ENPM 808A Final project

Soumosir Dutta
M.Eng Software

UID: 117380040

Code And Steps

Code :-

Data preprocessing - Done in local using jupyter
Total 107 csv training data
Load csv one by one

Process the field
Store in a numpy array

Append to the numpy array from 2nd iteration to 107th.
Headers used

```
In [3]:
import numpy as np
import pandas as pd
import seaborn as sns
import os
import pickle
import json
from glob import glob
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
```

For training data

```
In [4]:
PATH = "/Users/sdutta7/Documents/mlfinalproject/training_data"
EXT = "*.csv"
all_csv_files = []
for path, subdir, files in os.walk(PATH):
    for file in glob(os.path.join(path, EXT)):
        all_csv_files.append(file)
print(len(all_csv_files))

107
```

For testing data

```
In [11]:
PATH = "/Users/sdutta7/Documents/mlfinalproject/testing_data"
EXT = "*.csv"
all_csv_files = []
for path, subdir, files in os.walk(PATH):
    for file in glob(os.path.join(path, EXT)):
        all_csv_files.append(file)
print(len(all_csv_files))

16
```

Same process followed for testing and training data.

Training Data loaded in (X_nn.npy,y_nn.npy)
Testing data stored in (X_nn_test.npy,y_nn_test.npy)

```
In [10]:
        headers = []
        with open('config.json') as f:
            headers = json.load(f)
        i=0
        # df = pd.DataFrame(columns = headers)
        fX = []
        fy = []
        for path in all_csv_files:
            i+=1
            print("file number- ",i)
            df = pd.read_csv(path,names=headers)
        #     df = df.append(df_new, ignore_index = True)
            df['laser_1_180'] = df[['Laser'+str(i+1) for i in range(0,180)]].mean(axis=1)
            df['laser_181_360'] = df[['Laser'+str(i+1) for i in range(180,360)]].mean(axis=1)
            df['laser_361_540'] = df[['Laser'+str(i+1) for i in range(360,540)]].mean(axis=1)
            df['laser_541_720'] = df[['Laser'+str(i+1) for i in range(540,720)]].mean(axis=1)
            df['laser_721_900'] = df[['Laser'+str(i+1) for i in range(720,900)]].mean(axis=1)
            df['laser_901_1080'] = df[['Laser'+str(i+1) for i in range(900,1080)]].mean(axis=1)

            df = df.iloc[: , 1080:]
            X = df[["Final_goal_x","Final_goal_y","Final_goal_qk","Final_goal_qr",
                    "Local_goal_x","Local_goal_y","Local_goal_qk","Local_goal_qr",
                    "Robot_pos_x","Robot_pos_y","Robot_pos_qr","Robot_pos_qk",
                    "laser_1_180","laser_181_360",
                    "laser_361_540","laser_541_720",
                    "laser_721_900","laser_901_1080"]]

            y = df[["Cmd_vel_v","Cmd_vel_w"]]
            X = np.array(X)
            y = np.array(y)
        #   # due to large number of data load a Dataframe from csv process it and stor ein numpy array
            if i>1:
                X_prev = np.load('X_nn.npy')
                y_prev = np.load('y_nn.npy')
                X = np.vstack((X_prev, X))
                y = np.vstack((y_prev, y))
                print("inside concatenate - ", X.shape,y.shape)
                np.save('X_nn.npy', X)
                np.save('y_nn.npy', y)
            else:
                np.save('X_nn.npy', X)
                np.save('y_nn.npy', y)
```

Now train data was divided in 0.2 ratio to validation and Training Set from numpy Array
Done in google collab

```
[2] # %tensorflow_version 2.x  # this line is not required unless you are in a notebook
    # # TensorFlow and tf.keras
    import numpy as np
    import pandas as pd
    import seaborn as sns
    import os
    import pickle
    import json
    from glob import glob
    from sklearn.multioutput import RegressorChain
    from sklearn.svm import LinearSVR
    from sklearn.neighbors import KNeighborsRegressor
    from sklearn.tree import DecisionTreeRegressor
    from sklearn.metrics import mean_squared_error
    from sklearn.linear_model import LinearRegression
```

```
[6] X = np.load('X_nn.npy')
    y = np.load('y_nn.npy')
    from sklearn.model_selection import train_test_split
    X, XValidation, y, yValidation = train_test_split(X,y,test_size=0.2,random_state=42)
    print(X.shape,y.shape,XValidation.shape,yValidation.shape)
```

```
(2195595, 18) (2195595, 2) (548899, 18) (548899, 2)
```

```
X_test = np.load('X_nn_test.npy')
y_test = np.load('y_nn_test.npy')
print(X_test.shape,y_test.shape)
```

```
(389817, 18) (389817, 2)
```

Linear Regression
MSE as error

```
model_linear = LinearRegression()

model_linear.fit(X, y)
# pickle.dump(model_linear, open('model_linear.sav', 'wb'))

y_pred = model_linear.predict(X)
# print(y.shape,y_pred.shape,X.shape)
mse_l = mean_squared_error(y, y_pred)
print(" mse in training data -LR: ",mse_l)

y_pred_test = model_linear.predict(X_test)
mse_l_t = mean_squared_error(y_test, y_pred_test)
print(" mse in testing date -LR : ",mse_l_t)
```

```
 mse in training data -LR:  0.08469151131341271
 mse in testing date -LR :  0.08275897426599729
```

Ridge Regressor (Linear least squares with l2 regularization.)

```
from sklearn.multioutput import MultiOutputRegressor
from sklearn.linear_model import Ridge

model_Ridge = MultiOutputRegressor(Ridge(random_state=123)).fit(X, y)
y_pred_ridge = model_Ridge.predict(X)
# print(y.shape,y_pred.shape,X.shape)
mse_lr = mean_squared_error(y, y_pred_ridge)
print(" mse in training data -Ridge: ",mse_lr)

y_pred_test_ridge = model_Ridge.predict(X_test)
mse_lr_t = mean_squared_error(y_test, y_pred_test_ridge)
print(" mse in testing date -Ridge : ",mse_lr_t)
```

```
 mse in training data -Ridge:  0.08469151131345751
 mse in testing date -Ridge :  0.08275897544343264
```

Linear model fitted by minimizing a regularized empirical loss with SGD.

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

```
from sklearn.linear_model import SGDRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
reg = MultiOutputRegressor(make_pipeline(StandardScaler(),SGDRegressor(max_iter=1000, tol=1e-3)))
reg.fit(X, y)

y_pred_reg = reg.predict(X)
# print(y.shape,y_pred.shape,X.shape)
mse_reg = mean_squared_error(y, y_pred_reg)
print(" mse in training data -SDGRegressor: ",mse_reg)

y_pred_test_reg = reg.predict(X_test)
mse_reg_t = mean_squared_error(y_test, y_pred_test_reg)
print(" mse in testing date -SDGRegressor : ",mse_reg_t)
```

```
 mse in training data -SDGRegressor:  0.08484172256219111
 mse in testing date -SDGRegressor :  0.08305160480836499
```

Linear Model trained with L1 prior as regularizer (aka the Lasso).

```
from sklearn import linear_model
model_lasso = linear_model.Lasso(alpha=0.1)
model_lasso.fit(X,y)

y_pred_model_lasso = model_lasso.predict(X)
# print(y.shape,y_pred.shape,X.shape)
mse_model_lasso = mean_squared_error(y, y_pred_model_lasso)
print(" mse in training data -model_lasso: ",mse_model_lasso)

y_pred_test_model_lasso = reg.predict(X_test)
mse_model_lasso_t = mean_squared_error(y_test, y_pred_test_model_lasso)
print(" mse in testing date -model_lasso : ",mse_model_lasso_t)
```

```
 mse in training data -model_lasso:  0.08971326567345395
 mse in testing date -model_lasso :  0.08305160480836499
```

Neural Network model After Validation and Regularization

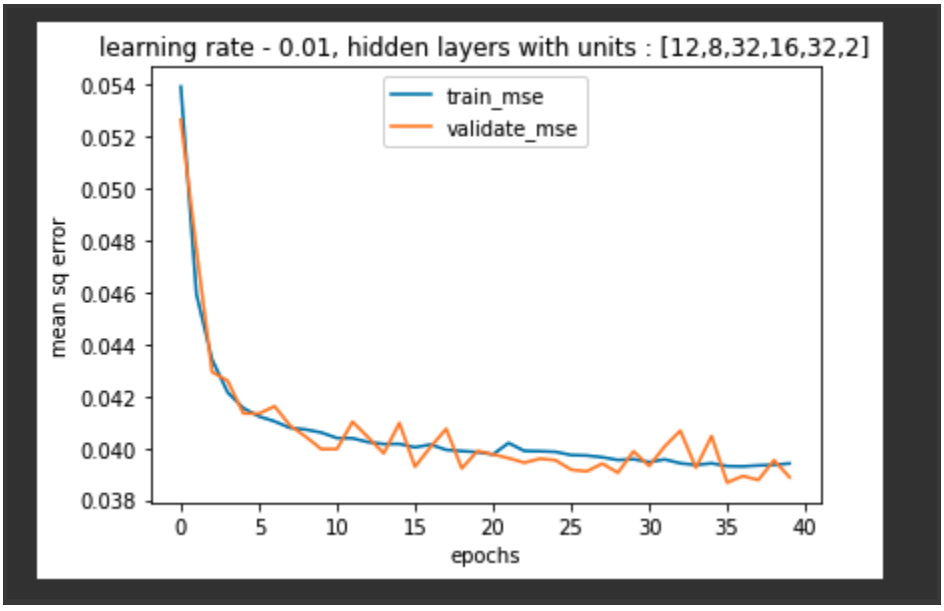
```
from tensorflow.python.keras.layers.core import Dropout
model = keras.Sequential()
model.add(layers.Dense(12, input_dim=18, activation='relu'))
model.add(layers.Dense(8, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
# model.add(Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
# ,kernel_regularizer=keras.regularizers.l2(0.01)
model.add(layers.Dense(2, activation='linear'))
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss='mse',
    metrics=['mse','accuracy'])
model_copy = model
```

```
[ ] history_v1 = model.fit(X, y, epochs=40,batch_size=1000,validation_data=(XValidation,yValidation))

poch 1/40
196/2196 [==============================] - 9s 4ms/step - loss: 0.0539 - mse: 0.0539 - accuracy: 0.8008 - val_loss: 0.0526 - val_mse:
poch 2/40
196/2196 [==============================] - 9s 4ms/step - loss: 0.0459 - mse: 0.0459 - accuracy: 0.7577 - val_loss: 0.0476 - val_mse:
poch 3/40
196/2196 [==============================] - 9s 4ms/step - loss: 0.0434 - mse: 0.0434 - accuracy: 0.7574 - val_loss: 0.0429 - val_mse:
poch 4/40
196/2196 [==============================] - 8s 4ms/step - loss: 0.0421 - mse: 0.0421 - accuracy: 0.7552 - val_loss: 0.0426 - val_mse:
poch 5/40
```

Learning rate - 0.01

loss: 0.0394 - mse: 0.0394 - accuracy: 0.7967 - val_loss: 0.0389 - val_mse: 0.0389 - val_accuracy: 0.9234
[12,8,32,16,32,2]'




```
history_v1.model.evaluate(X_test,y_test)

12182/12182 [==============================] - 18s 2ms/step - loss: 0.0397 - mse: 0.0397 - accuracy: 0.9228
[0.039745766669511795, 0.039745766669511795, 0.9228150844573975]
```

```
12182/12182 [==============================] - 18s 2ms/step - loss: 0.0397 - mse:
0.0397 - accuracy: 0.9228
[0.039745766669511795, 0.039745766669511795, 0.9228150844573975]
```