Module7_NeuralNets_Tutorial

November 20, 2020

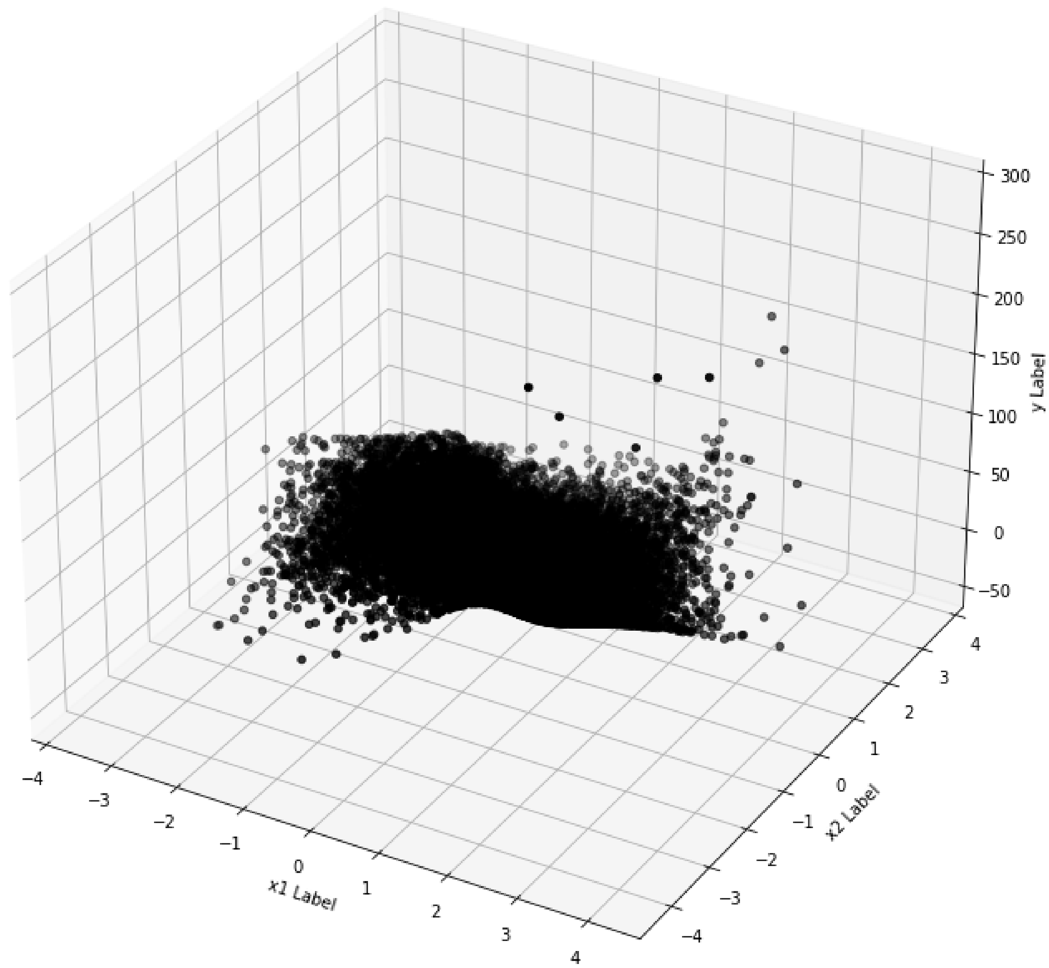
[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

from sklearn.preprocessing import scale
from sklearn.metrics import mean_squared_error, roc_curve, auc
import statsmodels.formula.api as smf
from sklearn.model_selection import train_test_split
```

[2]:
```python
# Simulated Example I
```

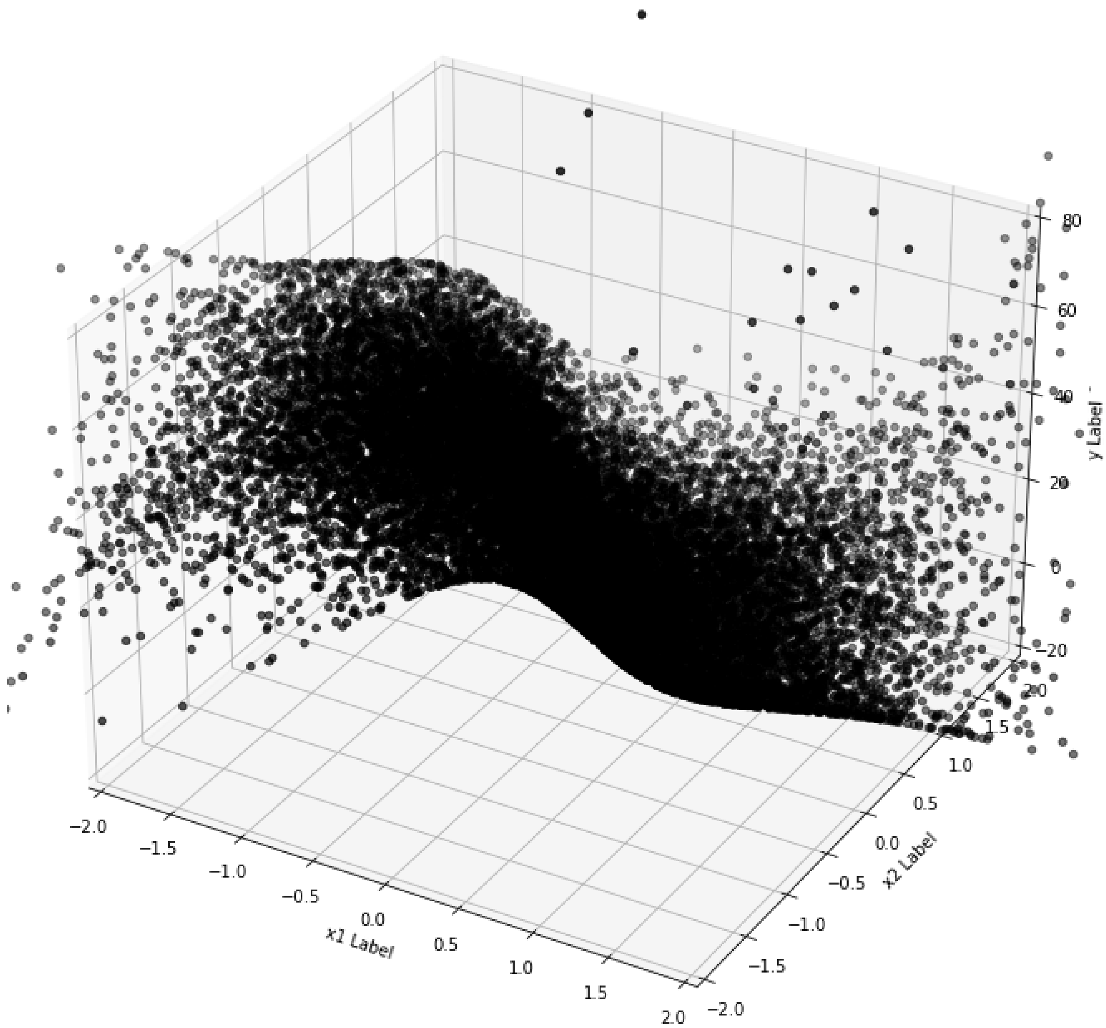
[3]:
```python
def sigmoid(x):
    return(1 / (1 + np.exp(-x)))
```

[4]:
```python
np.random.seed(1)
x1 = np.random.normal(0, 1, 20000)
x2 = np.random.normal(0, 1, 20000)
y = (x1+0.5)*(x1+0.5)*(x1+0.5)*(x2-0.3)*(x2-0.3)+50*1/(1+np.exp(3*x1+2*x2))
```

[5]:
```python
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y, c='k', marker='o')

ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[6]:
```
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y, c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,80)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[7]:
```
mydata1 = pd.DataFrame({'y':y,'x1':x1,'x2':x2})
```

[8]:
```
mydata1_sc = scale(mydata1)
```

[9]:
```
X_train = mydata1_sc[:,1:3]
y_train = mydata1_sc[:,0]
```

[10]:
```
inputs = keras.Input(shape=(2,))
x = layers.Dense(2, activation="sigmoid", name="dense_1")(inputs)
outputs = layers.Dense(1, activation="linear", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
```

[11]:
```
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),  # Optimizer
    # Loss function to minimize
    loss=keras.losses.MeanSquaredError(),
    # List of metrics to monitor
    metrics=[keras.metrics.MeanSquaredError()],
)
```

[12]:
```
history = model.fit(
    X_train,
    y_train,
    # batch_size=64,
    epochs=5
)
```

```
Train on 20000 samples
Epoch 1/5
20000/20000 [==============================] - 2s 95us/sample - loss: 0.3070 -
mean_squared_error: 0.3070s - loss: 0.5575
Epoch 2/5
20000/20000 [==============================] - 1s 61us/sample - loss: 0.1966 -
mean_squared_error: 0.1966
Epoch 3/5
20000/20000 [==============================] - 1s 50us/sample - loss: 0.1924 -
mean_squared_error: 0.1924
Epoch 4/5
20000/20000 [==============================] - 1s 53us/sample - loss: 0.1906 -
mean_squared_error: 0.1906
Epoch 5/5
20000/20000 [==============================] - 1s 51us/sample - loss: 0.1899 -
mean_squared_error: 0.1899
```

[13]:
```
pred_sc = model.predict(X_train)
preds_1 = pred_sc*np.std(mydata1).y + np.mean(mydata1).y
compare = pd.DataFrame({'y':y,'preds_1':preds_1.T[0]})
compare.head()
```

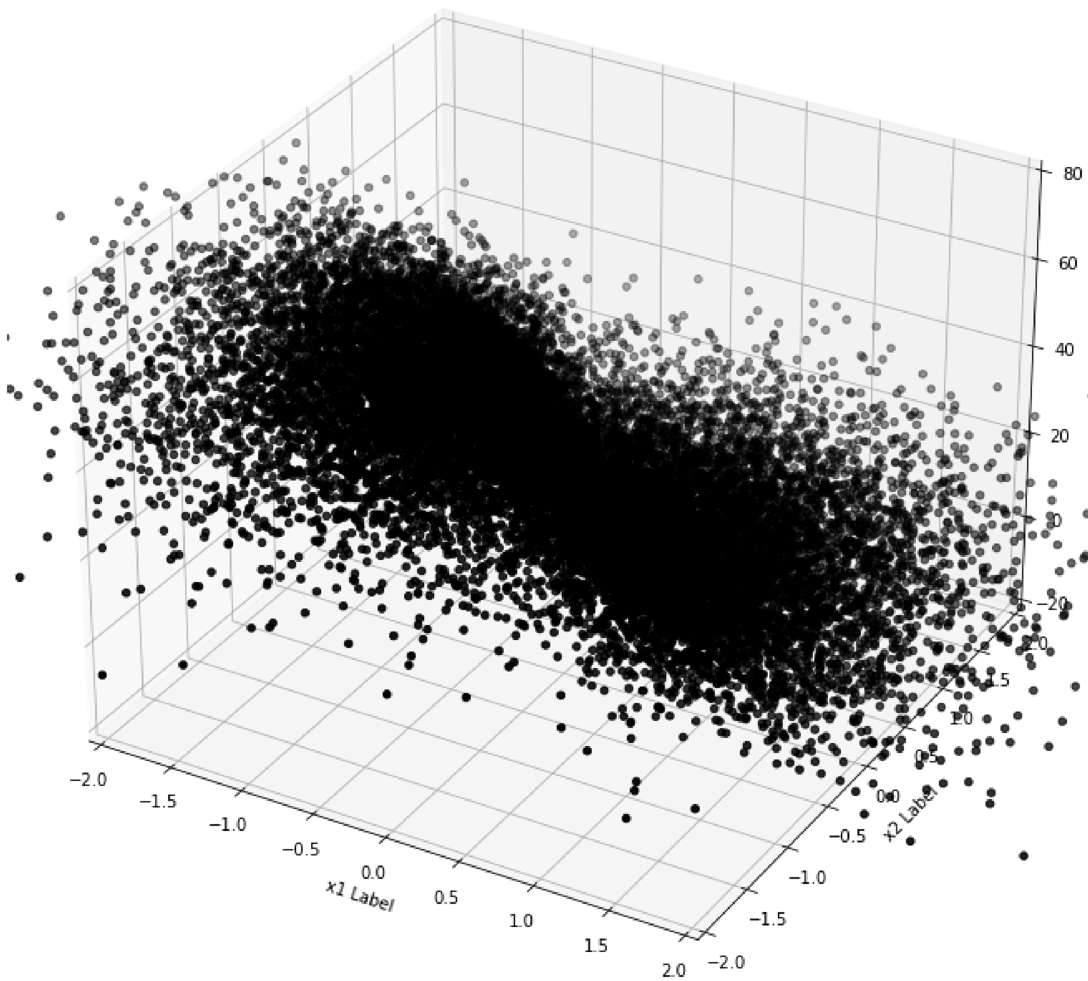
[13]:

	y	preds_1
0	61.091165	26.125568
1	44.811675	45.814064
2	48.131747	48.987610
3	48.661676	48.472984
4	1.816804	5.159172

[14]:
```
mean_squared_error(mydata1.y, preds_1.T[0])
```

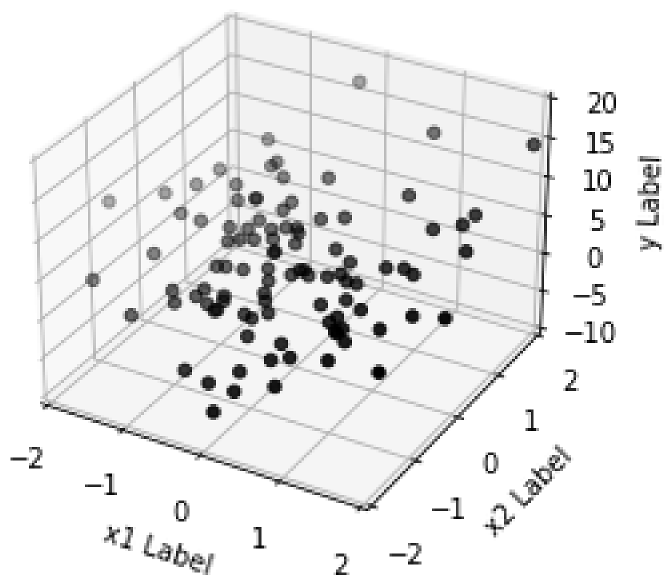
[14]: 73.26767286031229

[15]:
```
fig = plt.figure(figsize=(16, 12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, preds_1.T[0], c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,80)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



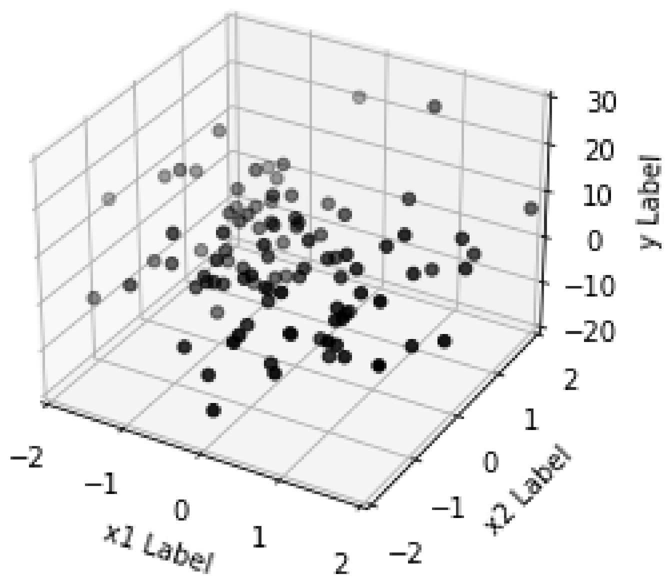
[16]:
```
# Simulated Example II
```

[17]:
```
np.random.seed(3)
x1 = np.random.normal(0, 1, 100)
x2 = np.random.normal(0, 1, 100)
x3 = np.random.normal(0, 1, 100)
y2_true = 3 + 2 * x1 + 4 * x2
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y2_true, c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-10,20)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[18]:
```
y2 = y2_true + 5 * np.random.normal(0, 1, 100)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y2, c='k', marker='o')

ax.set_xlim3d(-2,2)
```

```
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,30)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[19]:
```
mydata2 = pd.DataFrame({'y2':y2,'x1':x1,'x2':x2,'x3':x3})
mydata2_sc = scale(mydata2)
```

[ ]:
```

```

[20]:
```
X_train = mydata2_sc[:,1:4]
y_train = mydata2_sc[:,0]

inputs = keras.Input(shape=(3,))
x = layers.Dense(5, activation="sigmoid", name="dense_1")(inputs)
x = layers.Dense(3, activation="sigmoid", name="dense_2")(x)
outputs = layers.Dense(1, activation="linear", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),  # Optimizer
    # Loss function to minimize
    loss=keras.losses.MeanSquaredError(),
```

```
    # List of metrics to monitor
    metrics=[keras.metrics.MeanSquaredError()],
)

history = model.fit(
    X_train,
    y_train,
    # batch_size=64,
    epochs=5
)
```

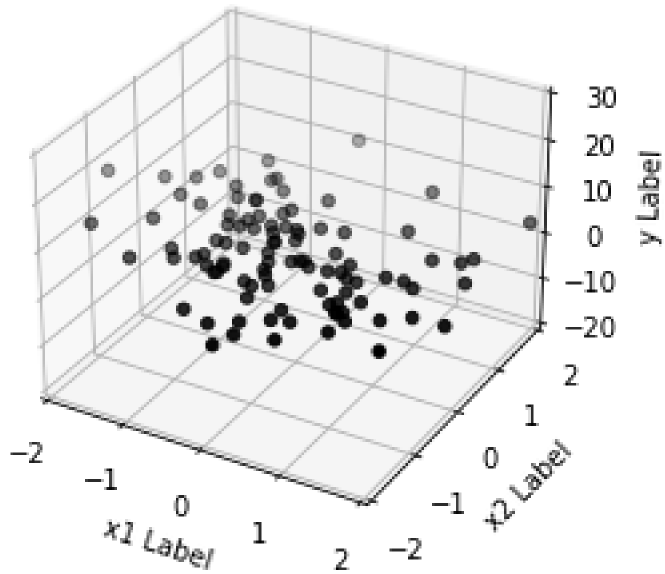
```
Train on 100 samples
Epoch 1/5
100/100 [==============================] - 1s 10ms/sample - loss: 1.0204 -
mean_squared_error: 1.0204
Epoch 2/5
100/100 [==============================] - 0s 332us/sample - loss: 0.9848 -
mean_squared_error: 0.9848
Epoch 3/5
100/100 [==============================] - 0s 489us/sample - loss: 0.9667 -
mean_squared_error: 0.9667
Epoch 4/5
100/100 [==============================] - 0s 340us/sample - loss: 0.9611 -
mean_squared_error: 0.9611
Epoch 5/5
100/100 [==============================] - 0s 264us/sample - loss: 0.9708 -
mean_squared_error: 0.9708
```

[21]:

```
pred2_sc = model.predict(X_train)
preds_2 = pred2_sc*np.std(mydata2).y2 + np.mean(mydata2).y2
```

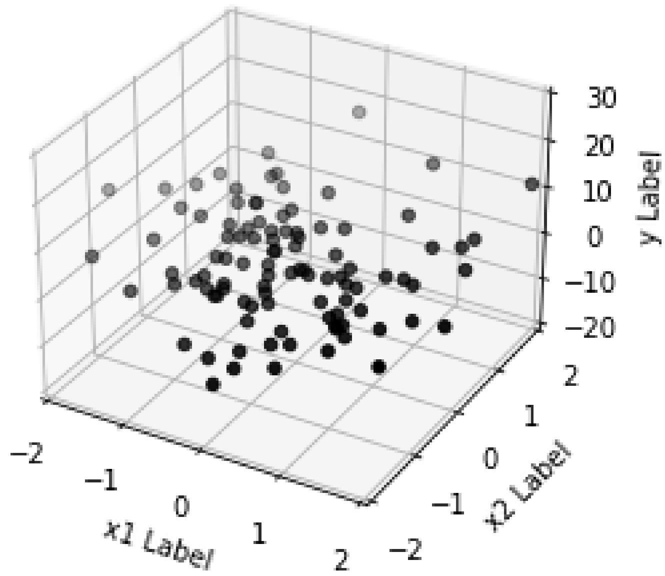
[22]:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, preds_2, c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,30)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[23]:

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y2_true-preds_2.T[0], c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,30)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[24]:
```
# single-layer networks with five neurons
```

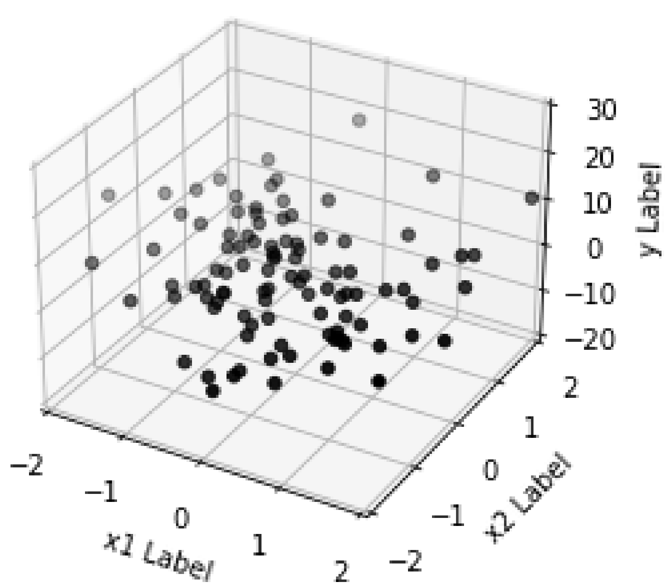
[25]:
```
inputs = keras.Input(shape=(3,))
x = layers.Dense(5, activation="sigmoid", name="dense_1")(inputs)
outputs = layers.Dense(1, activation="linear", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),  # Optimizer
    # Loss function to minimize
    loss=keras.losses.MeanSquaredError(),
    # List of metrics to monitor
    metrics=[keras.metrics.MeanSquaredError()],
)

history = model.fit(
    X_train,
    y_train,
    # batch_size=64,
    epochs=5
)
```

```
Train on 100 samples
Epoch 1/5
100/100 [==============================] - 1s 7ms/sample - loss: 1.0854 -
mean_squared_error: 1.0854
```

```
Epoch 2/5
100/100 [==============================] - 0s 201us/sample - loss: 1.0101 -
mean_squared_error: 1.0101
Epoch 3/5
100/100 [==============================] - 0s 267us/sample - loss: 0.9814 -
mean_squared_error: 0.9814
Epoch 4/5
100/100 [==============================] - 0s 404us/sample - loss: 0.9651 -
mean_squared_error: 0.9651
Epoch 5/5
100/100 [==============================] - 0s 316us/sample - loss: 0.9515 -
mean_squared_error: 0.9515
```

[26]:
```
pred2_2_sc = model.predict(X_train)
preds_2_2 = pred2_2_sc*np.std(mydata2).y2 + np.mean(mydata2).y2
```

[27]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y2_true-preds_2_2.T[0], c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,30)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[28]:
```
mean_squared_error(y2_true, preds_2_2.T[0])
```

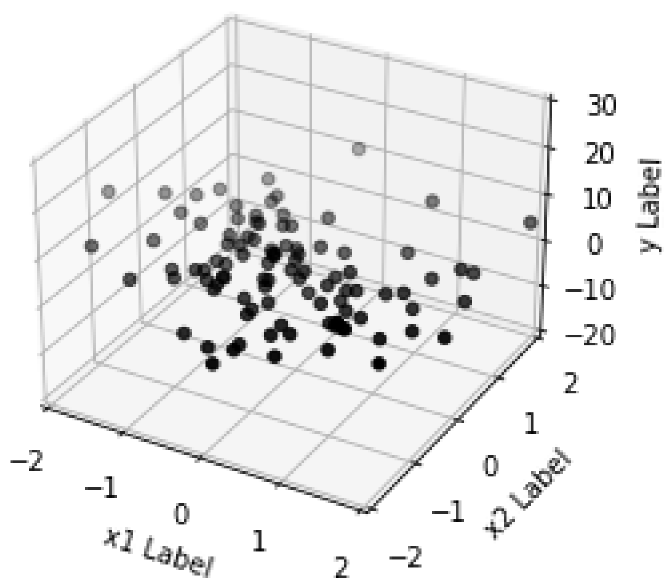
[28]: 12.106177665098372

[29]:
```
# With different hyperparameter: learning rate in this case

inputs = keras.Input(shape=(3,))
x = layers.Dense(5, activation="sigmoid", name="dense_1")(inputs)
outputs = layers.Dense(1, activation="linear", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.05),  # Optimizer
    # Loss function to minimize
    loss=keras.losses.MeanSquaredError(),
    # List of metrics to monitor
    metrics=[keras.metrics.MeanSquaredError()],
)

history = model.fit(
    X_train,
    y_train,
    # batch_size=64,
    epochs=5
)
```

```
Train on 100 samples
Epoch 1/5
100/100 [==============================] - 1s 10ms/sample - loss: 1.0917 -
mean_squared_error: 1.0917
Epoch 2/5
100/100 [==============================] - 0s 161us/sample - loss: 0.8764 -
mean_squared_error: 0.8764
Epoch 3/5
100/100 [==============================] - 0s 148us/sample - loss: 0.8084 -
mean_squared_error: 0.8084
Epoch 4/5
100/100 [==============================] - 0s 171us/sample - loss: 0.7499 -
mean_squared_error: 0.7499
Epoch 5/5
100/100 [==============================] - 0s 125us/sample - loss: 0.7324 -
mean_squared_error: 0.7324
```

[30]:
```
pred2_3_sc = model.predict(X_train)
preds_2_3 = pred2_3_sc*np.std(mydata2).y2 + np.mean(mydata2).y2
```

```python
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x1, x2, y2_true-preds_2_3.T[0], c='k', marker='o')

ax.set_xlim3d(-2,2)
ax.set_ylim3d(-2,2)
ax.set_zlim3d(-20,30)
ax.set_xlabel('x1 Label')
ax.set_ylabel('x2 Label')
ax.set_zlabel('y Label')

plt.show()
```



[31]:
```python
mean_squared_error(y2_true, preds_2_3.T[0])
```

[31]: 1.7573505065580433

[32]:
```python
lmfit = smf.ols(formula="y2 ~ x1 + x2 + x3", data=mydata2).fit()
yhat_OOS = lmfit.predict(mydata2.drop(columns = ["y2"]))
print(lmfit.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                     y2   R-squared:                       0.309
Model:                            OLS   Adj. R-squared:                  0.288
Method:                 Least Squares   F-statistic:                     14.33
```

```
Date:                Fri, 20 Nov 2020   Prob (F-statistic):          8.68e-08
Time:                        14:24:45   Log-Likelihood:               -308.23
No. Observations:                 100   AIC:                            624.5
Df Residuals:                      96   BIC:                            634.9
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      2.2953      0.561      4.089      0.000       1.181       3.410
x1             1.8233      0.524      3.481      0.001       0.784       2.863
x2             3.8645      0.634      6.096      0.000       2.606       5.123
x3            -0.7185      0.504     -1.426      0.157      -1.719       0.282
==============================================================================
Omnibus:                        0.466   Durbin-Watson:                   2.037
Prob(Omnibus):                  0.792   Jarque-Bera (JB):                0.577
Skew:                          -0.150   Prob(JB):                        0.749
Kurtosis:                       2.779   Cond. No.                         1.52
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

[33]:
```
mean_squared_error(y2_true, yhat_OOS)
```

[33]:
```
1.4136322850403886
```

[34]:
```
# Credit Card Defaults
```

[35]:
```
mydata = pd.read_csv('UCI_Credit_Card.csv', index_col=0)
```

[36]:
```
mydata.head()
```

[36]:
```
    LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
ID
1     20000.0    2          2         1   24      2      2     -1     -1
2    120000.0    2          2         2   26     -1      2      0      0
3     90000.0    2          2         2   34      0      0      0      0
4     50000.0    2          2         1   37      0      0      0      0
5     50000.0    1          2         1   57     -1      0     -1      0

    PAY_5  …  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
ID         …
1      -2  …        0.0        0.0        0.0       0.0     689.0       0.0
2       0  …     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
3       0  …    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
```

```
4        0  …     28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
5        0  …     20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

    PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
ID
1        0.0       0.0       0.0                           1
2     1000.0       0.0    2000.0                           1
3     1000.0    1000.0    5000.0                           0
4     1100.0    1069.0    1000.0                           0
5     9000.0     689.0     679.0                           0

[5 rows x 24 columns]
```

[37]:
```python
mydata.describe()
```

[37]:
```
            LIMIT_BAL           SEX     EDUCATION      MARRIAGE           AGE  \
count    30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean    167484.322667      1.603733      1.853133      1.551867     35.485500
std     129747.661567      0.489129      0.790349      0.521970      9.217904
min      10000.000000      1.000000      0.000000      0.000000     21.000000
25%      50000.000000      1.000000      1.000000      1.000000     28.000000
50%     140000.000000      2.000000      2.000000      2.000000     34.000000
75%     240000.000000      2.000000      2.000000      2.000000     41.000000
max    1000000.000000      2.000000      6.000000      3.000000     79.000000

              PAY_0         PAY_2         PAY_3         PAY_4         PAY_5  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean      -0.016700     -0.133767     -0.166200     -0.220667     -0.266200
std        1.123802      1.197186      1.196868      1.169139      1.133187
min       -2.000000     -2.000000     -2.000000     -2.000000     -2.000000
25%       -1.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%        0.000000      0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000      0.000000
max        8.000000      8.000000      8.000000      8.000000      8.000000

       …      BILL_AMT4      BILL_AMT5      BILL_AMT6       PAY_AMT1  \
count  …   30000.000000   30000.000000   30000.000000   30000.000000
mean   …   43262.948967   40311.400967   38871.760400    5663.580500
std    …   64332.856134   60797.155770   59554.107537   16563.280354
min    … -170000.000000  -81334.000000 -339603.000000       0.000000
25%    …    2326.750000    1763.000000    1256.000000    1000.000000
50%    …   19052.000000   18104.500000   17071.000000    2100.000000
75%    …   54506.000000   50190.500000   49198.250000    5006.000000
max    …  891586.000000  927171.000000  961664.000000  873552.000000

           PAY_AMT2      PAY_AMT3       PAY_AMT4       PAY_AMT5  \
count  3.000000e+04   30000.00000   30000.000000   30000.000000
```

```
mean   5.921163e+03    5225.68150    4826.076867    4799.387633
std    2.304087e+04   17606.96147   15666.159744   15278.305679
min    0.000000e+00       0.00000       0.000000       0.000000
25%    8.330000e+02     390.00000     296.000000     252.500000
50%    2.009000e+03    1800.00000    1500.000000    1500.000000
75%    5.000000e+03    4505.00000    4013.250000    4031.500000
max    1.684259e+06  896040.00000  621000.000000  426529.000000

            PAY_AMT6  default.payment.next.month
count   30000.000000                30000.000000
mean     5215.502567                    0.221200
std     17777.465775                    0.415062
min         0.000000                    0.000000
25%       117.750000                    0.000000
50%      1500.000000                    0.000000
75%      4000.000000                    0.000000
max    528666.000000                    1.000000

[8 rows x 24 columns]
```

[38]:
```
factor = ['SEX', 'EDUCATION', 'MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4',␣
 ↪'PAY_5', 'PAY_6', 'default.payment.next.month']
mydata_numcols = mydata.drop(columns = factor)
mydata_faccols = mydata[factor].drop(columns = ['default.payment.next.month']).
 ↪astype('category')
dummies = pd.get_dummies(mydata_faccols, drop_first=True)
mydata_numcols_sc_0 = scale(mydata_numcols)
mydata_numcols_sc = pd.DataFrame(data=mydata_numcols_sc_0, columns =␣
 ↪mydata_numcols.columns, index = dummies.index)
mydata_sc = pd.concat([mydata_numcols_sc, dummies], axis = 1)
mydata_sc = pd.concat([mydata_sc, mydata['default.payment.next.month']], axis =␣
 ↪1)
```

[39]:
```
train, test = train_test_split(mydata_sc, test_size=0.25)
X_train = train.drop(columns = ['default.payment.next.month']).values
y_train = train['default.payment.next.month'].values
X_test = test.drop(columns = ['default.payment.next.month']).values
y_test = test['default.payment.next.month'].values
```

[40]:
```
inputs = keras.Input(shape=(82,))
x = layers.Dense(5, activation="relu", name="dense_1")(inputs)
x = layers.Dense(3, activation="sigmoid", name="dense_2")(x)
outputs = layers.Dense(1, activation="sigmoid", name="predictions")(x)

model = keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),  # Optimizer
```

```python
    # Loss function to minimize
    loss='binary_crossentropy',
    # List of metrics to monitor
    metrics=['accuracy'],
)

history = model.fit(
    X_train,
    y_train,
    # batch_size=64,
    epochs=15
)
```

```
Train on 22500 samples
Epoch 1/15
22500/22500 [==============================] - 4s 174us/sample - loss: 0.4542 -
accuracy: 0.8048
Epoch 2/15
22500/22500 [==============================] - 2s 81us/sample - loss: 0.4399 -
accuracy: 0.8172
Epoch 3/15
22500/22500 [==============================] - 2s 80us/sample - loss: 0.4372 -
accuracy: 0.8197
Epoch 4/15
22500/22500 [==============================] - 2s 84us/sample - loss: 0.4351 -
accuracy: 0.8194
Epoch 5/15
22500/22500 [==============================] - 2s 85us/sample - loss: 0.4338 -
accuracy: 0.8207
Epoch 6/15
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4330 -
accuracy: 0.8192
Epoch 7/15
22500/22500 [==============================] - 2s 80us/sample - loss: 0.4318 -
accuracy: 0.8212
Epoch 8/15
22500/22500 [==============================] - 2s 82us/sample - loss: 0.4315 -
accuracy: 0.8206
Epoch 9/15
22500/22500 [==============================] - 2s 80us/sample - loss: 0.4307 -
accuracy: 0.8217
Epoch 10/15
22500/22500 [==============================] - 2s 80us/sample - loss: 0.4301 -
accuracy: 0.8204
Epoch 11/15
22500/22500 [==============================] - 2s 83us/sample - loss: 0.4300 -
accuracy: 0.8216
```

```
Epoch 12/15
22500/22500 [==============================] - 2s 82us/sample - loss: 0.4293 -
accuracy: 0.8216
Epoch 13/15
22500/22500 [==============================] - 2s 91us/sample - loss: 0.4285 -
accuracy: 0.8213
Epoch 14/15
22500/22500 [==============================] - 2s 81us/sample - loss: 0.4285 -
accuracy: 0.8224
Epoch 15/15
22500/22500 [==============================] - 3s 120us/sample - loss: 0.4283 -
accuracy: 0.8220
```

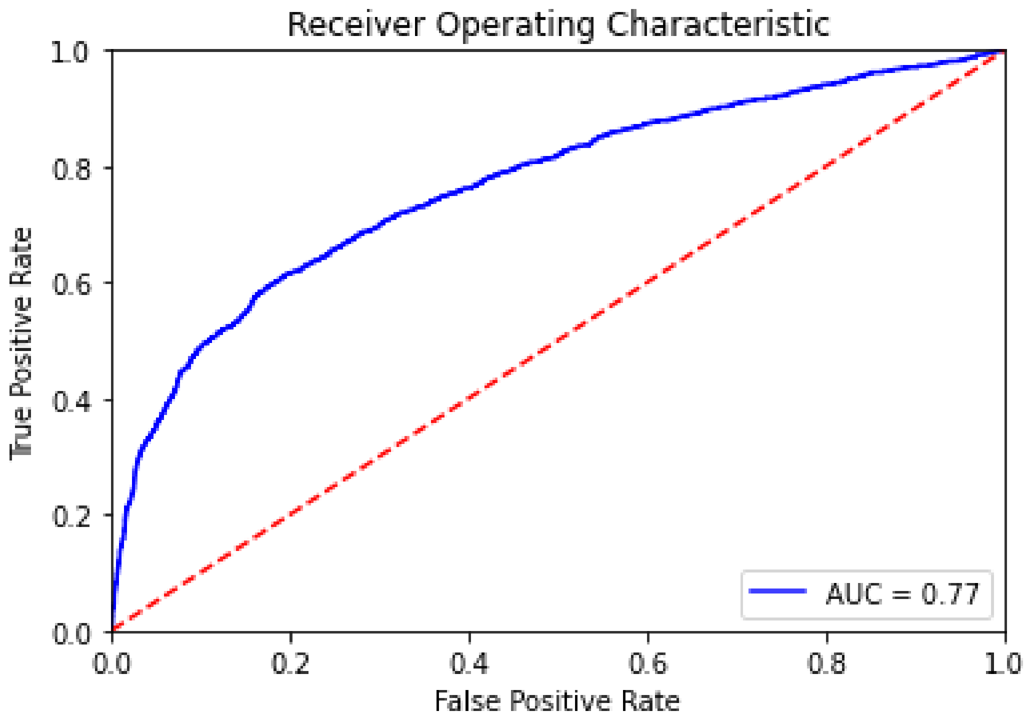
[41]:
```
mypreds = model.predict(X_test)
```

[42]:
```
mypreds_bin = mypreds > 0.5
table = pd.DataFrame({'True':y_test,'pred':mypreds_bin.T[0]})
table.groupby(['True','pred']).size().unstack('True')
```

[42]:

True	0	1
pred		
False	5624	1049
True	276	551

[43]:
```
fpr, tpr, threshold = roc_curve(y_test, mypreds)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.77
0.0
0.2
0.4
0.6
0.8
1.0

[ ]:

```

```