1. Back- propagation is the main substance of neural networking. Appropriate tuning of the weights, which are obtained within the previous iteration, guarantees lower error rates. We will have to take a look at loss functions, optimization functions, etc., in order to form this instance as subjective as possible. We are going to be creating or training a neural network that can perform an XOR, exclusive OR, functionality. We all know how the XOR table looks like,

A	B	result
0	0	0
0	1	1
1	0	1
1	1	0

In order to answer this question, we will first look at the front propagation and then move forward by using the results and prove it with back propagation.

We will also select an activation function that determines the activation value at every node within the neural net. We will start by choosing an identity function which might look like this, f(a) = a. With every neural training's identity function, comes a hypothesis function. In our case, we will have a hypothesis function that determines what the input to the activation function is. It will look something like this: $h(X) = W_0X_0 + W_1X_1 + W_2X_2$ or $h(X) = sigma(W * X)$ for all Ws (weights) and Xs (inputs). Other things that can be considered is the loss function to be the standard cost function of logistic regression. The main goal is to always adjust the weights to urge a lower loss than the one we currently have, and we will use the Batch Gradient Descent optimization function to achieve that. Finally, the training rate are going to be 0 or 1 and every weight is going to be initialized to 1.

We are now going to draw a diagram of our neural network and start explaining one training iteration using the back-propagation algorithm. There are three layers to it, the leftmost, middle, and the output layer. The leftmost layer is that the input layer, which takes X_0 because the bias term useful 1, and X_1 and X_2 as input features, and the output layer has just one output unit D_0 whose activation value is that the actual output of the model (i.e. h(x)). We will now forward-propagate to understand more. It is now the time to feed-forward the knowledge from one layer to subsequent. Here are the two steps that happen at every node within the network:

   a. Getting the weighted sum of inputs of a specific unit using the h(x) function we defined earlier.
   b. Using the value we get from step 1 into the activation function which we defined as f(a)=a for our case and using the activation value we get which is the output of the activation function. Note that units X_0, X_1, X_2 and Z_0, where Z_0 is the bias term that the middle layer takes, don't have any units connected to them and because

of that, the steps mentioned above don't occur in those nodes. Here is the math for what we just explained,

Unit Z_1:

$$
\begin{aligned}
h(x) &= W0.X0 + W1.X1 + W2.X2\ (defined\ above) \\
&= 1.1 + 1.0 + 1.0 \\
&= 1 \\
&= a \\
\\
z &= f(a) \\
&= a => z \\
&= f(1) \\
&= 1
\end{aligned}
$$

and same goes for the remainder of the units:
Unit Z_2 and Z_3:

$$
\begin{aligned}
h(x) &= W0.X0 + W1.X1 + W2.X2\ (defined\ above) \\
&= 1.1 + 1.0 + 1.0 \\
&= 1 \\
&= a \\
\\
z &= f(a) \\
&= a => z \\
&= f(1) \\
&= 1
\end{aligned}
$$

Again, as explained above, all the Zs are the biases for the middle layer.
Unit D_0:

$$
\begin{aligned}
h(x) &= W0.Z0 + W1.Z1 + W2.Z2 + W3.Z3 \\
&= 1.1 + 1.1 + 1.1 + 1.1 \\
&= 4 = a \\
\\
z &= f(a) \\
&= a => z \\
&= f(4) = 4
\end{aligned}
$$

We can say that it doesn't give us accurate results because we expected the answer to be 1 but it gave us 4 instead. Looking at the above results, our model predicted an output of 1 for the set of inputs {0, 0}. Calculating the loss/cost of the present iteration would follow:

$$
\begin{aligned}
Loss &= actual\_results - predicted\_results \\
&= 0 - 4
\end{aligned}
$$

$= -4$

The actual_results value comes from the training set, while the predicted_results value is what our model presented. Therefore, the cost at this iteration is close to -4.

Now the main question here is, how is back-propagation used here? We'd basically like to seek out the loss at every node within the neural network because every loss that arrives to deep learning model is really the sum of all losses that was caused by all the nodes. Therefore, we'd like to seek out which node is liable for most of the loss in every layer as the whole point is to find out have the least amount of weights. This is exactly how back-propagation works. We do the calculation step at every node, backpropagating the loss into the neural network, and check the loss value for each node. In order to answer that in more detail, we will study this. In our case, we have a model that doesn't give accurate predictions which means that its weights haven't been used yet. Back-propagation is all about feeding the loss backwards and eventually see which weights supported which. In order to prove that, we are going to use the optimization function. This will be the same as explained above,

We mention in the forward propagation that the subsequent functions look like this: f(a) = a. Thus, in back propagation we will do the partial derivation of that function. Hence,

$f'(a) = 1$
$J'(w) = Z\ x\ delta$ (Z is just the value from the activation function)

We will have to calculate the delta which will lead us to our final results.

All that's left now's to update all the weights we've within the neural net and that will be achieved by traversing through it. In our case, we will just have one traversal/iteration, and we will get the weights/loss that we wanted.

Putting everything together, we can say that the model is not completely trained as we only did one iteration. The whole point of using back propagation in this case was to minimize the loss or minimize the weights or basically penalize what node is causing the most amount of loss.

1. Self-regulated learning enables us to misuse of a variety of labels that accompany the data. The inspiration is very direct. Creating a dataset with clean labels is costly yet unlabeled information is being produced constantly. To utilize this a lot bigger measure of unlabeled information, one route is to set the learning goals appropriately to get supervision from the information itself.

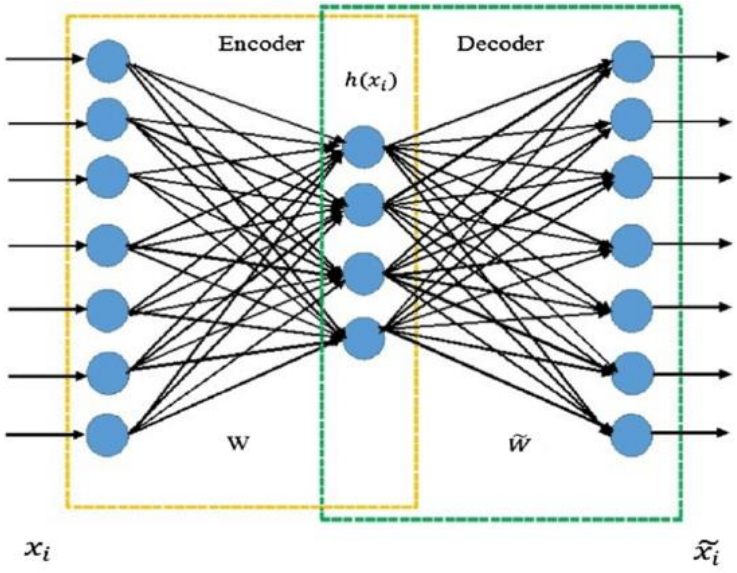
   Oneself directed task, otherwise called pretext task, guides us to a regulated supervised loss function. Be that as it may, we typically couldn't care less about the last execution

of this designed task. Or maybe we are keen on the scholarly middle of the road portrayal with the desire that this portrayal can convey great semantic or basic implications and can be helpful to an assortment of functional downstream tasks.

For instance, we may pivot pictures indiscriminately and train a model to anticipate how each information picture is turned. The pivot forecast task is made-up, so the real exactness is insignificant, similar to how we treat assistant tasks. In any case, we anticipate that the model should learn great dormant factors for real tasks, for example, building an article acknowledgment classifier with not very many marked samples. Broadly, all the generative models can be considered as self-directed, however with various objectives: Generative models center around making assorted and reasonable pictures, while self-supervised representation learning care about creating great features commonly supportive for some undertakings. An example is presented here,

Autoencoders:

Autoencoder is a neural system which attempts to reproduce the information, accordingly, learning input information portrayals. This is an image found online:



The system takes in a picture and yields a similar picture. The loss for the most part utilized is the MSE loss. For our case, we can prepare an autoencoder on our unlabeled information, load the encoder's loads and train on the marked subset. That solves our problem

3. The video “final” is attached which plays the original and swapped videos side by side. All the extraction data, fsw file, main and reference videos, images, logs etc. is saved in deepFake_output.zip folder. I screen recorded both videos from YouTube and tried to replace Jimmy Fallon’s face with Obama’s face! Since, I didn’t have enough time, I

couldn’t get the entire face to be replaced but tried my best to finish it in the given time frame