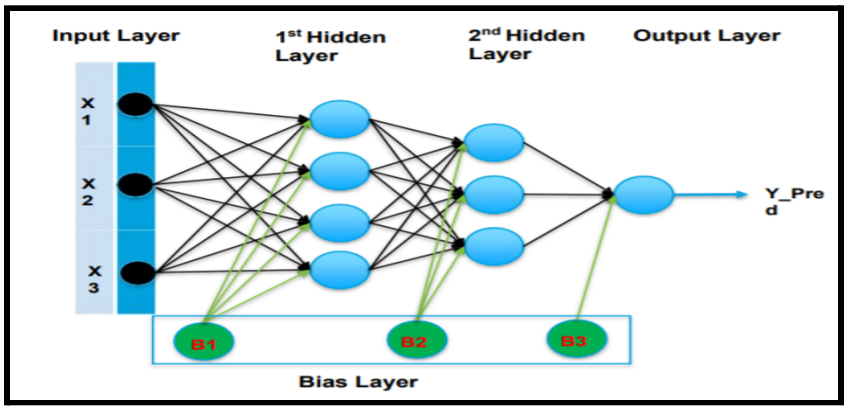
Regression using Neural Networks

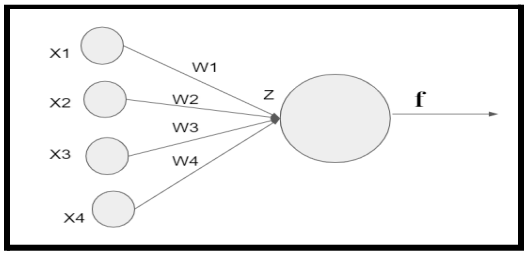
Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Neural networks are comprised of layers with neurons, containing an input layer, one or more hidden layers, and an output layer. Each neuron connects to another and has an associated weight and threshold. If the output of any individual neuron is above the specified threshold value, that neuron is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.



- The input layer is passive, does no processing, only holds the input data to supply it to the **first hidden** layer. It is a vector that describes the features of the input.
- Anything that is not Input and Output layers are called **hidden layers**. Each neuron in the first hidden layer takes all input attributes, **multiplies** with the corresponding **weights**, adds **bias**, and the output is transformed using a **nonlinear function**.
- The weights for a given hidden neuron are randomly initialized and all the neurons in the hidden layer will have their **weights**.
- The output of each neuron is fed to output layer nodes or another set of hidden nodes in another hidden layer.
- The output value of each hidden neuron is sent to each output node in the output layer. Output for a regression problem will be a **real number**.

Let's see what operations are happening in a single neuron (unit)



- A neuron in a neural network works in three steps:
    1. First, it multiplies the input signals with corresponding weights.
    2. Second, the weighted sum of the inputs.
    3. Applies activation function on the weighted sum.
- The input to a neuron is a **summation** of **inputs** and respective **weights**.

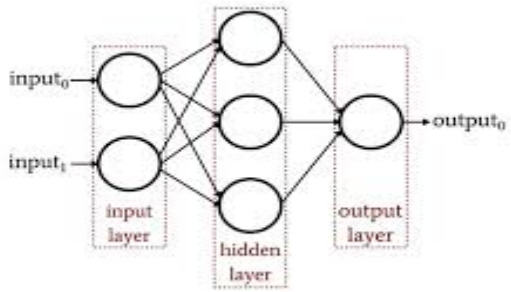
$$Z = X_1 * W_1 + X_2 * W_2 + X_3 * W_3 + X_4 * W4$$

- If the weight is **zero**, the respective input will be **ignored** by the neuron.
- For this Z we are going to add an activation function which is to add non-linearity.
- So, a neuron in a neural network works as a linear classifier of the inputs, now there are many linear classifiers in the hidden layers which have different weights, each neuron **detects** a different **pattern** but essentially is a linear classifier.

The idea over here is to use parameterized nonlinear transformations of linear combinations of the raw regressors as constructed regressors (called neurons) and produce the predicted value as a linear function of these regressors.

Neural networks also rely on many constructed regressors to approximate g(Z). The method and the name "neural networks" were loosely inspired by the mode of operation of the human brain, and developed by scientists working on Artificial Intelligence.

Single Layer Neural Networks:



Above is the single-layered neural network which has only one hidden layer, So in the hidden layer, we have 3 neurons and we have 6 weights .i.e for each neuron we have two inputs and two weights. The output of each neuron will be a linear combination of weight and inputs and an activation function,

For neuron 1, it will be $O1 = \sigma(X_1 * W_{11} + X_2 * W_{12})$

For neuron 2, it will be $O2 = \sigma(X_1 * W_{21} + X_2 * W_{22})$

For neuron 3, it will be $O3 = \sigma(X_1 * W_{31} + X_2 * W_{32})$

In the output layer, we have one neuron which takes input as output from the previous layer. So the output will be,

Final output = $\beta_1 * O1 + \beta_2 * O2 + \beta_3 * O3$

Formally,

The estimated prediction rule will be,

$$\hat{g}(Z) = \sum_{m=1}^{M} \hat{\beta}_m X_m(\hat{\alpha}_m)$$

M= number of neurons

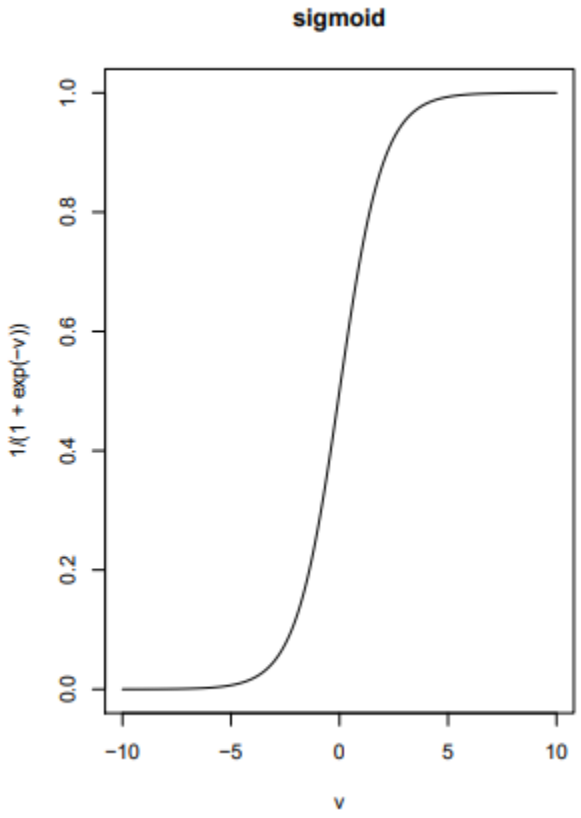
Z= Input values

The below images is nothing but the multiplication of weights with inputs and applying an activation function. σ is the activation function. Here α'_m are the weights for the hidden layer.

$$X_m(\alpha_m) = \sigma(\alpha'_m Z), \quad m = 1, \ldots, M,$$

Once we got the output of hidden neurons, for calculating output we need another set of weights which are denoted by $\hat{\beta}$.

The activation functions can be sigmoid, relu, tanh etc. For example the sigmoid function,



$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

The weights α'_m and β'_m, for each m from 1 to capital M, are obtained as the solution to the penalized nonlinear least-squares problem,

$$\min_{\{\alpha_m\},\{\beta_m\}} \sum_i \left(Y_i - \sum_{m=1}^{M} \beta'_m X_{im}(\alpha_m) \right)^2 + \lambda \left(\sum_m \sum_j |\alpha_{mj}| + \sum_m |\beta_m| \right)$$

Here we are minimizing the sum of squared prediction errors in the sample plus a penalty term given by the sum of the absolute values of components of α_m and β_m

multiplied by the penalty level λ. In this formula, we use the Lasso type penalty, but we can also use the Ridge and another type of penalty.

The estimates or weights are computed using sophisticated gradient descent algorithms, where sophistication is needed because the nonlinear least-squares optimization problem is generally not a convex problem, making the computation a difficult task. The procedure of fitting the neural network model has many tuning parameters, and in practice, we can choose them by cross-validation. The most important choices concern the number of neurons and the number of neuron layers. Having more neurons gives us more flexibility, just like having more constructed regressors gives us more flexibility with high-dimensional linear models. To prevent overfitting, we can rely on penalization as in the case of linear regression.

We encourage you to play with neural networks Tensorflow playground.

To summarize,
Prediction methods based on neural networks with several layers of neurons are called the “deep learning” methods. Neural networks represent a powerful and all-purpose method for prediction and regression analysis. Using many neurons and multiple layers gives rise to networks that are very flexible and that can approximate the best prediction rules quite well.