Homework 4

William Wu

Problem 1: Shallow vs. Deep Neural Network ¶

(A) Generate the simulated data first using following equation. Sample 120k data as X from uniform distribution [-2*Pi, 2*Pi], then feed the sampled X into the equation to get Y. Randomly select 60K as training and 60 K as testing.

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets
from sklearn import tree
from sklearn.tree import DecisionTreeRegressor
from sklearn import svm, datasets, neural_network
from sklearn import preprocessing
from sklearn.model_selection import cross_val_score, train_test_split, GridSea
rchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, Ex
traTreesRegressor, GradientBoostingRegressor
from sklearn import linear_model
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, precision_recall
_fscore_support, mean_absolute_error
from sklearn.metrics import precision_recall_curve, average_precision_score, r
oc_curve, auc, mean_squared_error
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
x = np.random.uniform(low = -2*np.pi, high = 2*np.pi, size = 120000)
```

In [5]:
```
y = 2*(2*np.cos(x)**2 -1) ** 2 - 1
```

In [6]:
```
x_train, x_test, y_train, y_test = train_test_split(x.reshape(-1,1), y.reshape
(-1,1), test_size= 0.5, random_state = 1)
```

In [7]:
```
print (x_train.size)
print (x_test.size)
print(x_train, y_train)
```

```
60000
60000
[[ 0.76893146]
 [ 3.34659125]
 [-2.90859116]
 ...
 [ 1.32535508]
 [-3.98301734]
 [-3.7408262 ]] [[-0.99783157]
 [ 0.68222532]
 [ 0.59622476]
 ...
 [ 0.55555588]
 [-0.97499315]
 [-0.73531942]]
```

(b) Train 3 versions of Neural Network, with different numbers of hidden layer (NN with 1 hidden layer, 2 hidden layers and 3 hidden layers), using Mean squared error as objective function and error measurement

In [8]:
```
mlpr = neural_network.MLPRegressor()
param_list = {"hidden_layer_sizes": [1], "activation": ["identity", "logistic"
, "tanh", "relu"]}
grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5, scoring =
'neg_mean_squared_error')
grid.fit(x_train, y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-0.5012380646829551
{'activation': 'relu', 'hidden_layer_sizes': 1}
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=1, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Squared Error:  0.5016134794137688
```

In [9]:
```
mlpr = neural_network.MLPRegressor()
param_list = {"hidden_layer_sizes": [2], "activation": ["identity", "logistic"
, "tanh", "relu"]}
grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5, scoring =
'neg_mean_squared_error')
grid.fit(x_train, y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-0.5008348192055695
{'activation': 'relu', 'hidden_layer_sizes': 2}
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=2, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Squared Error:  0.4971653280399569
```

In [11]:
```
mlpr = neural_network.MLPRegressor()
param_list = {"hidden_layer_sizes": [3], "activation": ["identity", "logistic"
, "tanh", "relu"]}
grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5, scoring =
'neg_mean_squared_error')
grid.fit(x_train, y_train)


print (grid.best_score_)
print (grid.best_params_)
print (grid.best_estimator_)
print("Mean Squared Error: ",mean_squared_error(y_test, grid.predict(x_test)))
```

```
-0.4995072235179445
{'activation': 'relu', 'hidden_layer_sizes': 3}
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=3, learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
       random_state=None, shuffle=True, solver='adam', tol=0.0001,
       validation_fraction=0.1, verbose=False, warm_start=False)
Mean Squared Error:  0.4990389293731278
```

1 Hidden Layer(s): Mean Squared Error: 0.5016134794137688

2 Hidden Layer(s): Mean Squared Error: 0.4971653280399569

3 Hidden Layer(s): Mean Squared Error: 0.4990389293731278

c. For each version, try different number of neurals in your NN and replicate the following left plot (source: https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14849 (https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/viewPaper/14849)). (You don't need to replicate exactly same results below but need to show the performancedifference of 3 versions of Neural Networks)

In [12]:
```
hunits1 = [24, 48, 78, 128, 256]
hunits2 = [12, 24, 36]
hunits3 = [8, 16, 24]
```

In [13]:
```
def testerrors(layers, units):
    errorlist = []
    for i in units:
        mlpr = neural_network.MLPRegressor(hidden_layer_sizes= (layers, i))
        param_list = {"activation": ["identity", "logistic", "tanh", "relu"]}
        grid = GridSearchCV(estimator=mlpr, param_grid=param_list, cv = 5, sco
ring = 'neg_mean_squared_error')
        grid.fit(x_train, y_train)
        mse = mean_squared_error(y_test, grid.predict(x_test))
        errorlist += [mse]
    return errorlist
```

In [14]:
```
hl1 = testerrors(1,hunits1)
hl2 = testerrors(2,hunits2)
hl3 = testerrors(3,hunits3)
```

In [15]:
```
plt.plot(hunits1,hl1)
plt.plot(hunits2,hl2)
plt.plot(hunits3,hl3)
plt.title('f(x) = 2(2cos^2(x)-1)^2 - 1')
plt.xlabel('Number of Units')
plt.ylabel('Test Error')
plt.show()
```