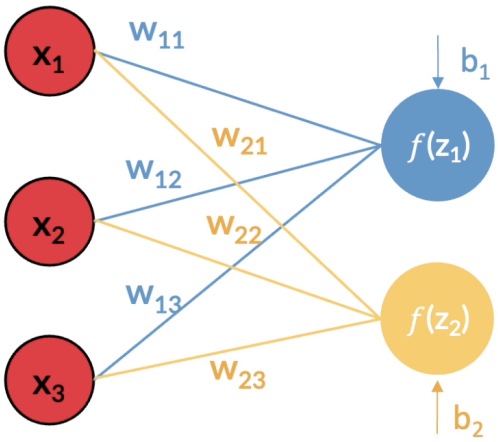
upGrad

Session 2: Feedforward in Neural Networks

Flow of information between layers

In artificial neural networks, the output from one layer is used as input to the next layer. Such networks are called **feedforward neural networks.** This means that there are no loops in the network, i.e., information is always fed forward, never fed backwards.

We know that the weight matrix between layer 0 (input layer) and layer 1 (the first hidden layer) is denoted by W. The dot product between the matrix W and the input vector x_i along with the bias vector b, i.e., $W*x_i + b$, acts as the cumulative input z to layer 1. The activation function is applied to this cumulative input z to compute the output h of layer 1.



Let's take the aforementioned example and perform matrix multiplication to get a vectorised method to compute the output of layer 1 from the inputs of layer 0.

Here, the following input is given.

$$x^i = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The dimensions of the input are (3,1).
There are two neurons in the first hidden layer. Hence, the cumulative input z^1 will be as given below.

$$z^1 = \begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix}$$

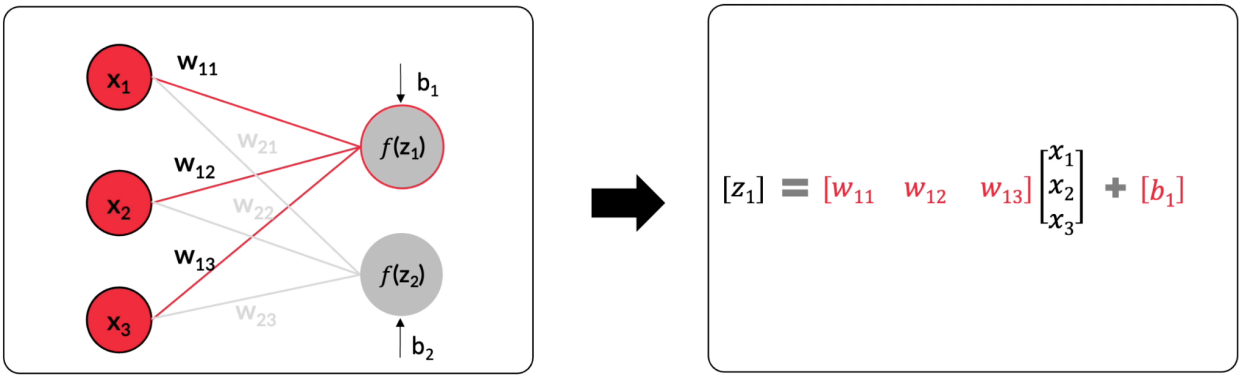
Also, the weight matrix will be of dimension 3x3 and represented as shown below.

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix}$$

The bias vector can be represented as shown below.

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}$$

The matrix representation of obtaining z_1^1 is given below.



$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$$

Here, z_1^1 is obtained by taking a dot product of the input vector and the corresponding weights. The same goes for obtaining the value of z_2^1. Hence, we get:

$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2$$

The two equations can be written as a matrix multiplication.

$$\begin{bmatrix} z_1^1 \\ z_2^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1 \\ w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1 \end{bmatrix}$$

The next step is to apply the activation function to the z^1 vector to obtain the output h^1. The activation function is applied to each element of the vector. Thus, the final output h^1 of layer 1 is as given below.

$$h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \sigma(W^1 . x^i + b^1) = \begin{bmatrix} \sigma(w_{11}^1 x_1 + w_{12}^1 x_2 + w_{13}^1 x_3 + b_1^1) \\ \sigma(w_{21}^1 x_1 + w_{22}^1 x_2 + w_{23}^1 x_3 + b_2^1) \end{bmatrix}$$

Here, $\sigma(x)$ is a vector function, i.e., it is applied element-wise to a vector.

This completes the forward propagation of a single data point through one layer of the network.

To summarise, following are the steps involved in computing the output of the i^{th} neuron in layer l:

- Multiply each row of the weight matrix by the output from the previous layer to obtain the weighted sum of inputs from the previous layer.
- Convert the weighted sum into the cumulative sum by adding the bias vector.
- Apply the activation function $\sigma(x)$ to the cumulative input to obtain the output vector h.

Forward Pass: Demonstration

The problem statement is to predict the prices of houses, given the size of the houses and the number of rooms available.

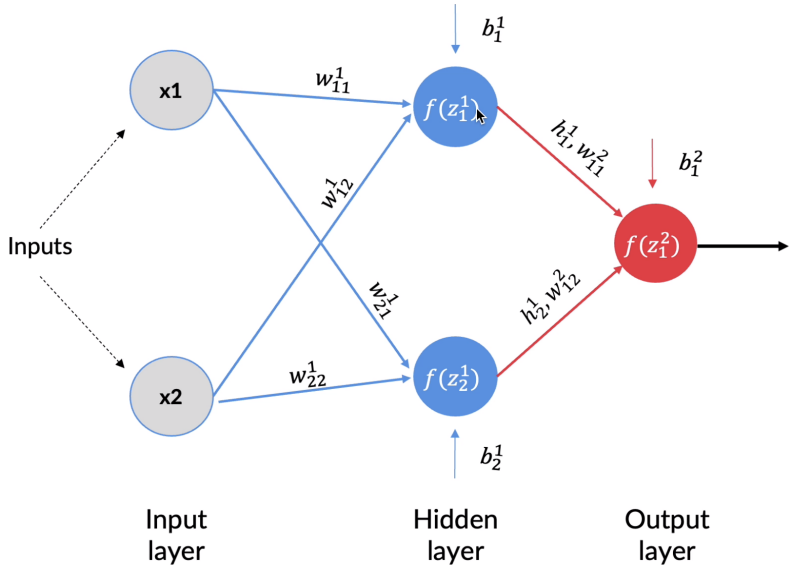
Std. Number of Rooms	Std. House Size (sq. ft.)	Price ($)
3	1,340	31,3000
5	3,650	2,38,4000
3	1,930	34,2000

Std. Number of Rooms	Std. House Size (sq. ft.)	Price ($)
3	2,000	4,20,000
4	1,940	5,50,000
2	880	4,90,000

In this case, we first scale the data for these six observations using the formula: (obs - mean)/standard deviation. So, we get the following table.

Std. Number of Rooms	Std. House Size (sq. ft.)	Price ($)
-0.32	-0.66	-0.54
1.61	1.80	2.03
-0.32	-0.03	-0.51
-0.32	0.05	-0.41
0.65	-0.02	-0.25
-1.29	-1.15	-0.32

We want to build a neural network that will predict the price of a house, given two input attributes: number of rooms and the size of the house. Let's start with the structure of the neural network that we will consider for this case. We have an input layer with two input nodes, x1 and x2; one hidden layer with two nodes; a sigmoid activation function and finally an output layer with a linear activation function (since this is a regression problem) as shown below.



To understand how the data moves forward in the network to enable the neural network to make predictions, we will initialise the weights and biases with random values. The intention is that as this network gets trained, the weights and biases will be updated as per the data such that the predicted output will eventually be the same as, or at least similar to, the actual output.

Let's start by initialising the weights and biases to the following values.

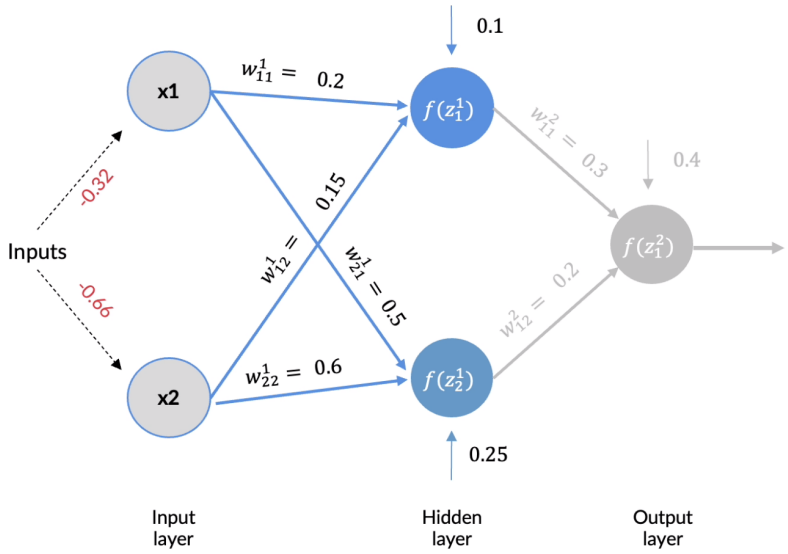
$$W^1, W^2 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix}, \begin{bmatrix} w_{11}^2 \\ w_{12}^2 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.15 \\ 0.5 & 0.6 \end{bmatrix}, \begin{bmatrix} 0.3 \\ 0.2 \end{bmatrix}$$

$$b^1, b^2 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}, [b_1^2] = \begin{bmatrix} 0.1 \\ 0.25 \end{bmatrix}, [0.4]$$

Remember, the superscript denotes the layer to which it belongs, and the subscript denotes the node in that particular layer.

To showcase the step-by-step computation of the output, let's take the first example as the input vector.

$$X^1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -0.32 \\ -0.66 \end{bmatrix}$$



Layer 1: Node 1

Let's compute the output from the first node in layer 1. To do this, we will first compute the cumulative input to this node and apply an activation function to this cumulative input to obtain the output from the node.

Computing the cumulative input for the first node of the hidden layer:

$$z_1^1 = w_{11}^1 x_1 + w_{12}^1 x_2 + b_1^1 = 0.2 * (-0.32) + 0.15 * (-0.66) + 0.1 = -0.063$$

Applying the sigmoid activation function to obtain the output from the first node:

$$h_1^1 = \sigma(-0.063) = \frac{1}{1+e^{-z_1^1}} = \frac{1}{1+e^{-(-0.063)}} = 0.484$$

Layer 1: Node 2

Next, let's compute the output from the second node in layer 1 by following a similar process.

Computing the cumulative input for the second node of the hidden layer:

$$z_2^1 = w_{21}^1 x_1 + w_{22}^1 x_2 + b_2^1 = 0.5 * (-0.32) + 0.6 * (-0.66) + 0.25 = -0.306$$

Applying the sigmoid activation function to get the output from the second node:

$$h_2^1 = \sigma(-0.306) = \frac{1}{1+e^{-z_2^1}} = \frac{1}{1+e^{-(-0.306)}} = 0.424$$

These individual operations can be done together using **matrix multiplication.** We have the input vector x^1, the weight matrix W^1 and the bias vector b^1 with the following values.

$$X^1 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -0.32 \\ -0.66 \end{bmatrix}$$

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 \\ w_{21}^1 & w_{22}^1 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.15 \\ 0.5 & 0.6 \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.25 \end{bmatrix}$$

We know that:

$$h^1 = \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \sigma(W.x_i + b)$$

$$h^1 = \sigma(\begin{bmatrix} w_{11}x_1 + w_{12}x_2 + b_1 \\ w_{21}x_1 + w_{22}x_2 + b_2 \end{bmatrix})$$

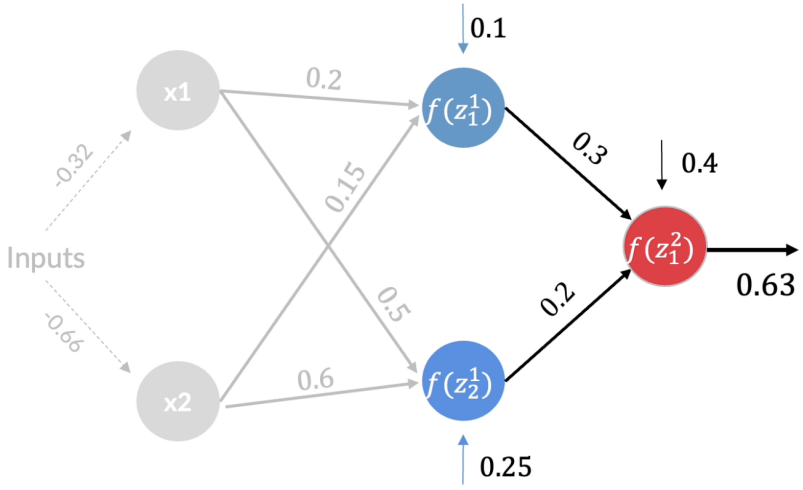
$$h^1 = \sigma(\begin{bmatrix} 0.2 * (-0.32) + 0.15 * (-0.66) + 0.1 \\ 0.5 * (-0.32) + 0.6 * (-0.66) + 0.25 \end{bmatrix})$$

$$= \sigma(\begin{bmatrix} -0.063 \\ -0.306 \end{bmatrix})$$

$$h^1 = \begin{bmatrix} 0.484 \\ 0.424 \end{bmatrix}$$

Now that we have the outputs for the two neurons in the hidden layer, we can calculate the final output.

Layer 2 (Output layer): Node 1



Moving on to the output layer with the linear activation function, we first compute the cumulative input to the neuron.

$$z_1^2 = w_{11}^2 h_1^1 + w_{12}^2 h_2^1 + b_1^2 = 0.3 * 0.484 + 0.2 * 0.424 + 0.4 = 0.63$$

Since this is a regression problem, we have considered the activation function to be the linear activation function, i.e., the input is sent as the output without any modification. Hence, the output is the same as the cumulative input.

$$h_1^2 = z_1^2 = 0.63$$

This value of 0.63 is the prediction that the neural network makes in the first forward pass.

The matrix multiplication method will give us the same output as shown below.

$$h^2 = (W^2 h^1 + b^2) = \left(\begin{bmatrix} w_{11}^2 w_{12}^2 \end{bmatrix} \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}\right) + b^2$$

$$h^2 = (W^2 h^1 + b^2) = \left([0.3 0.2] \begin{bmatrix} 0.484 \\ 0.424 \end{bmatrix}\right) + 0.4$$

$$h^2 = [0.63]$$

Hence, performing the forward pass through the neural network using the input as [-0.32, -0.66] gives us the output as 0.63. The predicted value is different from the

actual value of -0.54, but this is to be expected because we initialised the neural network with random weights and biases. Backpropagation (covered in the next session) allows us to update the weights and biases such that the network makes accurate predictions.

Feedforward Algorithm

Having understood how information flows in the network for a regression problem, let's write the pseudocode for a feedforward pass through the network for a single data point x_i.

Note that h0 has been initialised with xi, the input to the network.

1) We initialise the variable h^0 as the input.

$$h^0 = x_i$$

2) We loop through each of the layers, computing the corresponding output for each layer, i.e., h^l.

For l in $[1, 2, \ldots\ldots, L]$:

$$h^l = \sigma(W^l . h^{l-1} + b^l)$$

3) We compute the prediction p by applying an activation function to the output from the previous layer, i.e., we apply a function to h^L as shown below.

$$p = f(h^L)$$

In both the regression and classification problems, the same algorithm is used until the last step. In the final layer, in the classification problem, p defines the probability vector, which gives the probability of the data point belonging to a particular class among different possible classes or categories. In the regression problem, p represents the predicted numeric output obtained, which we will normally refer to as h^L.

We now use the **softmax output,** which gives us the probability vector p_i of an input belonging to one of the multiple output classes (c).

$$p_i = \begin{bmatrix} p_{i1} \\ p_{i2} \\ \vdots \\ p_{ic} \end{bmatrix}$$

As per our understanding of the softmax function, we know that $p_{ij} = \frac{e^{w_j.h^L}}{\sum_{t=1}^{c} w_t.h^L}$ for $j = [1, 2,..., c]$ and c = Number of classes. Note that calculating $p_{ij} = \frac{e^{w_j.h^L}}{\sum_{t=1}^{c} w_t.h^L}$ is often called **normalising** the vector p_i.

Hence, the complete feedforward algorithm for the **classification problem** becomes:

1) $h^0 = x_i$

2) For l in $[1, 2,....., L]$:

$$h^l = \sigma(W^l.h^{l-1} + b^l)$$

3) $p_i = e^{W^o.h^L}$

4) $p_i = normalise(p_i)$

For the **regression problem**, we can **skip the third and fourth steps**, i.e., computing the probability and normalising the 'predicted output vector' p, because in a regression problem, the output is h^L, *i. e.*, the value we obtain from the single output node, and we usually compare the output obtained from the ANN directly with the ground truth output. We do not perform any further operations on the predicted output to get probabilities in a regression problem.

Note that W^o (the weights of the output layer) can also be written as W^{L+1}.

The classification feedforward algorithm has been extensively used in different sectors of multiple industries such as healthcare, finance and supply chain. Considering the finance industry, one of the applications of this algorithm is categorising customer applications for credit cards as 'good', 'bad' or 'needing further analysis' by credit card companies. For this, credit card companies consider different factors such as annual salary, outstanding debts and age. These can be

the features in the input vector that is fed into a neural network, which then predicts the category to which the customer belongs.

Loss Function

Std. Number of Rooms	Std. House Size (sq. ft.)	Predicted Price	Actual Price
-0.32	-0.66	0.63	-0.54

In this table, you can see that the predicted price is not the same as, or even close to, the actual price. A loss function or cost function will help us quantify such errors.

A **loss function** or **cost function** is a function that maps an **event** or **values** of one or more variables onto a real number intuitively, representing some 'cost' associated with the 'event'.

$$L(y, \hat{y}) = f: (y, \hat{y}) \rightarrow R$$

Neural networks minimise the error in the prediction by optimising the loss function with respect to the parameters in the network. In other words, this optimisation is done by adjusting the weights and biases. You will see how this adjustment is done, in subsequent sessions. For now, we will concentrate on how to compute the loss.

In the case of regression, the most commonly used loss function is **MSE/RSS.**

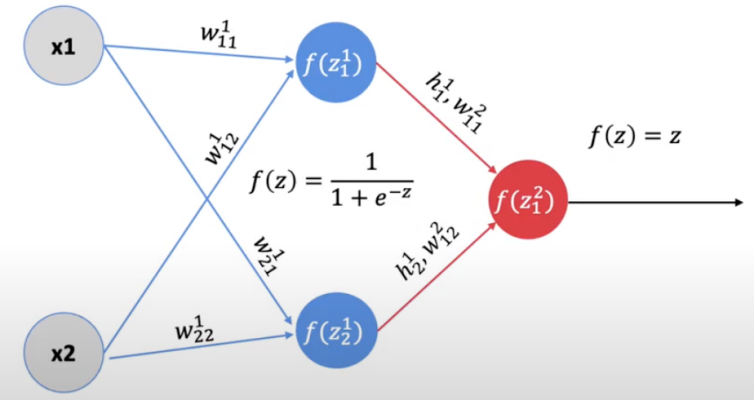
In the case of classification, the most commonly used loss function is **cross entropy / log loss.**

Let's consider the regression problem where we predict the house price, given the number of rooms and the size of the house. Here, we will use the RSS method to calculate the loss.

Std. Number of Rooms	Std. House Size (sq. ft.)	Predicted Price	Actual Price
-0.32	-0.66	0.63	-0.54

In this example, we get a prediction 0.63, but the expected output is -0.54. Let's calculate the loss using RSS.

$$Loss\ (L) = \frac{1}{2}(actual - predicted)^2 = \frac{1}{2}(y - h_1^2)^2 = \frac{1}{2}(-0.54 - 0.63)^2 = 0.68445$$



As given above, MSE is the mean square error of all the samples in the given data. This gives us a method of quantifying how well the neural network is predicting the output.

Learning in Neural Networks

The task of training neural networks is similar to that of other ML models such as linear regression and logistic regression. The desired output (output from the last layer) minus the actual output is the **cost** (or the **loss**). We need to tune the parameters w and b such that **the total cost is minimised.**

$$Loss(L) = RSS = \sum(actual - h^L)^2;\ Loss(L) = f(W, b)$$

The total loss is the sum of losses of all the individual data points. If you have a million data points, they will be fed into the network (in batches), the output will be

calculated using feedforward, and the loss/cost Li (for i^{th} data point) will be calculated.

The total loss, L, is a function of *w*'s and *b*'s. Once the total loss is computed, the weights and biases are updated (in the direction of decreasing loss during backpropagation). In other words, L is **minimised with respect to the *w*'s and *b*'s.**

Minimising the average loss implies that the total loss is getting minimised.

This can be done using any optimisation routine such as **gradient descent.**

In gradient descent, the parameter being optimised is iterated in the direction of reducing cost according to the following rule.

$$W_{new} = W_{old} - \alpha \frac{\partial L}{\partial W}$$

Note that weights and biases are often collectively represented by one matrix called W. Going forward, W will, by default, refer to the matrix of all weights and biases.

Comprehension: Gradient Descent

Gradient descent is an optimisation algorithm used to find the minima of a function. The basic idea is to use the gradient of the function to find **the direction of the steepest descent**, i.e., the direction in which the value of the function decreases most rapidly, and move towards the minima iteratively.

The algorithm starts with an initial arbitrary guess of w, computes the gradient at that point and updates w according to the rule iteratively.

$$w_{new} = w_{old} - \alpha \frac{\partial L}{\partial w}$$

The gradient has the following **two critical pieces of information:**

- The **sign of the gradient** (positive here) is the 'direction' in which the function value increases, and thus, a negative sign denotes the **direction of decrease.**
- The **value of the gradient** (10 here) represents how steeply the function value increases or decreases at that point.

Gradient descent can be easily extended to multivariate functions, i.e., functions of multiple variables.

upGrad

To summarise, you learnt how information flows from the input layer to the output layer in artificial neural networks (feedforward).

You studied feedforward for a regression problem based on the housing price prediction problem statement. You also learnt how to specify the dimensions and representations of the weight matrices, biases, inputs and outputs of the various layers.

You gained an understanding of how **feedforward** can be done in a **vectorised form** and how it can be made efficient using **parallelisation.**

Further, you learnt that in order to train a neural network, you need to optimise the weights and biases of the network using optimisation techniques, such as gradient descent.