Implement MLP. Use the sigmoidal activation function used in the algorithm in your text book. Fix the number of hidden neurons based on experimentation. With the same number of hidden neurons experiment using three different activation functions.

DATASET : Pima Indians Diabetes

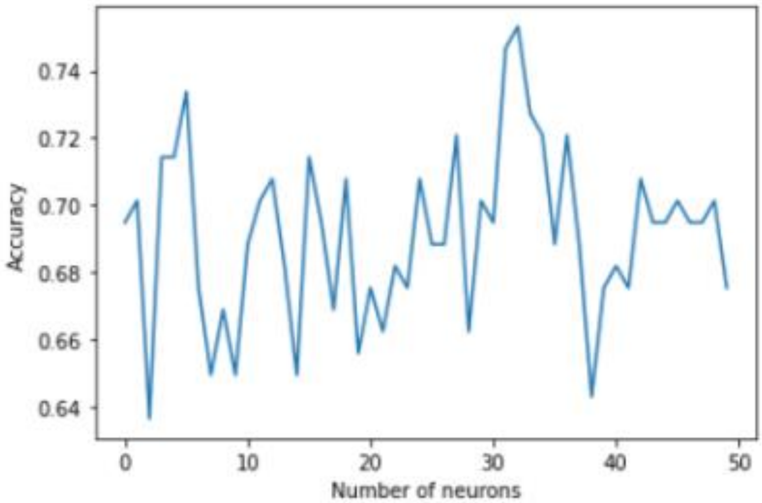
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import pandas as pd
df = pd.read_csv('diabetes.csv')
data = df.iloc[:,0:-1]
target = df.iloc[:,-1]
from sklearn.model_selection import train_test_split
datasets = train_test_split(data, target,
 test_size=0.2)
X_train, X_test, y_train, y_test = datasets
df
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [2]:
```
acc = np.zeros(50)
for i in range(50):
 mlp = MLPClassifier(hidden_layer_sizes=i+1, max_iter=5000, activation = 'logistic')
 mlp.fit(X_train, y_train)
 predictions_test = mlp.predict(X_test)
 acc[i] = accuracy_score(predictions_test, y_test)
 if i == np.argmax(acc):
 max_prediction = predictions_test
plt.xlabel("Number of neurons")
plt.ylabel("Accuracy")
plt.plot(acc)
```

```
[<matplotlib.lines.Line2D at 0x1a9dc3c7f10>]
```



In [3]:
```
n = np.argmax(acc)
print("Number of neurons for maximum accuracy =", n)
print("Accuracy = ", acc[n])
```

```
Number of neurons for maximum accuracy = 32
Accuracy =  0.7532467532467533
```

Performance metrics for sigmoid activation function:

In [4]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print("Performance metrics for sigmoid activation function: ")
y_true = y_test
y_pred = max_prediction
print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))
# classification report for precision, recall f1-score and accuracy
matrix = classification_report(y_true,y_pred)
print('\nClassification report : \n',matrix)
```

```
Performance metrics for sigmoid activation function:

Confusion Matrix:
 [[103   4]
 [ 34  13]]

Classification report :
               precision    recall  f1-score   support

           0       0.75      0.96      0.84       107
           1       0.76      0.28      0.41        47

    accuracy                           0.75       154
   macro avg       0.76      0.62      0.63       154
weighted avg       0.76      0.75      0.71       154
```

Rectified Linear Activation Function:

```
In [5]: print("Rectified Linear Activation Function:\n")
        mlp = MLPClassifier(hidden_layer_sizes=n, max_iter=5000, activation = 'relu')
        mlp.fit(X_train, y_train)
        predictions_test = mlp.predict(X_test)
        print('Number of hidden neurons = ', n)
        print('\nAccuracy = ', accuracy_score(predictions_test, y_test))
        y_true = y_test
        y_pred = predictions_test
        print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))
        # classification report for precision, recall f1-score and accuracy
        matrix = classification_report(y_true,y_pred)
        print('\nClassification report : \n',matrix)
```

```
Rectified Linear Activation Function:

Number of hidden neurons =  32

Accuracy  =  0.6688311688311688

Confusion Matrix:
 [[95 12]
 [39  8]]

Classification report :
               precision    recall  f1-score   support

           0       0.71      0.89      0.79       107
           1       0.40      0.17      0.24        47

    accuracy                           0.67       154
   macro avg       0.55      0.53      0.51       154
weighted avg       0.61      0.67      0.62       154
```

No Op Activation Function:

```
In [6]: print("No Op Activation Function:\n")
        mlp = MLPClassifier(hidden_layer_sizes=n, max_iter=5000, activation = 'identity')
        mlp.fit(X_train, y_train)
        predictions_test = mlp.predict(X_test)
        print('Number of hidden neurons = ', n)
        print('\nAccuracy = ', accuracy_score(predictions_test, y_test))
        y_true = y_test
        y_pred = predictions_test
        print('\nConfusion Matrix: \n', confusion_matrix(y_true, y_pred))
        # classification report for precision, recall f1-score and accuracy
        matrix = classification_report(y_true,y_pred)
        print('\nClassification report : \n',matrix)
```

```
No Op Activation Function:

Number of hidden neurons =  32

Accuracy =  0.6948051948051948

Confusion Matrix:
 [[98  9]
 [38  9]]

Classification report :
               precision    recall  f1-score   support

           0       0.72      0.92      0.81       107
           1       0.50      0.19      0.28        47

    accuracy                           0.69       154
   macro avg       0.61      0.55      0.54       154
weighted avg       0.65      0.69      0.64       154
```

Choose a dataset suitable for regression and apply regression using MLP

Dataset : Real estate

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import accuracy_score
import pandas as pd
df = pd.read_csv('Real_estate.csv')
data = df.iloc[:,0:-1]
target = df.iloc[:,-1]
from sklearn.model_selection import train_test_split
datasets = train_test_split(data, target,
 test_size=0.2)
X_train, X_test, y_train, y_test = datasets
df
```

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1
...	...	...	...	...	...	...	...	...
409	410	2013.000	13.7	4082.01500	0	24.94155	121.50381	15.4
410	411	2012.667	5.6	90.45606	9	24.97433	121.54310	50.0
411	412	2013.250	18.8	390.96960	7	24.97923	121.53986	40.6
412	413	2013.000	8.1	104.81010	5	24.96674	121.54067	52.5
413	414	2013.500	6.5	90.45606	9	24.97433	121.54310	63.9

414 rows × 8 columns

```
In [2]: from sklearn.metrics import mean_absolute_error
        from sklearn.metrics import mean_squared_error
        from math import sqrt
        regr = MLPRegressor(hidden_layer_sizes=(64,64,64), max_iter=2000, activation = 'logistic')
        regr.fit(X_train, y_train)
        y_pred = regr.predict(X_test)
        true_val = y_test.reset_index().iloc[:,-1]
        errors = [true_val[i] - y_pred[i] for i in range(len(true_val))]
        bias = sum(errors) * 1.0/len(true_val)
        print('Bias = ', bias)
        print('\nMean Absolute Error =',mean_absolute_error(true_val, y_pred))
        print('\nRoot Mean Squared Error =',sqrt(mean_absolute_error(true_val, y_pred)))
        print('\nR-Squared =', regr.score(X_test, y_test))
```

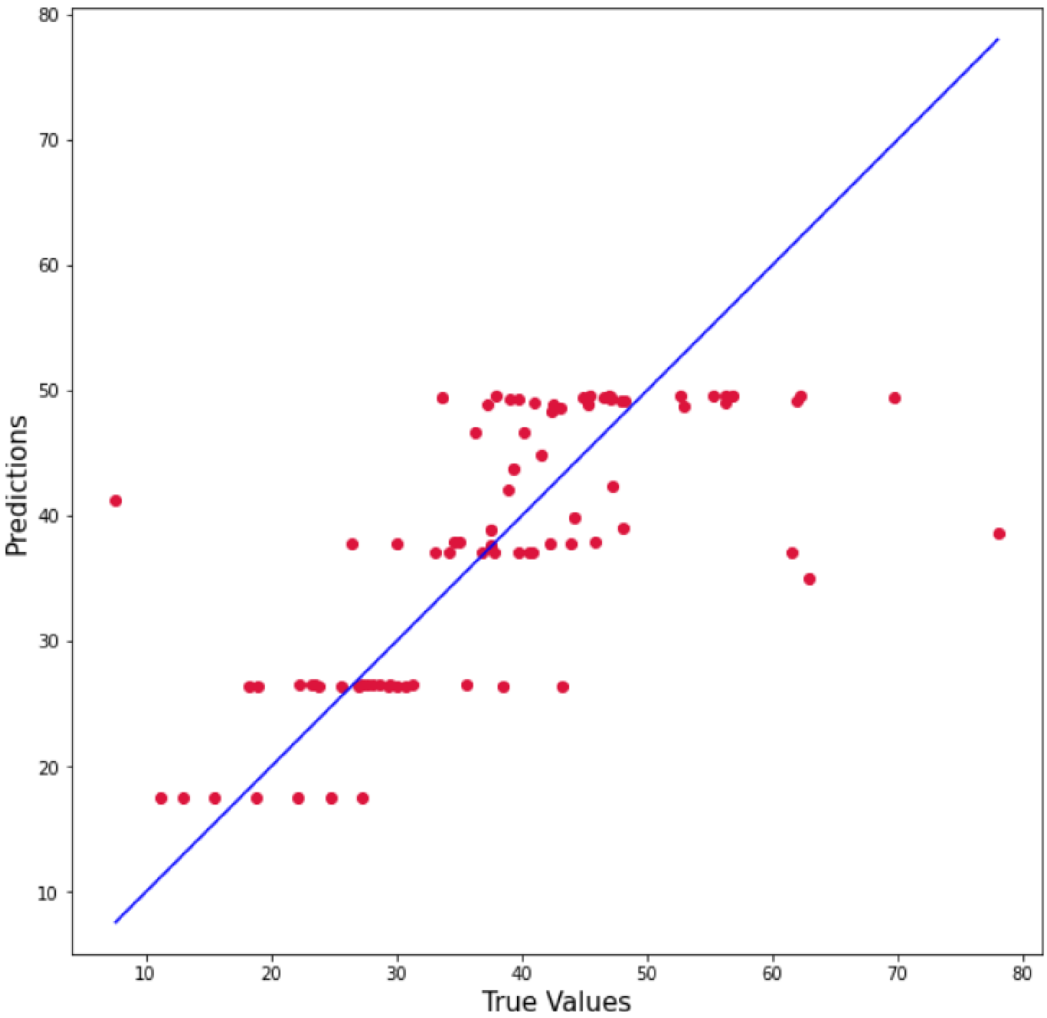
```
Bias =  0.6773006575517332

Mean Absolute Error = 6.660490841363061

Root Mean Squared Error = 2.5807926769430862

R-Squared = 0.47356780034212564
```

```
In [4]: plt.figure(figsize=(10,10))
        plt.scatter(y_test, y_pred, c='crimson')
        p1 = max(max(y_pred), max(y_test))
        p2 = min(min(y_pred), min(y_test))
        plt.plot([p1, p2], [p1, p2], 'b-')
        plt.xlabel('True Values', fontsize=15)
        plt.ylabel('Predictions', fontsize=15)
        plt.axis('equal')
        plt.show()
```



Predictions
True Values