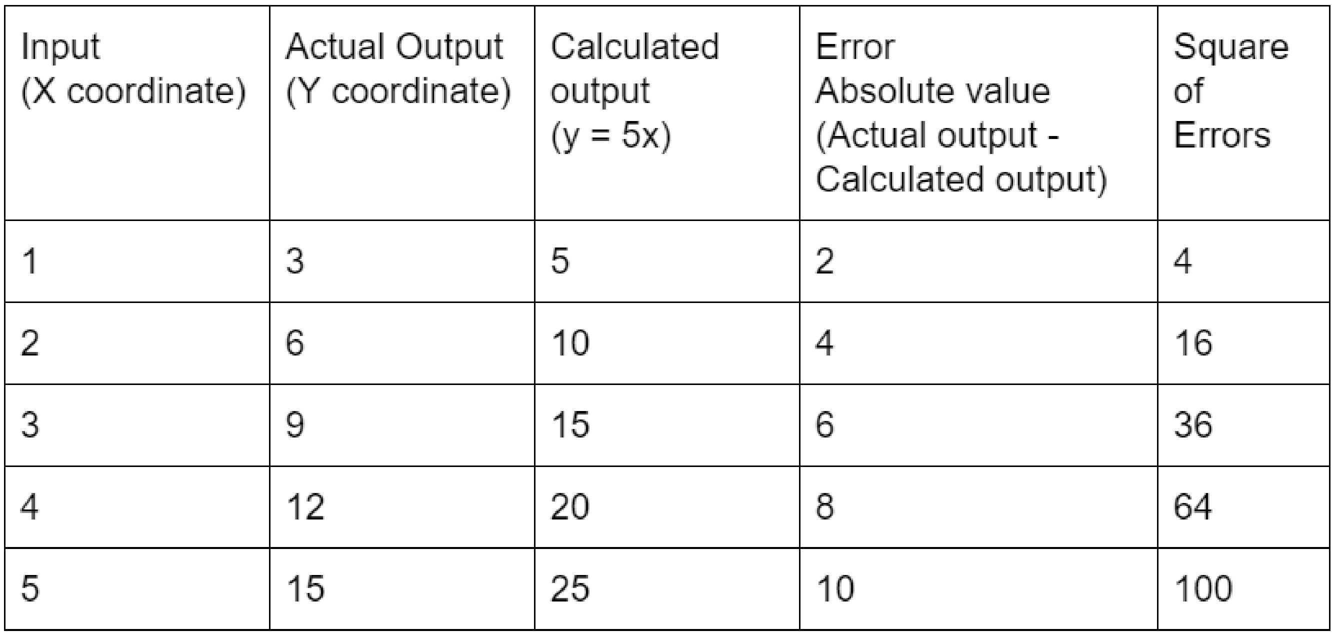


Input (X coordinate)	Actual Output (Y coordinate)	Calculated output (y = 5x)	Error Absolute value (Actual output - Calculated output)
1	3	5	2
2	6	10	4
3	9	15	6
4	12	20	8
5	15	25	10

If we take the sum of this error, we get the value 30. But why did we total the error? Since we are going to try multiple guesses to come to the closest answer, we need to know how close or how far we were from the previous answers. This helps us refine our guess and calculate the correct answer.

Wait. But if we just add up all the error values, it feels like we are giving equal weightage to all the answers. Shouldn't we penalise the values which are way off the mark? For example, 10 here is too high than 2. It is here that we introduce the somewhat famous "Sum of squared Errors" or SSE for short. In SSE, we square all the error values and then add them. Thus, the error values which are very high get exaggerated and thus helps us in knowing how to proceed further.

Let's put these values in the table below



Input (X coordinate)	Actual Output (Y coordinate)	Calculated output (y = 5x)	Error Absolute value (Actual output - Calculated output)	Square of Errors
1	3	5	2	4
2	6	10	4	16
3	9	15	6	36
4	12	20	8	64
5	15	25	10	100

Now the SSE for the weight 5 (Recall that we assumed y = 5x), is 145. We call this the loss function. The loss function is important to understand the efficiency of the neural network and also helps us when we incorporate backpropagation in the neural network.

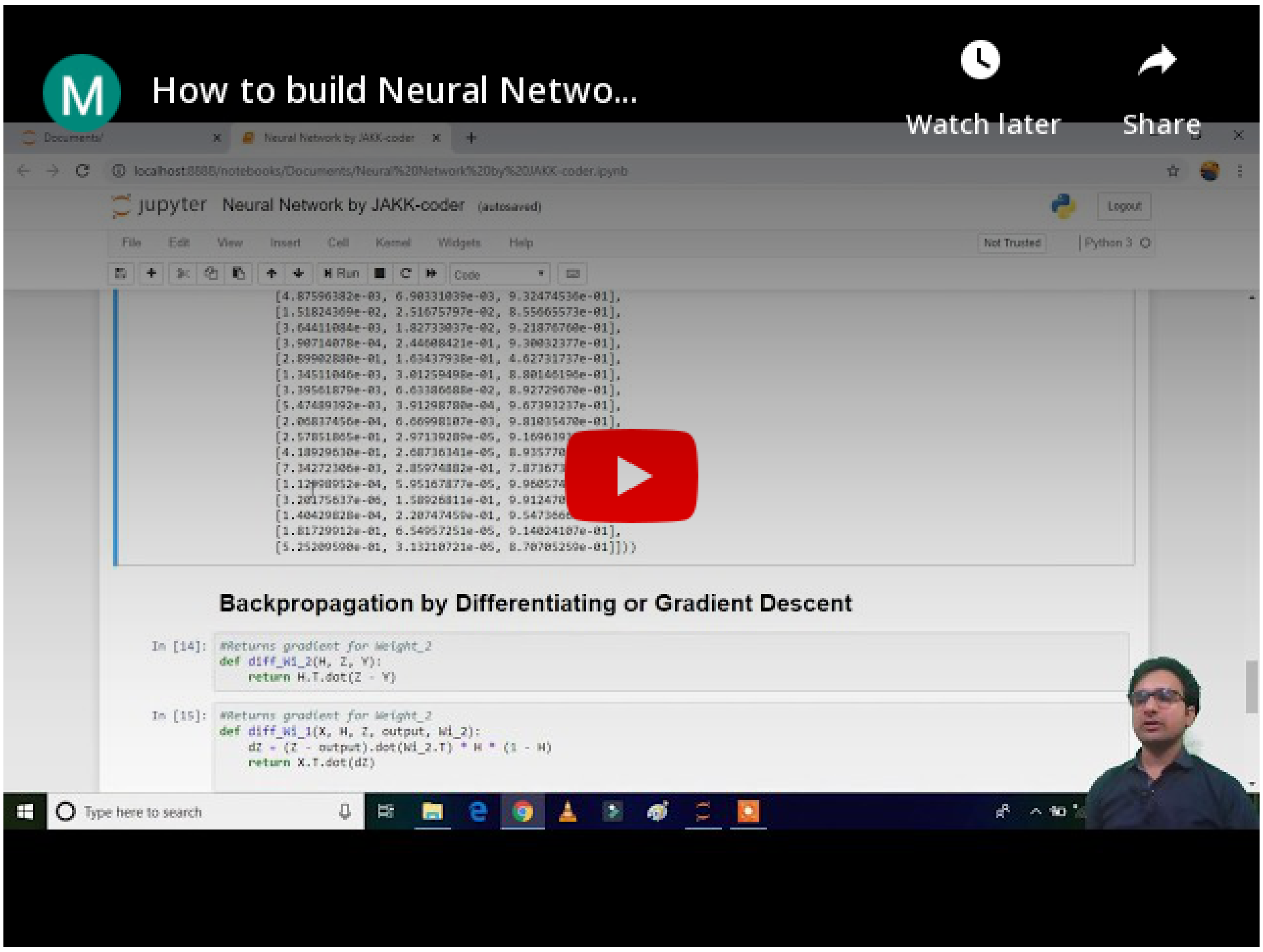
All right, so far we understood the principle of how the neural network tries to learn. We have also seen the basic principle of the neuron. Let us now understand forward propagation in the neural network itself.

Components of Forward Propagation model

↳ *2 cells hidden*

LET'S START CODING

Watch the below mentioned video before getting into code


How to build Neural Netwo...
Watch later
Share
jupyter Neural Network by JAKK-coder (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Not Trusted
Python 3
Logout
Type here to search

```
[4.87596382e-03, 6.90331039e-03, 9.32474536e-01],
[1.51824369e-02, 2.51675797e-02, 8.55665573e-01],
[3.64411084e-03, 1.82733037e-02, 9.21876760e-01],
[3.90714078e-04, 2.44608421e-01, 9.30032377e-01],
[2.89902880e-01, 1.63437938e-01, 4.62731737e-01],
[1.34511046e-03, 3.01259498e-01, 8.80146196e-01],
[3.39561879e-03, 6.63386688e-02, 8.92729670e-01],
[5.47489392e-03, 3.91298780e-04, 9.67393237e-01],
[2.06837456e-04, 6.66998107e-03, 9.81035470e-01],
[2.57851865e-01, 2.97139289e-05, 9.169619
[4.18929630e-01, 2.68736141e-05, 8.935770
[7.34272306e-03, 2.85974882e-01, 7.873673
[1.12998952e-04, 5.95167877e-05, 9.960574
[3.20175637e-06, 1.58926811e-01, 9.912470
[1.40429828e-04, 2.20747459e-01, 9.547366
[1.81729912e-01, 6.54957251e-05, 9.14024187e-01],
[5.25209590e-01, 3.13210721e-05, 8.70705259e-01]]))
```

Backpropagation by Differentiating or Gradient Descent

```
In [14]: #Returns gradient for Weight_2
         def diff_Wi_2(H, Z, Y):
             return H.T.dot(Z - Y)

In [15]: #Returns gradient for Weight_2
         def diff_Wi_1(X, H, Z, output, Wi_2):
             dZ = (Z - output).dot(Wi_2.T) * H * (1 - H)
             return X.T.dot(dZ)
```

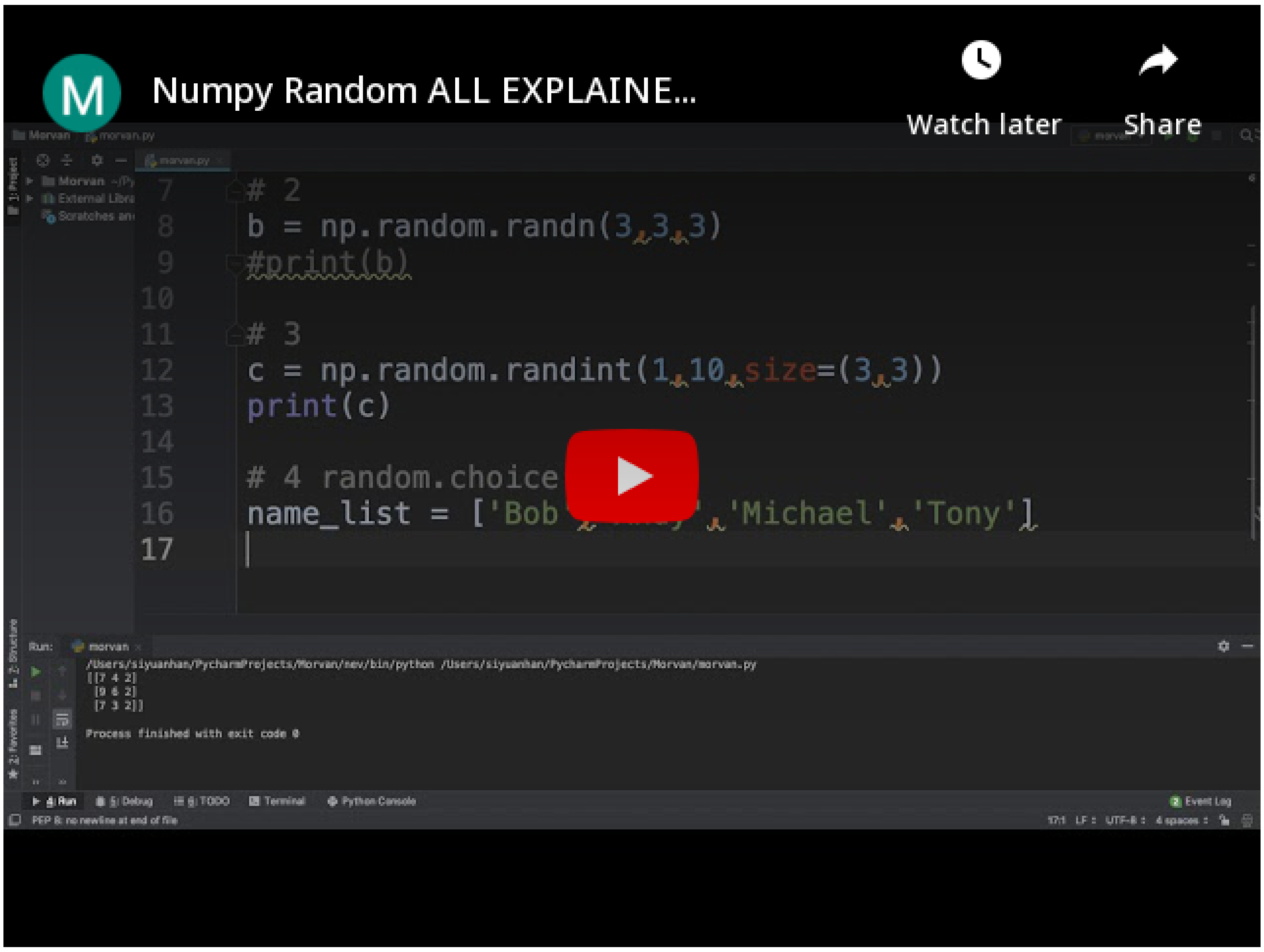
↳ *9 cells hidden*

Mention the number of neurons in the **Hidden Layer**

NOTE: There is only one hidden layer

[ ] ↳ *3 cells hidden*

Watch the below mentioned video before moving ahead



```
# 2
b = np.random.randn(3,3,3)
#print(b)

# 3
c = np.random.randint(1,10,size=(3,3))
print(c)

# 4 random.choice
name_list = ['Bob'  'Michael','Tony']
```

```
/Users/siyuanhan/PycharmProjects/Morvan/nev/bin/python /Users/siyuanhan/PycharmProjects/Morvan/morvan.py
[[7 4 2]
 [9 6 2]
 [7 3 2]]

Process finished with exit code 0
```

So now the next step is basically to generate weights with random values. So for the sake of generating random values you need use **random** module of numpy and the **randn()** function of the **random** module. It will be written as:

```
np.random.randn()
```

NOTE:

1. Generate the first weight with random values which will be a 3 dimensional array and each subarray will be of 5 elements i.e 5 dimensional layer

Let's understand this with a simple example:

weight 1:
[[1,2,3,4,5]
[1,2,3,4,5]
[1,2,3,4,5]]

weight 1 will be for input layer to hidden layer

[ ] ↳ *6 cells hidden*

Now as you generated the random values for first weighrt, generate the values for the second weight in a similar manner.

NOTE: The second weight is for **Hidden Layer** to **Output Layer**

[ ] ↳ *10 cells hidden*

So what the heck is this Back Propagation?

↳ *3 cells hidden*

Watch the below mentioned video before moving ahead

Let's see how Backprop algorithm basically work.

↳ *7 cells hidden*

What is the need of Backpropagation?

[ ] ↳ *14 cells hidden*

Please fill the below feedback form about this assignment

https://forms.zohopublic.in/cloudyml/form/CloudyMLDeepLearningFeedbackForm/formperma/VCFbldnXAnbcgAIl0lWv2blgHdSldheO4RfktMdgK7s