```
# scaling function
def scaling(df):
    scaled_data = df/255.0
    return scaled_data

my_data= scaling(train_data)
my_data.shape

# Just tried relu function
def relu(x):
    return np.maximum(x, 0, x)

#Let us create dummy variables for the labels
def dummy(labels):
    dff = pd.get_dummies(labels)
    return dff

dummy_labels = dummy(train_labels)

# initialize bias units
def initialize_bias(df):
    input_bias_unit = np.ones((len(df),1))
    output_bias_unit = np.ones((len(df),1))
    return input_bias_unit,output_bias_unit

input_bias_unit,output_bias_unit = initialize_bias(my_data)

# add_bias_to the dataframe
def add_bias(df):
    bias_unit = np.ones((len(df),1))
    biased = np.c_[bias_unit,df]
    return biased

# initialize weights
def initialize_weights(num_hidden_neurons,num_inputs,num_outputs):
    input_weights = np.random.randn(num_hidden_neurons,num_inputs) *0.05
    output_weights = np.random.randn(num_outputs,num_hidden_neurons)*0.05
    input_bias_weight = np.zeros((1,num_hidden_neurons))
    output_bias_weight = np.zeros((1,num_outputs))
    return input_weights,output_weights,input_bias_weight,output_bias_weight

# calculates the sigmoid
def sigmoid(x):
    return 1.0/(1.0 + np.exp(-x))

# let us initialize the weights
input_weights,output_weights,input_bias_weight,output_bias_weight = initialize_weights
(350,784,10)

def randomize(df,df_l):
    df2 = df.reindex(np.random.permutation(df.index))
    df_l2 = df_l.reindex(np.random.permutation(df_l.index))
    return df2, df_l2

# forward propagation
def forward_prop
(my_data,input_bias_weight,input_weights,output_bias_weight,output_weights):
    #add bias unit to the input vector. Activation of other layers is also dene in
the same manner
    i1 = add_bias(my_data)
    hidden_values = np.dot(i1,np.c_[input_bias_weight.transpose
(),input_weights].transpose())
```

```
    s1 = relu(hidden_values)
    h1 = add_bias(s1)
    output_value = np.dot(h1,np.c_[output_bias_weight.transpose
(),output_weights].transpose())
    s2 = sigmoid(output_value)
    return hidden_values,s1,output_value,s2

# calculates errors of the network at every stage
def error(dummy_labels,s2,):
    total_er = (dummy_labels-s2)**2/2
    output_error = s2*(1-s2)*(dummy_labels-s2)
    hidden_layer_error = hidden_values*(1-hidden_values)*np.dot
(output_error,output_weights)

    return total_er,output_error,hidden_layer_error


# iterating the forward and backward propagation process 1000 times to see how much
the weights are updated.
for i in range(1,1001):
    random.seed(i)
    my_data,dummy_labels= randomize(my_data,dummy_labels)

    # forward propagate
    # calling the forward propagation function and storing the values in the variables
    hidden_values,s1,output_value,s2=forward_prop
(my_data,input_bias_weight,input_weights,output_bias_weight,output_weights)
    # calculates the errors for each iteration and stores them
    total_er,output_error,hidden_layer_error = error(dummy_labels,s2)

    # updation output_weights
    output_weights = output_weights+ 0.02*np.dot(output_error.transpose(),s1)/31000
    #input_weights
    input_weights = input_weights+ 0.02* np.dot(hidden_layer_error.transpose
(),my_data)/31000
    # output bias ka update
    output_bias_weight = output_bias_weight + 0.02*np.dot(output_bias_unit.transpose
(),output_error)/31000
    # input bias
    input_bias_weight = input_bias_weight + 0.02*np.dot(input_bias_unit.transpose
(),hidden_layer_error)/31000
    #print(i)
```