Neural Networks (NN) / Multilayer Perceptrons (MLP)

Recall that machine learning algorithms are nothing but function approximation. This is true for Neural Networks too.

Let's start with a very simple neural network algorithm called the Perceptron

The **Perceptron** (shown below) is a very simple **linear binary classifier**. It basically maps and input vector x to a binary output f(x).

Given a weight vector w, the Perceptron's classfication rule is: f(x) = 1 if w · x + b > 0 or f(x) = 0 otherwise.

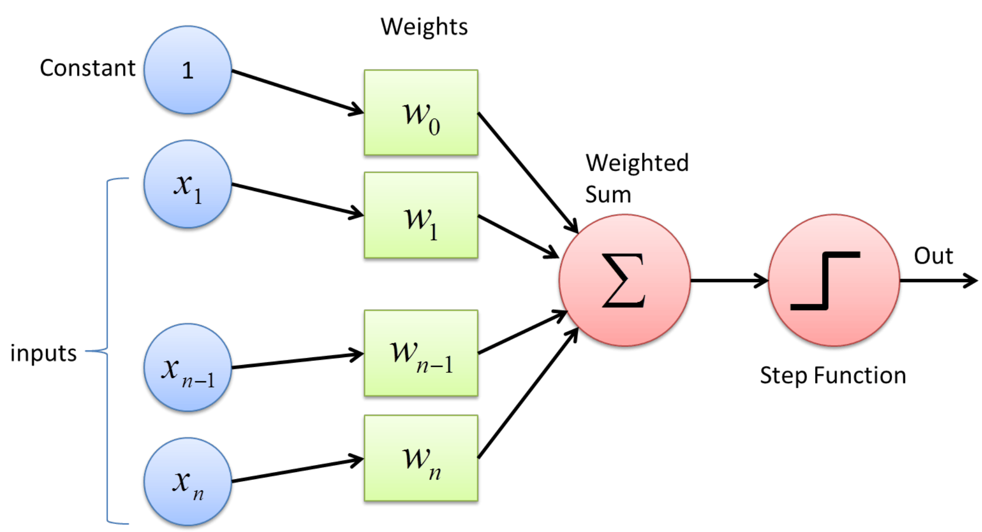
The training step is used to learn the optimal values of the weights w and b.

Here, b is a bias value which is responsible for shifting the Perceptron's hyperplane away from the origin.

Question: Does this remind you of something you have recently seen?

In [70]:
```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "https://cdn-images-1.medium.com/max/1600/1*n6sJ4yZQzwKL9wnF5
wnVNg.png", width=500)
```

Out[70]:



The step function at the output is called the *activation function*.

The most simple examples for Perceptron are the basic logic operations, such as: AND, OR and XOR. The truth tables for these logic functions is shown below.

```
In [71]: Image(url= "http://www.talkingelectronics.com/pay/PIC/TruthTable-2.gif",
         width=300)
```

Out[71]:

AND

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

XOR

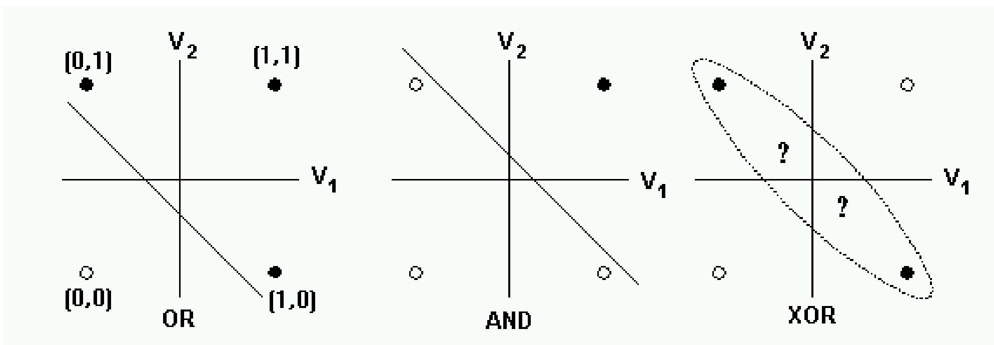
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

exclusive-OR

A graphical view of these three logic functions is shown below. We see that the AND and OR logic functions are linearly separable but the XOR is not. What do you think this means?

```
In [72]: Image(url="http://ecee.colorado.edu/~ecen4831/lectures/xor2.gif", width=
         500)
```

Out[72]:



Let's run the Perceptron algorithm to learn these three logical functions.

```
In [73]: import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         from matplotlib.colors import ListedColormap
         from sklearn.preprocessing import LabelEncoder, StandardScaler
         from sklearn.linear_model import Perceptron
         from sklearn.neural_network import MLPClassifier
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score

         # Setting random seed.
         seed = 10
```

```
In [74]: # Setting the input samples
         X = np.array([[0, 0],
         [0, 1],
         [1, 0],
         [1, 1]],
         dtype=np.double)
```

In [75]:

```
# Setting the expected outputs for AND
y_AND = np.array([0, 0, 0, 1])

# Creating and training a Perceptron.
p = Perceptron(random_state=seed, eta0=0.1, max_iter=1000)
p.fit(X, y_AND)
y_pred = p.predict(X)
print('Perceptron for AND')
print('Expected Output:',y_AND)
print('Predicted Output:',y_pred)
print('Accuracy: %.2f' % accuracy_score(y_AND, y_pred))

# Obtaining confidence scores.
np.set_printoptions(precision=2)
pred_scores = p.decision_function(X)
print("Confidence scores: {}".format(pred_scores))
```

```
Perceptron for AND
Expected Output: [0 0 0 1]
Predicted Output: [0 0 0 1]
Accuracy: 1.00
Confidence scores: [-2.00e-01 -1.00e-01 -2.78e-17  1.00e-01]
```

In [76]:

```
# Setting the expected outputs for OR
y_OR = np.array([0, 1, 1, 1])
p.fit(X, y_OR)

# Creating and training a Perceptron.
p = Perceptron(random_state=seed, eta0=0.1, max_iter=1000)
p.fit(X, y_OR)
y_pred = p.predict(X)
print('Perceptron for OR')
print('Expected Output:',y_OR)
print('Predicted Output:',y_pred)
print('Accuracy: %.2f' % accuracy_score(y_OR, y_pred))

# Obtaining confidence scores.
np.set_printoptions(precision=2)
pred_scores = p.decision_function(X)
print("Confidence scores: {}".format(pred_scores))
```

```
Perceptron for OR
Expected Output: [0 1 1 1]
Predicted Output: [0 1 1 1]
Accuracy: 1.00
Confidence scores: [-0.1  0.1  0.1  0.3]
```

In [77]:

```
# Setting the expected outputs for XOR
y_XOR = np.array([0, 1, 1, 0])
p.fit(X, y_XOR)

# Creating and training a Perceptron.
p = Perceptron(random_state=seed, eta0=0.1, max_iter=1000)
p.fit(X, y_XOR)
y_pred = p.predict(X)
print('Perceptron for XOR')
print('Expected Output:',y_XOR)
print('Predicted Output:',y_pred)
print('Accuracy: %.2f' % accuracy_score(y_XOR, y_pred))

# Obtaining confidence scores.
np.set_printoptions(precision=2)
pred_scores = p.decision_function(X)
print("Confidence scores: {}".format(pred_scores))
```

```
Perceptron for XOR
Expected Output: [0 1 1 0]
Predicted Output: [0 0 0 0]
Accuracy: 0.50
Confidence scores: [0. 0. 0. 0.]
```

Clearly, XOR is not a linearly separable problem. In other words, it is not possible to separate the two classes with a single hyperplane.

This kind of problem motivates us to use Multilayer Perceptrons (MLPs), which we explore next.

A MLP is a neural network which is composed by at least three different layers: an input layer, a hidden layer and an output layer.

Except for the input layer, the remaining ones are composed by Perceptrons (we call them nodes) with nonlinear activation functions (e.g., sigmoid or tanh).

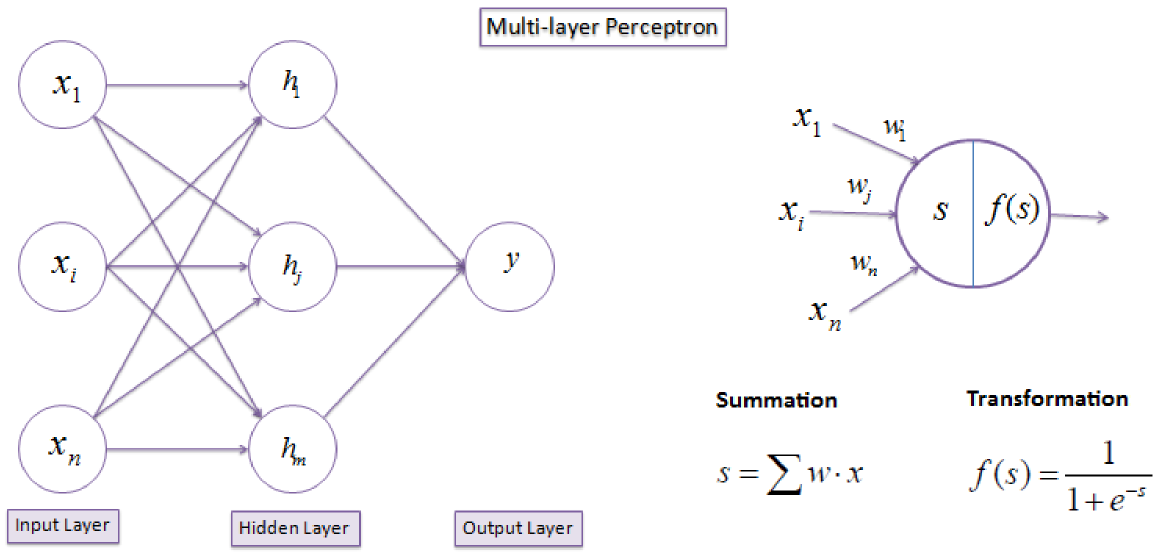
MLPs are usually trained using the backpropagation algorithm and are able to deal with not linearly separable problems. The training step is used to learn the weights on the edges between the nodes.

Below we use the MLP for the XOR problem.

In [78]:
```
# Image(url= "https://upload.wikimedia.org/wikipedia/commons/thumb/4/46/
Colored_neural_network.svg/296px-Colored_neural_network.svg.png", width=
300)

Image(url= "http://www.saedsayad.com/images/Perceptron_bkp_1.png", width
=700)
```

Out[78]:



In [79]:
```
# Creating a MLPClassifier.
# hidden_layer_sizes receive a tuple where each position i indicates the
number of neurons
# in the ith hidden layer
# activation specifies the activation function (other options are: 'iden
tity', 'logistic' and 'relu')
# max_iter indicates the maximum number of training iterations
# There are other parameters which can also be changed.
# See http://scikit-learn.org/stable/modules/generated/sklearn.neural_ne
twork.MLPClassifier.html

mlp = MLPClassifier(hidden_layer_sizes=(10,10,),
activation='tanh',
max_iter=10000,
random_state=seed)

# Training and plotting the decision boundary.
mlp.fit(X, y_XOR)

y_pred = mlp.predict(X)
print('MLP for XOR')
print('Expected Output:',y_XOR)
print('Predicted Output:',y_pred)
print('Accuracy: %.2f' % accuracy_score(y_XOR, y_pred))

# Obtaining probabiliteis
pred = mlp.predict_proba(X)
print("MLP's XOR probabilities:\n[class0, class1]\n{}".format(pred))
```

```
MLP for XOR
Expected Output: [0 1 1 0]
Predicted Output: [0 1 1 0]
Accuracy: 1.00
MLP's XOR probabilities:
[class0, class1]
[[0.99 0.01]
 [0.05 0.95]
 [0.05 0.95]
 [0.97 0.03]]
```

MLP for Wisconsin Diagnostic Breast Cancer Data

```
In [80]: from sklearn.datasets import load_breast_cancer
         from sklearn.preprocessing import StandardScaler

         # Loading Breast Cancer dataset.
         wdbc = load_breast_cancer()
         df = pd.DataFrame(wdbc.data, columns=wdbc.feature_names)
         wdbc.data[:1]
```

```
Out[80]: array([[1.80e+01, 1.04e+01, 1.23e+02, 1.00e+03, 1.18e-01, 2.78e-01,
                 3.00e-01, 1.47e-01, 2.42e-01, 7.87e-02, 1.09e+00, 9.05e-01,
                 8.59e+00, 1.53e+02, 6.40e-03, 4.90e-02, 5.37e-02, 1.59e-02,
                 3.00e-02, 6.19e-03, 2.54e+01, 1.73e+01, 1.85e+02, 2.02e+03,
                 1.62e-01, 6.66e-01, 7.12e-01, 2.65e-01, 4.60e-01, 1.19e-01]])
```

```
In [81]: wdbc.target[:40]
```

```
Out[81]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
         1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0])
```

```
In [82]: list(wdbc.target_names)
```

```
Out[82]: ['malignant', 'benign']
```

```
In [87]: X_train, X_test, y_train, y_test = train_test_split\
         (wdbc.data, wdbc.target, test_size=0.33, stratify=wdbc.target, \
          random_state=np.random.randint(1,10))
         mlp = MLPClassifier(hidden_layer_sizes=(10,),
         activation='tanh',
         max_iter=10000,
         random_state=np.random.randint(1,10))
         mlp.fit(X_train, y_train)
         y_pred = mlp.predict(X_test)
```

```
In [88]: from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         accuracy = accuracy_score(y_test, y_pred)
         print("MLP's accuracy score: {}".format(accuracy))
```

```
MLP's accuracy score: 0.6276595744680851
```

We can observe that its accuracy score is rather low.

Unfortunately, MLPs are very sensitive to different feature scales. So, it is normally necessary to normalize or rescale the input data. With data normalization, we see that the accuracy improves considerably.

In [89]:
```
# Creating a StandardScaler. This object normalizes features to zero mea
n and unit variance.
scaler = StandardScaler()
scaler.fit(X_train)
# Normalizing train and test data.
X_train_scaled, X_test_scaled = scaler.transform(X_train), scaler.transf
orm(X_test)
# Training MLP with normalized data.
mlp.fit(X_train_scaled, y_train)
# Testing MLP with normalized data.
y_pred = mlp.predict(X_test_scaled)
```

In [90]:
```
from sklearn.metrics import classification_report
accuracy = accuracy_score(y_test, y_pred)
print("MLP's accuracy score: {}".format(accuracy))
print(classification_report(y_test, y_pred, target_names=wdbc.target_nam
es))
confusion_matrix(y_test, y_pred)
```

```
MLP's accuracy score: 0.9840425531914894
             precision    recall  f1-score   support

  malignant       0.97      0.99      0.98        70
     benign       0.99      0.98      0.99       118

avg / total       0.98      0.98      0.98       188
```

Out[90]:
```
array([[ 69,   1],
       [  2, 116]])
```