Suppose we have some information about obesity, smoking habits, and exercise habits of five people. We also know whether these people are diabetic or not. Our dataset looks like this:

Person	Smoking	Obesity	Exercise	Diabetic
Person 1	0	1	0	1
Person 2	0	0	1	0
Person 3	1	0	0	0
Person 4	1	1	0	1
Person 5	1	1	1	1

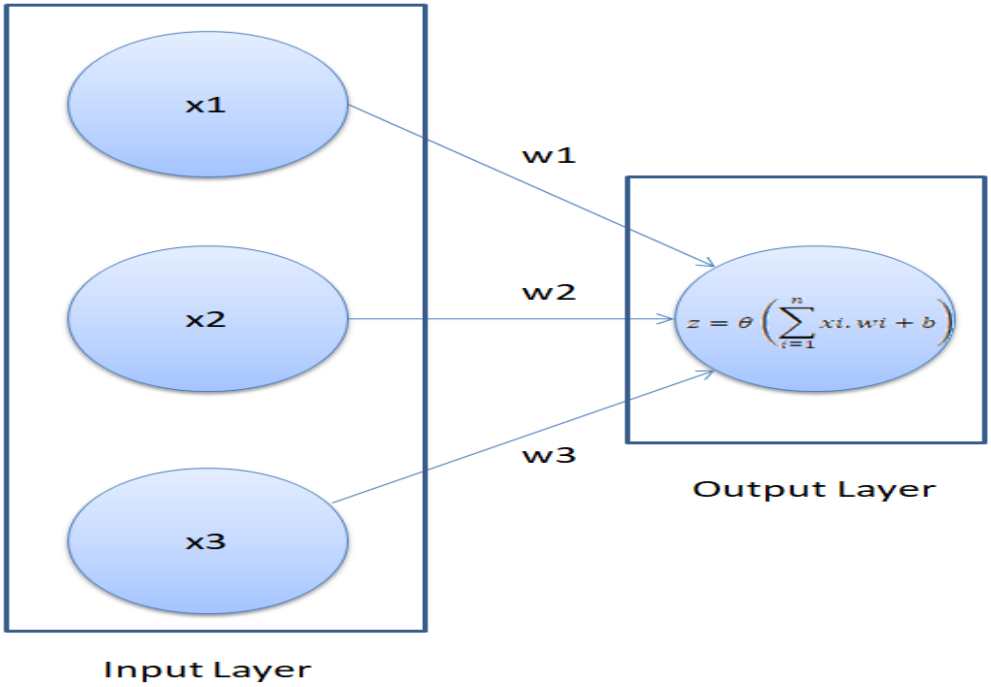
In the above table, we have five columns: Person, Smoking, Obesity, Exercise, and Diabetic. Here 1 refers to true and 0 refers to false. For instance, the first person has values of 0, 1, 0 which means that the person doesn't smoke, is obese, and doesn't exercise. The person is also diabetic.

It is clearly evident from the dataset that a person's obesity is indicative of him being diabetic. Our task is to create a neural network that is able to predict whether an unknown person is diabetic or not given data about his exercise habits, obesity, and smoking habits. This is a type of supervised learning problem where we are given inputs and corresponding correct outputs and our task is to find the mapping between the inputs and the outputs.

Neural Network Theory

For instance, in our example our independent variables are smoking, obesity and exercise. The dependent variable is whether a person is diabetic or not.

In the beginning, the neural network makes some random predictions, these predictions are matched with the correct output and the error or the difference between the predicted values and the actual values is calculated. The function that finds the difference between the actual value and the propagated values is called the cost function. The cost here refers to the error. Our objective is to minimize the cost function. Training a neural network basically refers to minimizing the cost function. We will see how we can perform this task.



A neural network executes in two steps: Feed Forward and Back Propagation

Feed Forward

In the feed-forward part of a neural network, predictions are made based on the values in the input nodes and the weights. If you look at the neural network in the above figure, you will see that we have three features in the dataset: smoking, obesity, and exercise, therefore we have three nodes in the first layer, also known as the input layer. We have replaced our feature names with the variable `x`, for generality in the figure above.

The weights of a neural network are basically the strings that we have to adjust in order to be able to correctly predict our output. For now, just remember that for each input feature, we have one weight.

The following are the steps that execute during the feedforward phase of a neural network:

Step 1: (Calculate the dot product between inputs and weights)

The nodes in the input layer are connected with the output layer via three weight parameters. In the output layer, the values in the input nodes are multiplied with their corresponding weights and are added together. Finally, the bias term is added to the sum. The `b` in the above figure refers to the bias term.

The bias term is very important here. Suppose if we have a person who doesn't smoke, is not obese, and doesn't exercise, the sum of the products of input nodes and weights will be zero. In that case, the output will always be zero no matter how much we train the algorithms. Therefore, in order to be able to make predictions, even if we do not have any non-zero information about the person, we need a bias term. The bias term is necessary to make a robust neural network.

Mathematically, in step 1, we perform the following calculation:

$$X.W=x1w1+x2w2+x3w3+b$$

Step 2: (Pass the result from step 1 through an activation function)

The result from Step 1 can be a set of any values. However, in our output we have the values in the form of 1 and 0. We want our output to be in the same format. To do so we need an activation function, which squashes input values between 1 and 0. One such activation function is the sigmoid function.

The sigmoid function returns 0.5 when the input is 0. It returns a value close to 1 if the input is a large positive number. In case of negative input, the sigmoid function outputs a value close to zero.

Mathematically, the sigmoid function can be represented as:

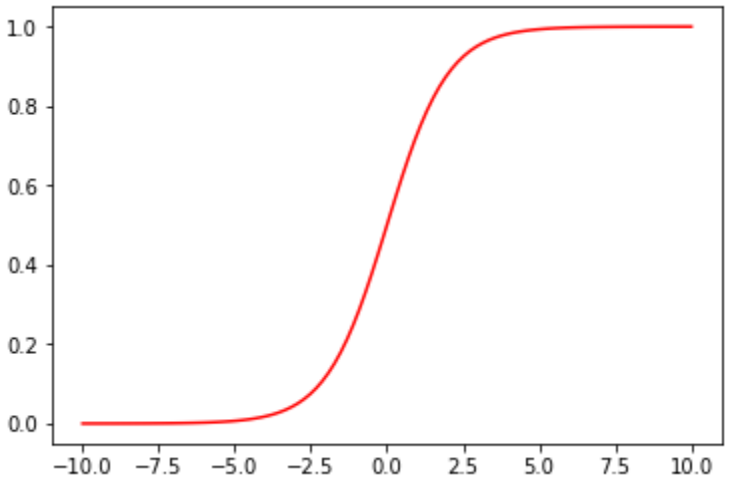
$$\theta_{X.W} = \frac{1}{1 + e^{-X.W}}$$

```
input = np.linspace(-10, 10, 100)

def sigmoid(x):
   return 1/(1+np.exp(-x))

from matplotlib import pyplot as plt
plt.plot(input, sigmoid(input), c="r")
```

we first randomly generate 100 linearly-spaced points between -10 and 10. To do so, we use the linspace method



You can see that if the input is a negative number, the output is close to zero, otherwise if the input is positive the output is close to 1. However, the output is always between 0 and 1.

Back Propagation

We start by letting the network make random predictions about the output. We then compare the predicted output of the neural network with the actual output. Next, we fine-tune our weights and the bias in such a manner that our predicted output becomes closer to the actual output, which is basically known as "training the neural network".

In the back propagation section, we train our algorithm.

Step 1: (Calculating the cost)

The first step in the back propagation section is to find the "cost" of the predictions. The cost of the prediction can simply be calculated by finding the difference between the predicted output and the actual output. The higher the difference, the higher the cost will be.

There are several other ways to find the cost, but we will use the mean squared error (MSE) cost function. A cost function is simply the function that finds the cost of the given predictions.

The mean squared error cost function can be mathematically represented as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(predicted - observed)^2$$

Here `n` is the number of observations.

Step 2: (Minimizing the cost)

Our ultimate purpose is to fine-tune the knobs of our neural network in such a way that the cost is minimized. If your look at our neural network, you'll notice that we can only control the weights and the bias. Everything else is beyond our control. We cannot control the inputs, we cannot control the dot products, and we cannot manipulate the sigmoid function.

In order to minimize the cost, we need to find the weight and bias values for which the cost function returns the smallest value possible. The smaller the cost, the more correct our predictions are.

This is an optimization problem where we have to find the function minima.

To find the minima of a function, we can use the gradient decent algorithm(GDA). The gradient decent algorithm can be mathematically represented as follows:

$$repeat\ until\ convergence: \left\{ w_j := w_j - \alpha \frac{\partial}{\partial w_j} J(w_0, w_1 \ldots\ldots w_n) \right\} \ldots\ldots\ldots\ldots (1)$$

Here in the above equation, J is the cost function. Basically what the above equation says is: find the partial derivative of the cost function with respect to each weight and bias and subtract the result from the existing weight values to get the new weight values.

The derivative of a function gives us its slope at any given point. To find if the cost increases or decreases, given the weight value, we can find the derivative of the function at that particular weight value. If the cost increases with the increase in weight, the derivative will return a positive value which will then be subtracted from the existing value.

On the other hand, if the cost is decreasing with an increase in weight, a negative value will be returned, which will be added to the existing weight value since negative into negative is positive.

In Equation 1, we can see there is an alpha symbol, which is multiplied by the gradient. This is called the learning rate. The learning rate defines how fast our algorithm learns.

We need to repeat the execution of Equation 1 for all the weights and bias until the cost is minimized to the desirable level. In other words, we need to keep executing Equation 1 until we get such values for bias and weights, for which the cost function returns a value close to zero.

We know that our cost function is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(predicted - observed)^2$$

We need to differentiate this function with respect to each weight. We will use the chain rule of differentiation for this purpose. Let's suppose "d_cost" is the derivate of our cost function with respect to weight "w", we can use chain rule to find this derivative, as shown below:

$$\frac{d\_cost}{dw} = \frac{d\_cost}{d\_pred}\frac{d\_pred}{dz}, \frac{dz}{dw}$$

Here,

$$\frac{d\_cost}{d\_pred}$$

can be calculated as:

$$2(predicted - observed)$$

Here, 2 is constant and therefore can be ignored. This is basically the error which we already calculated. In the code, you can see the line:

```
dcost_dpred = error # ........ (2)
```

Next we have to find:

$$\frac{d\_pred}{dz}$$

Here "d_pred" is simply the sigmoid function and we have differentiated it with respect to input dot product "z". In the script, this is defined as:

```
dpred_dz = sigmoid_der(z) # ......... (3)
```

Finally, we have to find:

$$\frac{d\_z}{dw}$$

z=x1w1+x2w2+x3w3+b

Therefore, derivative with respect to any weight is simply the corresponding input.