r

29 2021 .

1 № 3

1.1 .

: ..

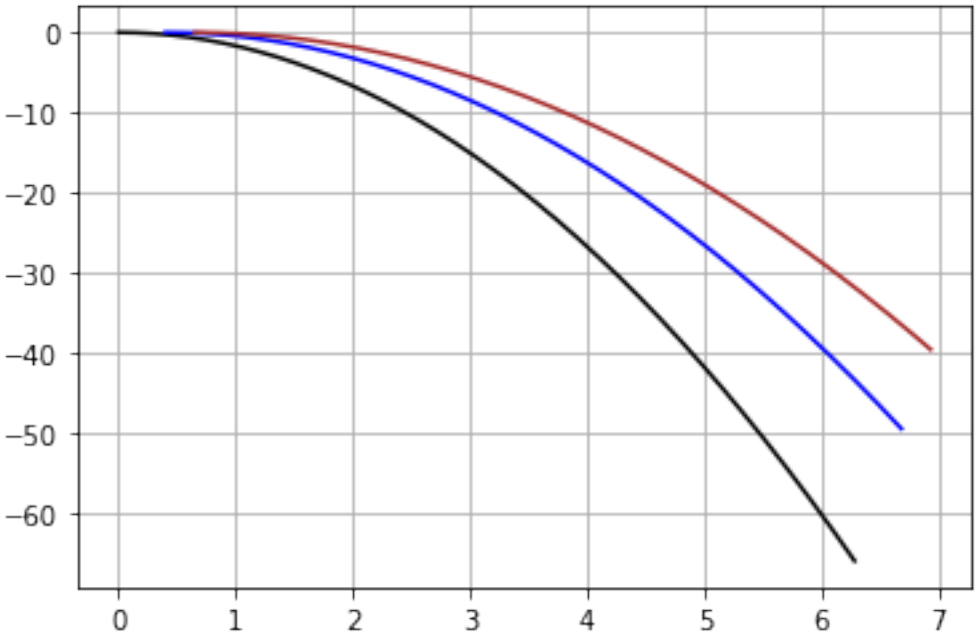
[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.model_selection import train_test_split
```

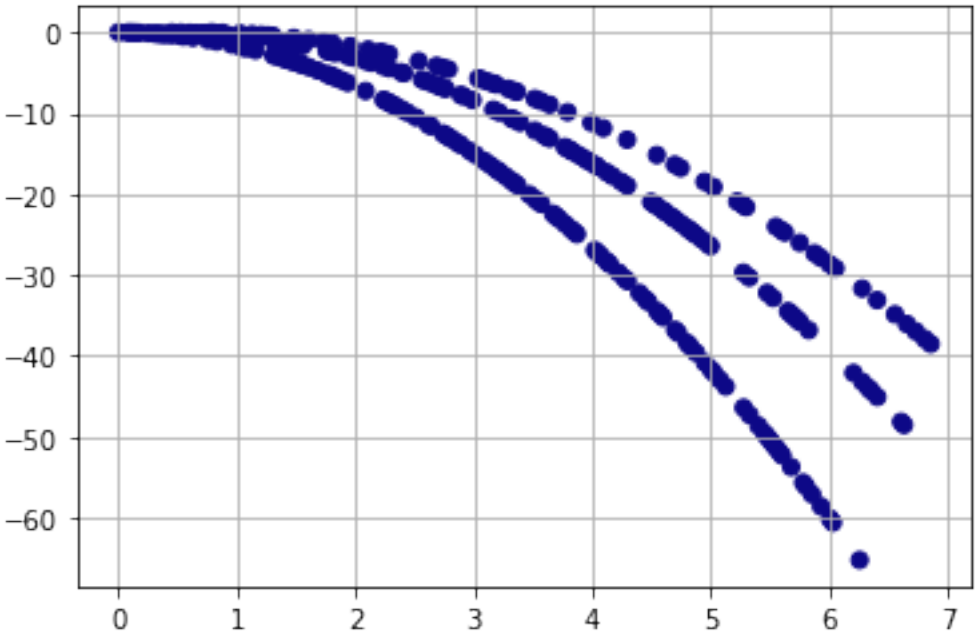
[ ]:
```
step = 0.025
tPoints = np.linspace(0, 2*np.pi, int(2*np.pi/step), endpoint=True)

def split_df(df):
    x_train, x_test = train_test_split(df, test_size=0.3, shuffle=True,␣
 ↪random_state=20)
    x_valid, x_test = train_test_split(x_test, test_size=0.3, shuffle=True,␣
 ↪random_state=76)
    return x_train, x_valid, x_test
```

[ ]:
```
p1 = -0.3
shiftX1 = 0

p2 = -0.4
shiftX2 = 0.4

p3 = -0.5
shiftX3 = 0.65
```

[ ]:
```
first = np.random.permutation(tPoints)[:120]
second = np.random.permutation(tPoints)[:100]
third = np.random.permutation(tPoints)[:60]
```

[ ]:
```
x1 = tPoints + shiftX1
y1 = tPoints * tPoints / (2*p1)

x2 = tPoints + shiftX2
y2 = tPoints * tPoints / (2*p2)

x3 = tPoints + shiftX3
y3 = tPoints * tPoints / (2*p3)

plt.plot(x1, y1, 'black')
plt.plot(x2, y2, 'blue')
plt.plot(x3, y3, 'brown')
plt.grid(True)
plt.show()

x1 = first + shiftX1
y1 = first * first / (2*p1)

x2 = second + shiftX2
y2 = second * second / (2*p2)

x3 = third + shiftX3
y3 = third * third / (2*p3)


df1 = pd.DataFrame({'x' : x1, 'y' : y1, 'target' : 0})
df2 = pd.DataFrame({'x' : x2, 'y' : y2, 'target' : 1})
df3 = pd.DataFrame({'x' : x3, 'y' : y3, 'target' : 2})

for idx, df in enumerate((df1, df2, df3)):
    plt.scatter(df.x, df.y, c= df.target, cmap=plt.cm.plasma)
    plt.grid(True)
plt.show()
```





[ ]:
```
train = []
valid = []
test = []
```

```
for df in (df1, df2 ,df3):
    tr, v, te = split_df(df)
    train.append(tr)
    valid.append(v)
    test.append(te)
train = pd.concat(train)
valid = pd.concat(valid)
test = pd.concat(test)
```

[ ]:
```
print(len(train))
print(len(valid))
print(len(test))
```

```
196
58
26
```

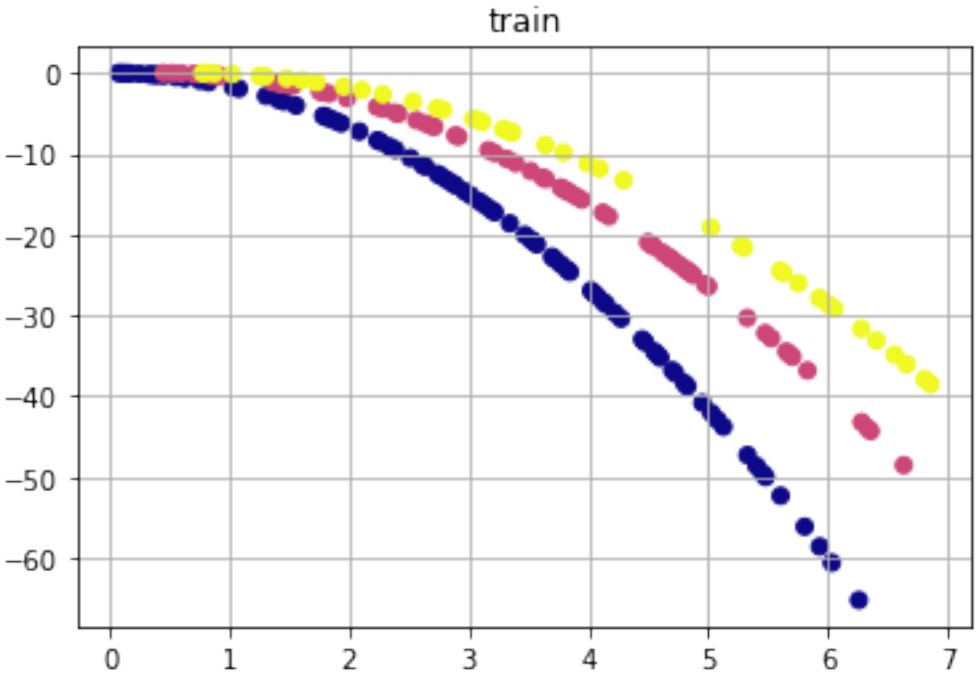
[ ]:
```
model = Sequential()
model.add(Dense(20, input_shape=(2,), activation='tanh'))
model.add(Dense(3, activation='softmax'))
model.compile(Adam(learning_rate=0.01), 'categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

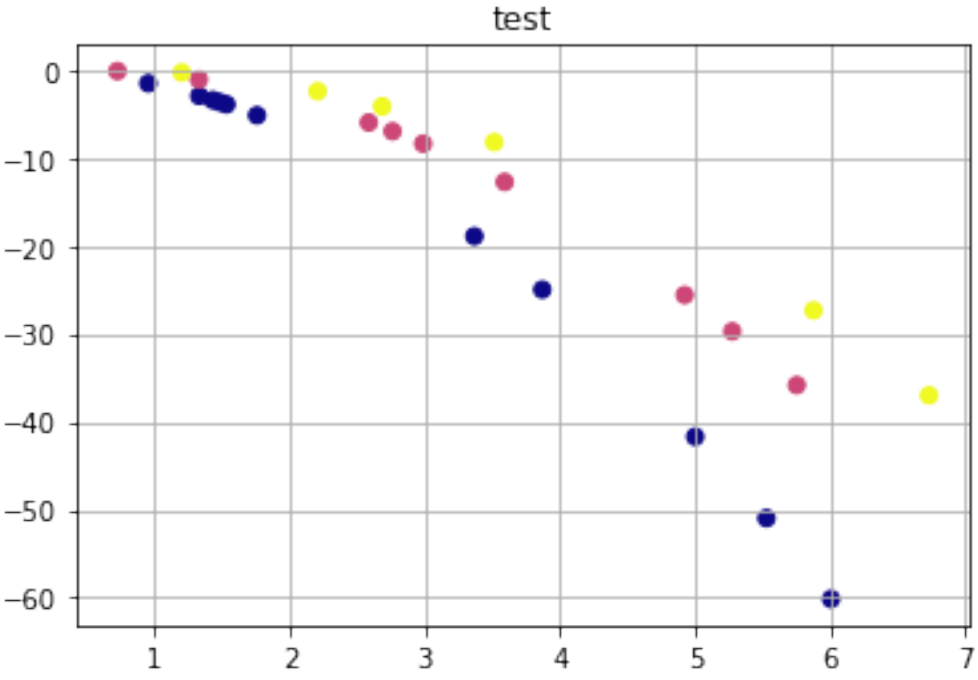
[ ]:
```
y = pd.get_dummies(train['target'])
history = model.fit(train.iloc[:, :-1], y, epochs=1500, verbose=False,␣
 ↪shuffle=True)

p = []

p.append(np.argmax(model.predict(train.iloc[:, :-1]), axis=-1))
print(accuracy_score(train['target'], p[-1]))

p.append(np.argmax(model.predict(test.iloc[:, :-1]), axis=-1))
print(accuracy_score(test['target'], p[-1]))

p.append(np.argmax(model.predict(valid.iloc[:, :-1]), axis=-1))
print(accuracy_score(valid['target'], p[-1]))
```

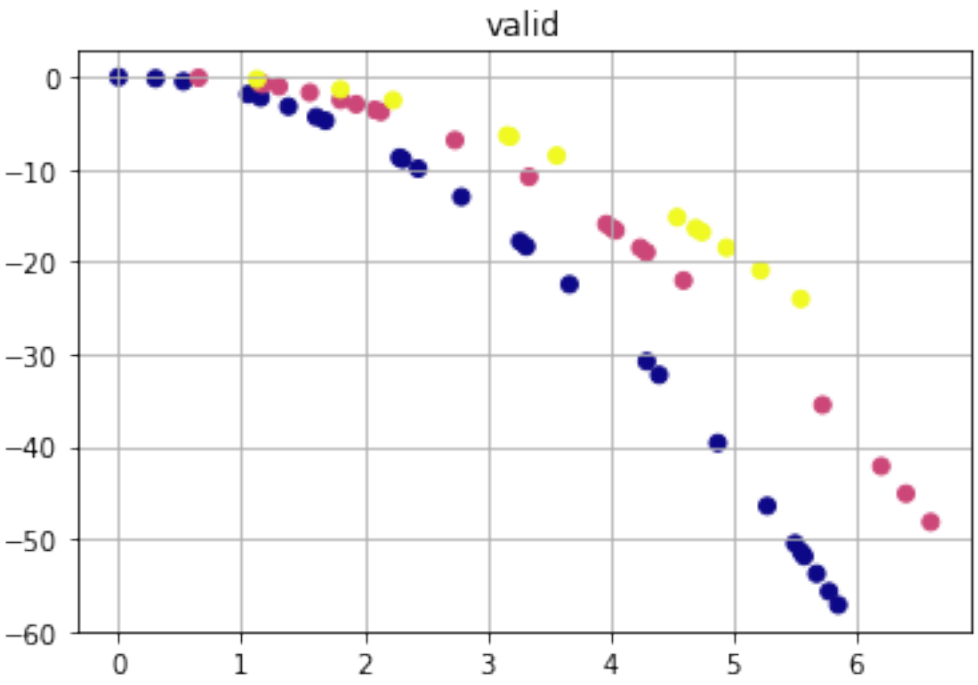
```
1.0
1.0
1.0
```

[ ]:
```
titles = ['train', 'test', 'valid']

for idx, df in enumerate((train, test, valid)):
    plt.scatter(df.x, df.y, c=p[idx], cmap=plt.cm.plasma)
    plt.grid(True)
    plt.title(titles[idx])
```

```
    plt.show()
```

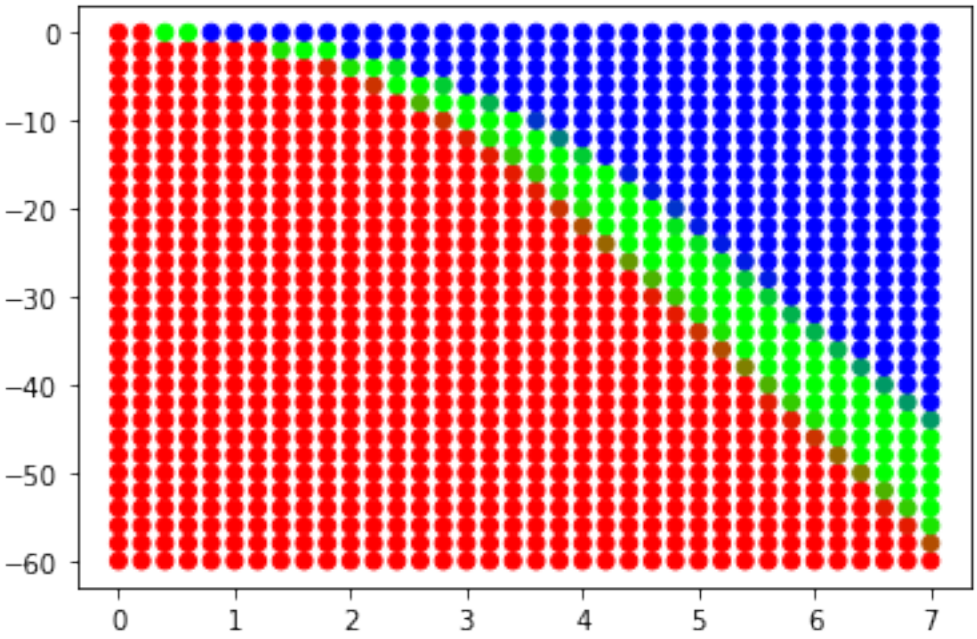

train
0
−10
−20
−30
−40
−50
−60
0
1
2
3
4
5
6
7


test
0
−10
−20
−30
−40
−50
−60
1
2
3
4
5
6
7



[ ]:
```
hx = 0.2
hy = 2
grid_pred = [model.predict(np.array([[i, j]])).round(1) for i in np.arange(0,␣
↪7+hx, hx) for j in np.arange(-60, 0+hy, hy)]
```

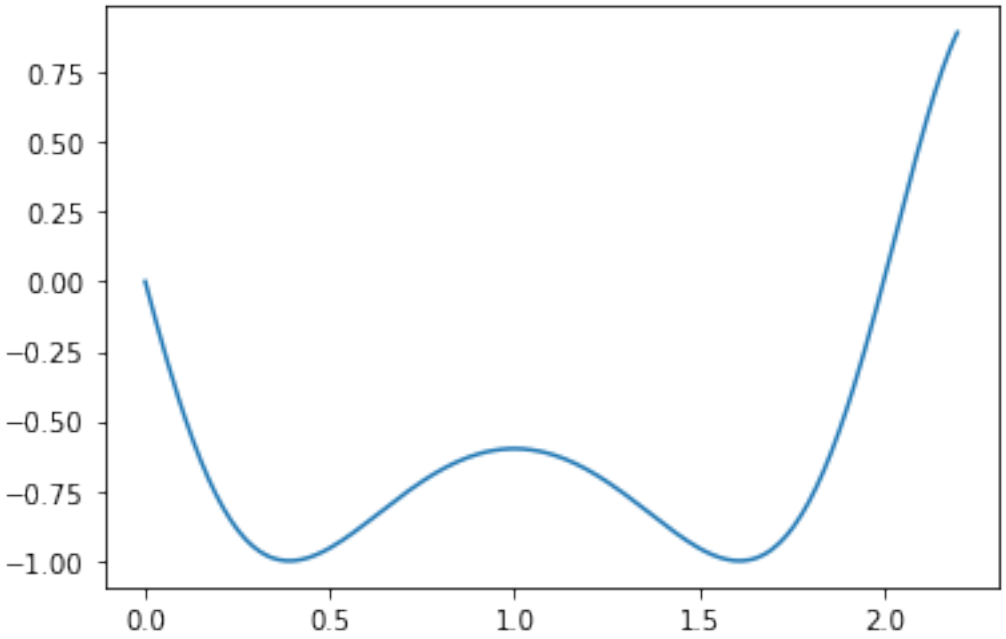
[ ]:
```
x_vals = np.arange(-60, 0+hy, hy)
y_vals = np.arange(0, 7+hx, hx)

xx, yy = np.meshgrid(x_vals, y_vals)

rows = len(grid_pred)
colors = np.array(grid_pred).reshape((rows, 3))
colors.shape


plt.scatter(yy, xx, c=colors, cmap=plt.cm.plasma);
plt.show()
```



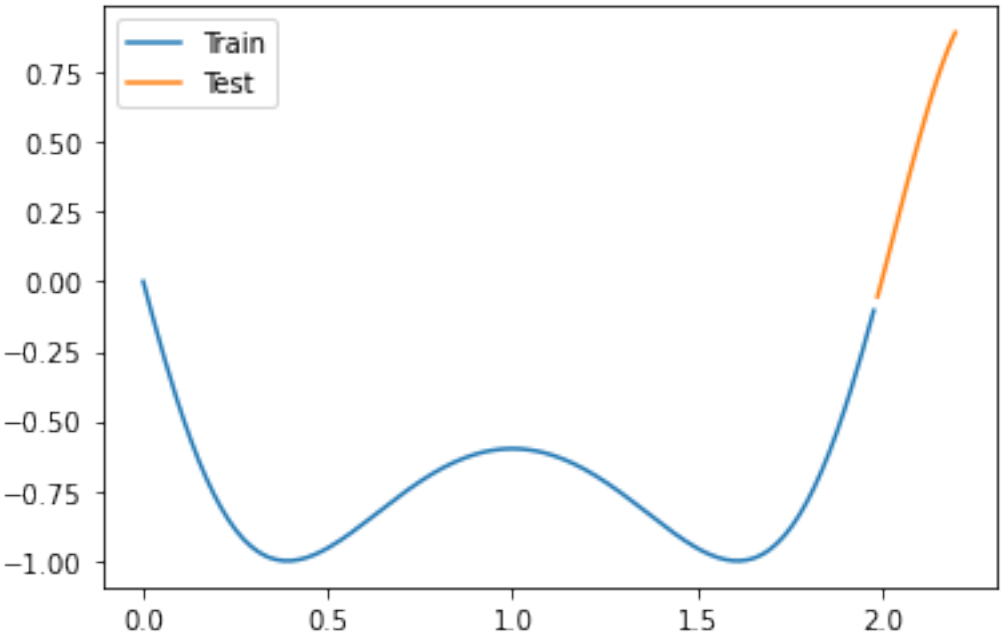
[ ]:
```
#
```

[ ]:
```
h = 0.01
t = np.linspace(0, 2.2, int(2.2/h), endpoint=True)
x = np.sin( (2.5) * t ** 2 -5*t)
plt.plot(t, x)
print(len(t))
```

```
220
```



[ ]:
```
train_size = int(len(t) * 0.9)

X_train = t[:train_size]
y_train = x[:train_size]
plt.plot(X_train, y_train, label='Train')

X_test = t[train_size:]
y_test = x[train_size:]

plt.plot(X_test, y_test, label = 'Test')
plt.legend()
plt.show()
```

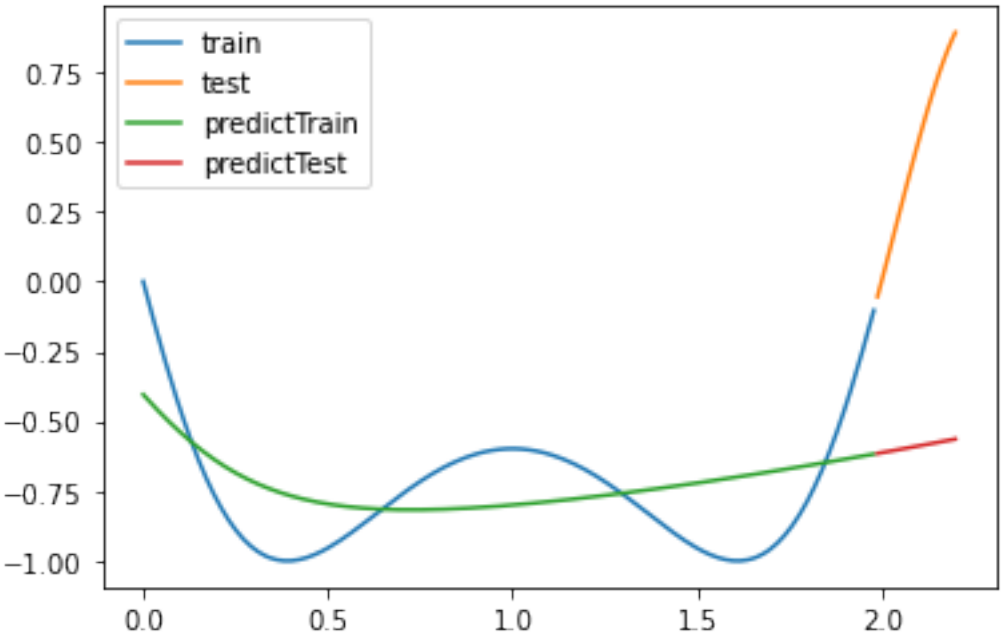


[ ]:
```
model = Sequential()
model.add(Dense(30, input_shape=(1,), activation='tanh'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

[ ]:
```
history = model.fit(X_train, y_train , epochs=600, verbose=0)
```

[ ]:
```
plt.plot(X_train, y_train, label='train')
plt.plot(X_test, y_test, label='test')

pred_x = model.predict(X_train[:])
mse = mean_squared_error(y_train, pred_x.flatten())
plt.plot(X_train, pred_x, label='predictTrain')
print(f'RMSE on train = {np.sqrt(mse)} ')

pred_x = model.predict(X_test[:])
mse = mean_squared_error(y_test, pred_x.flatten())
plt.plot(X_test, pred_x, label='predictTest')
print(f'RMSE on test = {np.sqrt(mse)}')

plt.legend()
plt.show()
```

```
RMSE on train = 0.19247786005621578
RMSE on test = 1.0809122695034137
```



[ ]:
```
#
```

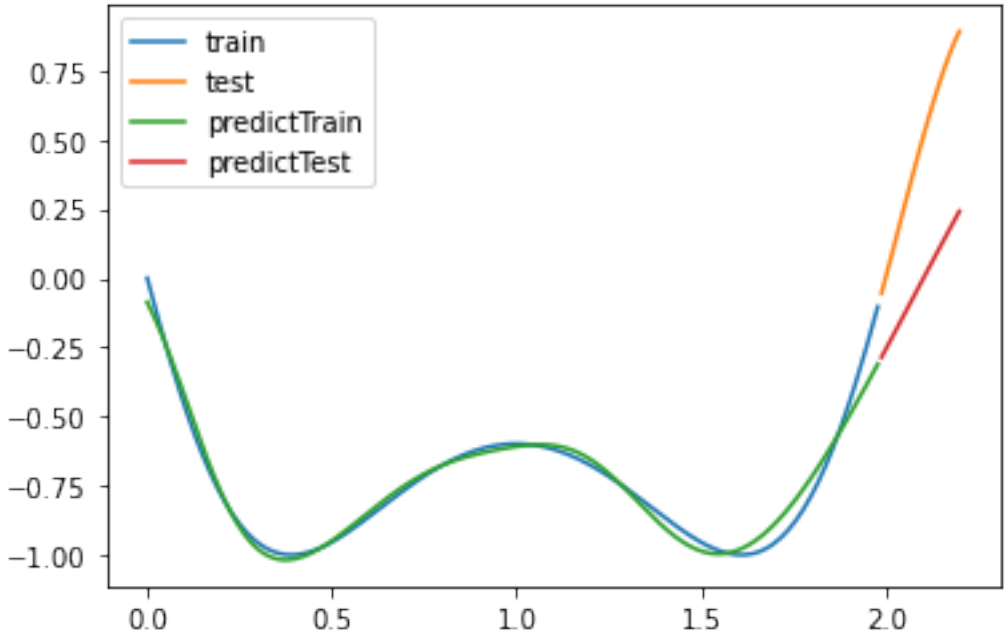
[ ]:
```
model = Sequential()
model.add(Dense(5, input_shape=(1,), activation='tanh'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

[ ]:
```
history = model.fit(X_train, y_train , epochs=10000, verbose=0)
```

[ ]:
```
plt.plot(X_train, y_train, label='train')
plt.plot(X_test, y_test, label='test')

pred_x = model.predict(X_train[:])
mse = mean_squared_error(y_train, pred_x.flatten())
plt.plot(X_train, pred_x, label='predictTrain')
print(f'RMSE on train = {np.sqrt(mse)} ')

pred_x = model.predict(X_test[:])
mse = mean_squared_error(y_test, pred_x.flatten())
plt.plot(X_test, pred_x, label='predictTest')
print(f'RMSE on test = {np.sqrt(mse)}')

plt.legend()
plt.show()
```

```
RMSE on train = 0.04007339514058405
```

```
RMSE on test = 0.49465601150255484
```



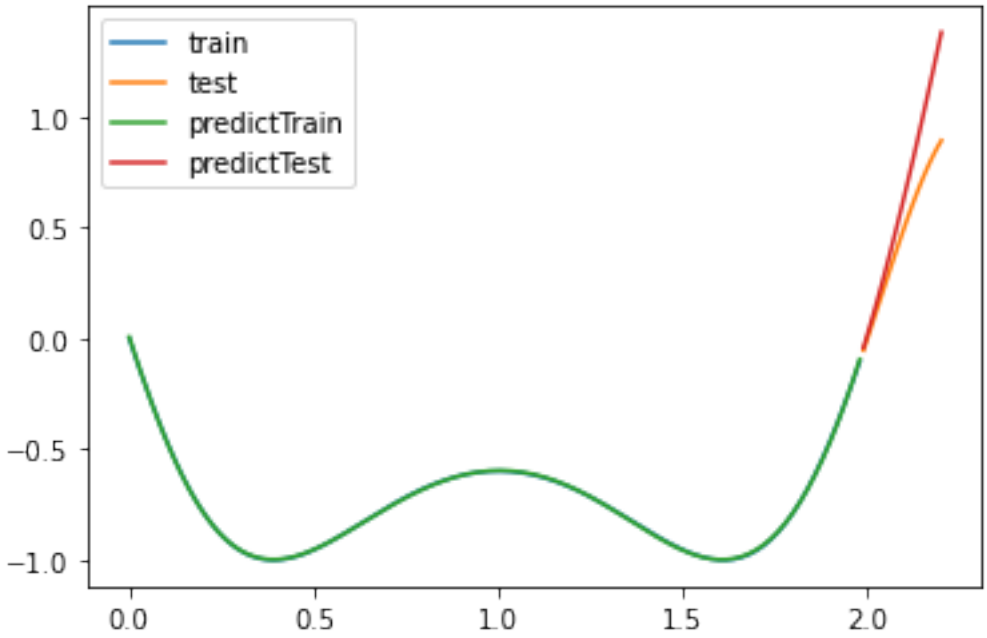
[ ]:
```
from sklearn.neural_network import MLPRegressor

model = MLPRegressor(hidden_layer_sizes=(30,), max_iter = 1000, tol=0.00000001,␣
 ↪activation='tanh', early_stopping=True, solver='lbfgs')
X_train_sk = [[i] for i in X_train]
y_train_sk = [[i] for i in y_train]

model.fit(np.array(X_train_sk), y_train)

# print(X_train)
```

[ ]:
```
MLPRegressor(activation='tanh', early_stopping=True, hidden_layer_sizes=(30,),
             max_iter=1000, solver='lbfgs', tol=1e-08)
```

[ ]:
```
plt.plot(X_train, y_train, label='train')
plt.plot(X_test, y_test, label='test')


pred_x = model.predict(X_train_sk[:])
mse = mean_squared_error(y_train, pred_x.flatten())
plt.plot(X_train, pred_x, label='predictTrain')
print(f'RMSE on train = {np.sqrt(mse)} ')

X_test_sk = [[i] for i in X_test]
```

```
pred_x = model.predict(X_test_sk[:])
mse = mean_squared_error(y_test, pred_x.flatten())
plt.plot(X_test, pred_x, label='predictTest')
print(f'RMSE on test = {np.sqrt(mse)}')

plt.legend()
plt.show()
```

```
RMSE on train = 0.00227225719563784
RMSE on test = 0.2255345500572716
```



2

: ,

, “ ” .

MINST (28*28).