In [69]:
```
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
import tkinter
import numpy
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score
# fix random seed for reproducibility
numpy.random.seed(7)
```

In [70]:
```
path = "C:/Users/sbekk/Downloads/housesalesprediction/kc_house_data.csv"
```

In [71]:
```
df=pd.read_csv(path)
```

In [72]:
```
#Clean up the unnecessary columns not required for analysis
df=df.drop(['id','date','zipcode','yr_renovated','view','waterfront'],1)
df.head(8)
```

Out[72]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	condition	grade	sqft_above	sqf
0	221900.0	3	1.00	1180	5650	1.0	3	7	1180	
1	538000.0	3	2.25	2570	7242	2.0	3	7	2170	
2	180000.0	2	1.00	770	10000	1.0	3	6	770	
3	604000.0	4	3.00	1960	5000	1.0	5	7	1050	
4	510000.0	3	2.00	1680	8080	1.0	3	8	1680	
5	1225000.0	4	4.50	5420	101930	1.0	3	11	3890	
6	257500.0	3	2.25	1715	6819	2.0	3	7	1715	
7	291850.0	3	1.50	1060	9711	1.0	3	7	1060	

In [73]:
```
#put predication column values in y and drop it from X which has other columns fo
y=df['price']
X=df.drop('price', axis=1)# df=df.values
# X.head(20)
```

In [74]:
```
#select 20 percent test and 80 percent training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
```

In [75]:
```
X_train=X_train.values
print(X_train.shape[1])
X_test=X_test.values
#Standardize the data
meanValue = X_train.mean(axis=0)
stdValue = X_train.std(axis=0)
X_train = (X_train-meanValue) / stdValue
X_test=(X_test-meanValue)/stdValue
print(X_train[0])
# define base model with neuron 130, layers=3, input=14 columns and output=1 colu
def build_ann_model():
  model = Sequential([Dense(130, activation=tf.nn.relu,input_shape=(X_train.shape
                      Dense(130, activation=tf.nn.relu),Dense(130, activation=tf.

  optimizer = tf.train.RMSPropOptimizer(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae'])
  return model

ann_model = build_ann_model()
```

```
14
[-0.39003389 -1.44813729 -0.55395019 -0.23023661 -0.91794048  0.9125681
 -0.56106598 -0.86348039  0.46635412 -1.05018286  0.96485795 -0.74469432
 -0.31691025 -0.23245796]
```

In [76]:
```
#Number of rounds(Epochs=500)

# Store training stats
history = ann_model.fit(X_train, y_train, epochs=500,
                    validation_split=0.2, verbose=0)
```

In [77]:
```
# model = Sequential()
# model.add(Dense(2, input_dim=14, activation='relu'))
# # model.add(Dense(3, activation='sigmoid'))
# model.add(Dense(1, activation='sigmoid'))
```

In [78]:
```
[loss, mae] = ann_model.evaluate(X_test, y_test, verbose=0)

print("Testing set Mean Abs Error: ${:f}".format(mae * 1000))
```

```
Testing set Mean Abs Error: $75884875.808719
```

In [79]:
```
y_pred = ann_model.predict(X_test).flatten()
print(y_pred)
mse = mean_squared_error(y_test, y_pred)
print(mse)
```

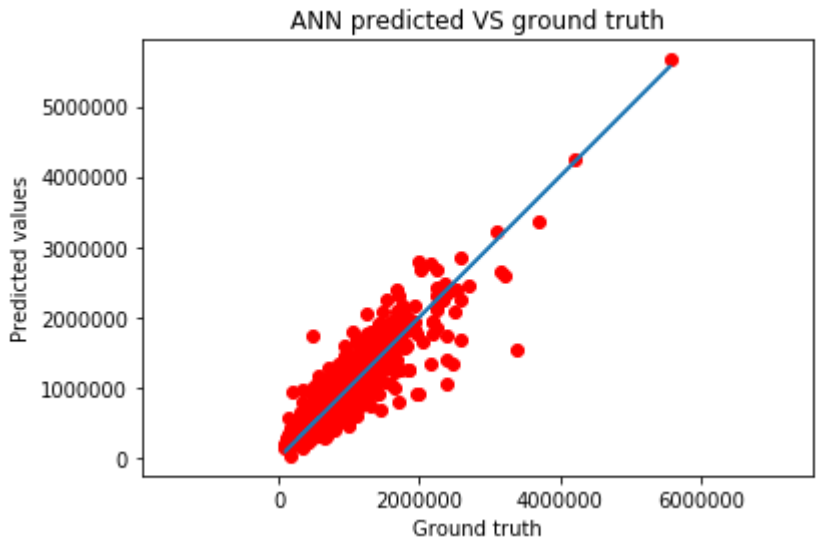
```
[ 360247.53 1927065.1   582564.75 ...  381194.1   220061.38  490633.47]
15966229803.899282
```

In [80]:
```
r2 = r2_score(y_test,y_pred)
print(r2)
```

```
0.8657445624990759
```

In [81]:
```
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
```

In [82]:
```
plt.title('ANN predicted VS ground truth')
plt.scatter(y_test, y_pred, color='r')
plt.xlabel('Ground truth')
plt.ylabel('Predicted values')
plt.axis('equal')
plt.xlim(plt.xlim())
plt.ylim(plt.ylim())
_=plt.plot(y_test,y_test)
```



In [39]:
```
#Second way to do ann on this data but, commenting out this since it wasn't givi
# X_train=X_train.values
# X_test=X_test.values
# y_train=y_train.values
# y_test=y_test.values
# X=X.values
# y=y.values
# X=X[0:20000]
# y=y[0:20000]
# X=X[0:1500]
# y=y[0:1500]
```

In [29]:
```
# define base model
# def baseline_model():
#     # create model
#     model = Sequential()
#     model.add(Dense(13, input_dim=14, kernel_initializer='normal', activation=
#     model.add(Dense(10,kernel_initializer='normal', activation='relu'))
#     model.add(Dense(8,kernel_initializer='normal', activation='relu'))
#     model.add(Dense(1, kernel_initializer='normal'))
#     # Compile model
#     model.compile(loss='mean_squared_error', optimizer='adam')
#     return model
```

In [14]:
```
# seed = 7
# numpy.random.seed(seed)
# # evaluate model with standardized dataset
# estimator = KerasRegressor(build_fn=baseline_model, epochs=10, batch_size=5, ve
# type(estimator)
# kfold = KFold(n_splits=10, random_state=seed)
# results = cross_val_score(estimator, X, y, cv=kfold)
# print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
# test_predictions = baseline_model().predict(X).flatten()
# print (test_predictions)
# r2_score(y,test_predictions)
# print(results)
# def coeff_determination(y_true, y_pred):
#     from keras import backend as K
#     SS_res =  K.sum(K.square( y_true-y_pred ))
#     SS_tot = K.sum(K.square( y_true - K.mean(y_true) ) )
#     return ( 1 - SS_res/(SS_tot + K.epsilon()) )
# evaluate model with standardized dataset
# numpy.random.seed(seed)
# estimators = []
# estimators.append(('standardize', StandardScaler()))
# estimators.append(('mlp', KerasRegressor(build_fn=baseline_model, epochs=50, ba
# pipeline = Pipeline(estimators)
# kfold = KFold(n_splits=10, random_state=seed)
# results = cross_val_score(pipeline, X, y, cv=kfold)
# print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```