In [2]:

```
# A simple Python example with a single neuron. It has two
# inputs and one output and executes the logic AND of the inputs.

from numpy import exp, array, random, dot
#Set up parameters ===============================================
inputs = array([[0, 0], [1, 1], [1, 0], [0, 1]])
outputs = array([[0, 1, 1, 1]]).T
random.seed(1)
weights = 2 * random.random((2, 1)) - 1
#Define a Single Neuron function =================================
def neuron(inputs, weights):
    output = 1 / (1 + exp(-(dot(inputs, weights))))
    return output
#Train the Neuron ================================================
for iteration in range(50000):
    output = 1 / (1 + exp(-(dot(inputs, weights))))
    weights += dot(inputs.T, (outputs - output) * output * (1 - output))
#Test the Neuron =================================================
x = array([1, 0])
print (neuron (x,weights))
```

```
[0.9968177]
```

In [3]:

```
# A Python multiple layer perceptron using the Scikit-Learn
# library. It has two inputs and one output and performs the logical AND of the inputs.

from sklearn.neural_network import MLPClassifier
X = [[0., 0.], [1., 1.], [0., 1.], [1., 0.]]
y = [0, 1, 1, 1]
clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                    hidden_layer_sizes=(5, 2), random_state=1)
clf.fit(X, y)
print(clf.predict([[2., 2.], [-1., -2.]]))
print([coef.shape for coef in clf.coefs_])
```

```
[1 0]
[(2, 5), (5, 2), (2, 1)]
```

In [4]:

```
# A simple Keras multiple layer perceptron for the breast
# cancer classification. It has an input layer, a hidden layer (dense layer) of 10 neurons,
# The dense layer is a neural network layer that each
# neuron in the dense layer receives input from all neurons of its previous layer.
# Multiple Layer Perceptron Classification

from keras.models import Sequential
from keras.layers import Activation, Dense
from keras import optimizers
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
whole_data = load_breast_cancer()
X_data = whole_data.data
y_data = whole_data.target
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, random
features = X_train.shape[1]
#MPL 3-layer Model ==============================================
model = Sequential()
model.add(Dense(10, activation='relu', input_shape=[X_train.shape[1]]))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))
print(model.summary())
print(model.get_config())
model.compile(optimizer = 'adam', loss = 'mean_squared_error',
metrics = ['mse'])
model.fit(X_train, y_train, batch_size = 50, validation_split=0.2,
epochs = 100, verbose = 1)
results = model.evaluate(X_test, y_test)
print('loss: ', results[0])
print('accuracy: ', results[1])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 10)                310       
                                                                 
 dense_1 (Dense)             (None, 10)                110       
                                                                 
 dense_2 (Dense)             (None, 1)                 11        
                                                                 
=================================================================
Total params: 431
Trainable params: 431
Non-trainable params: 0
_________________________________________________________________
None
{'name': 'sequential', 'layers': [{'class_name': 'InputLayer', 'config':
{'batch_input_shape': (None, 30), 'dtype': 'float32', 'sparse': False, 'ra
gged': False, 'name': 'dense_input'}}, {'class_name': 'Dense', 'config':
```