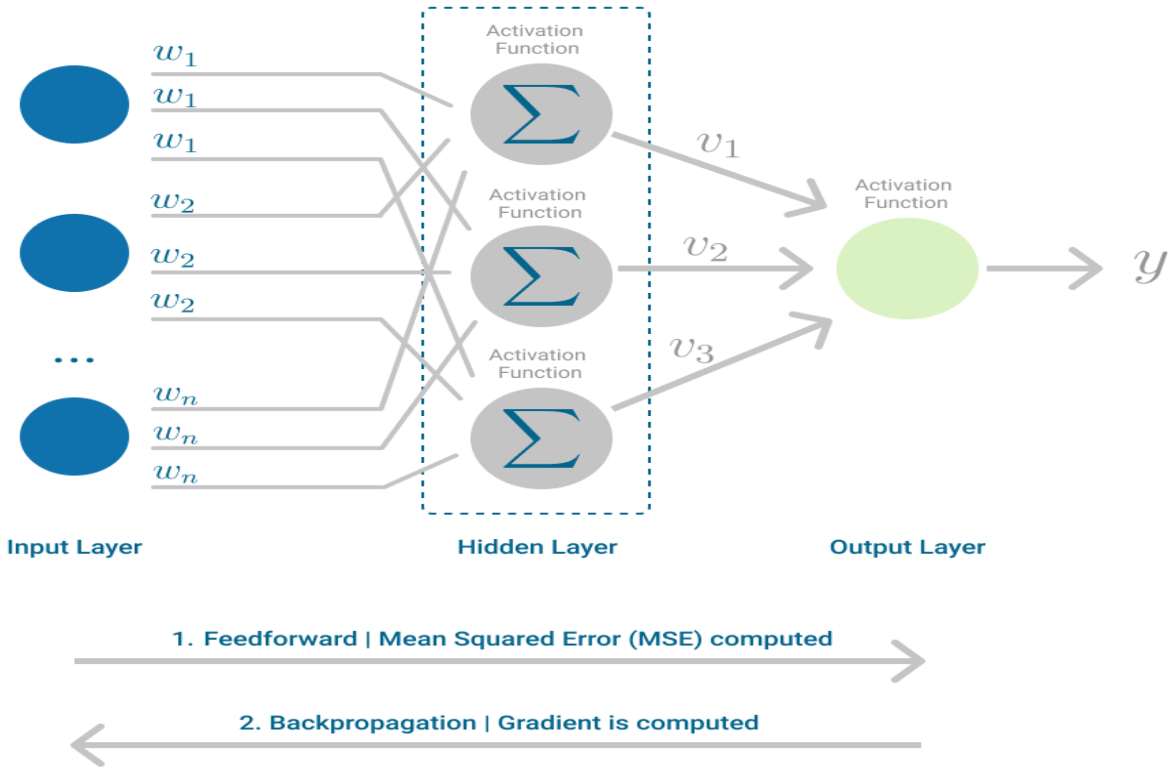
Multilayer Perceptron



Feed-Forward

Following are the steps performed during the feed-forward phase:

1. The values received in the input layer are multiplied with the weights. A bias is added to the summation of the inputs and weights in order to avoid null values.
2. Each neuron in the first hidden layer receives different values from the input layer depending upon the weights and bias. Neurons have an activation function that operates upon the value received from the input layer. The activation function can be of many types, like a step function, sigmoid function, relu function, or tanh function. As a rule of thumb, relu function is used in the hidden layer neurons and sigmoid function is used for the output layer neuron.

3. The outputs from the first hidden layer neurons are multiplied with the weights of the second hidden layer; the results are summed together and passed to the neurons of the proceeding layers. This process continues until the outer layer is reached. The values calculated at the outer layer are the actual outputs of the algorithm.

The feed-forward phase consists of these three steps. However, the predicted output is not necessarily correct right away; it can be wrong, and we need to correct it. The purpose of a learning algorithm is to make predictions that are **as accurate as** possible. To improve these predicted results, a neural network will then go through a back propagation phase. During back propagation, the weights of different neurons are updated in a way that the difference between the desired and predicted output is as small as possible.

Back Propagation

Back propagation phase consists of the following steps:

1. The error is calculated by quantifying the difference between the predicted output and the desired output. This difference is called "loss" and the function used to calculate the difference is called the "loss function". Loss functions can be of different types e.g. mean squared error or cross entropy functions. Remember, neural networks are supervised learning algorithms that need the desired outputs for a given set of inputs, which is what allows it to learn from the data.
2. Once the error is calculated, the next step is to minimize that error. To do so, partial derivative of the error function is calculated with respect to all the weights and biases. This is called gradient decent. The derivatives can be used to find the slope of the error function. If the slop is positive, the value of the weights can be reduced or if the slop is negative the value of weight can be increased. This reduces the overall error. The function that is used to reduce this error is called the optimization function.

This one cycle of feed-forward and back propagation is called one "epoch". This process continues until a reasonable accuracy is achieved. There is no standard for reasonable accuracy, ideally you'd strive for 100% accuracy, but this is extremely difficult to achieve for any non-trivial dataset. In many cases 90%+ accuracy is considered acceptable, but it really depends on your use-case.

The procedure for adjusting the weights is:

1. Present inputs for the first pattern to the input layer
2. Sum the weighted inputs to the next layer and calculate their activations
3. Present activations to the next layer, repeating (2) until the activations of the output layer are known
4. Compare output activations to the target values for the pattern and calculate deltas for the output layer
5. Propagate error backwards by using the output layer deltas to calculate the deltas for the previous layer
6. Use these deltas to calculate those of the previous layer, repeating until the first layer is reached
7. Calculate the weight changes for all weights and biases (treat biases as weights from a unit having an activation of 1)
8. If training by pattern, update all the weights and biases, else repeat the cycle for all patterns, summing the changes and applying at the end of the epoch
9. Repeat the entire procedure until the total sum of squared errors is less than a specified criterion

Importing Libraries

```
import pandas as pd

# To import the dataset and load it into our pandas dataframe,
execute the following code:
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"

# Assign colum names to the dataset
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-
width', 'Class']

# Read dataset to pandas dataframe
irisdata= pd.read_csv(url, names=names)


#from sklearn.datasets import load_iris
#dataset = load_iris()
```

To see what the dataset actually looks like, execute the following command:

```
irisdata.head()
```

The next step is to split our dataset into its attributes and labels. To do so, use the following code:

```
# Assign data from first four columns to X variable
 X = irisdata.iloc[:, 0:4]

# Assign data from first fifth columns to y variable
y = irisdata.iloc[:, -1]

#To see what y looks like, execute the following code:
y.head()
```

Let's convert these categorical values to numerical values. To do so we will use Scikit-Learn's LabelEncoder class.

Execute the following script:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

y = le.fit_transform(y)
```

To create training and test splits, execute the following script:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20)
```

The following script performs feature scaling:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Training and Predictions:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train)

predictions = mlp.predict(X_test)
```

Evaluating the Algorithm:

```
from sklearn.metrics import accuracy_score

predictions_train = mlp.predict(X_train)
print(accuracy_score(predictions_train,y_train))
predictions_test = mlp.predict(X_test)
print(accuracy_score(predictions_test, y_test))
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,predictions))
print(classification_report(y_test,predictions))
```