ENPM 808A: INTRODUCTION TO MACHINE LEARNING

FINAL PROJECT

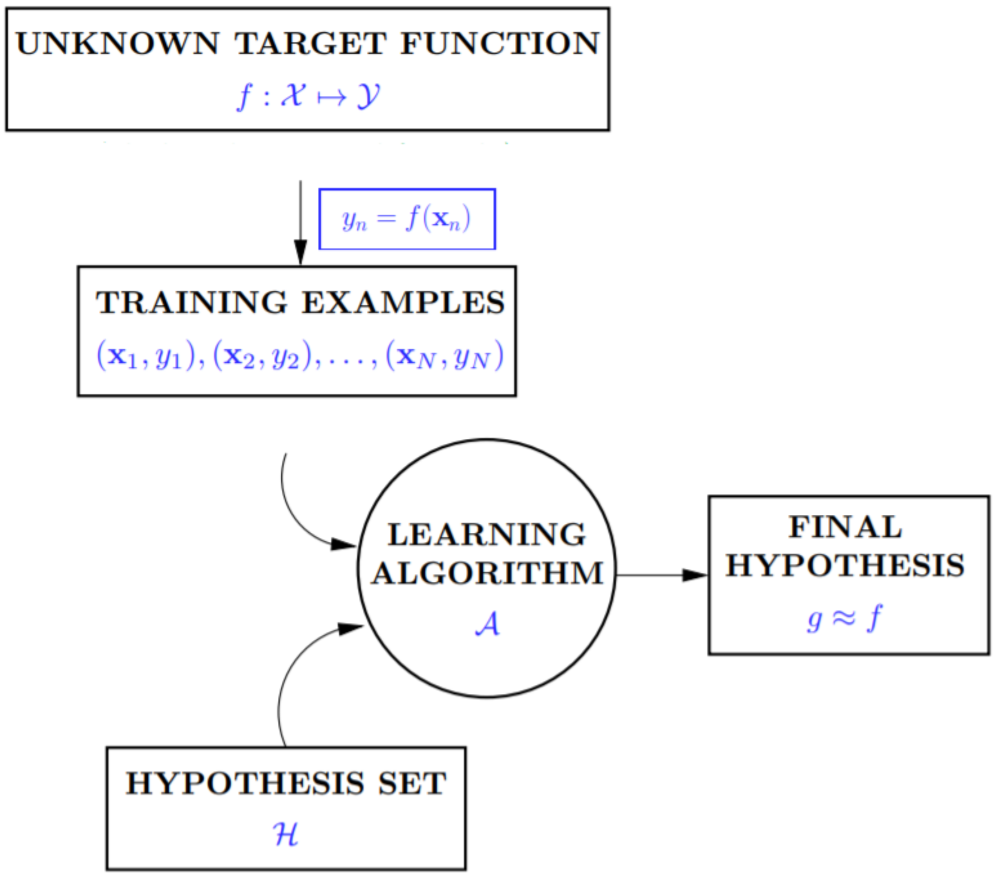
SUBMITTED BY

SHARMITHA GANESAN
(UID – 117518931)

TABLE OF CONTENTS

1. INTRODUCTION:

Machine Learning is used when a problem has no easy analytical framework to work with and when a lot of data is available. Model based learning types use data to train a particular model and then utilizing that model for prediction in the future. This learning type accounts for less computation. Generally, after choosing a learning algorithm, the training data and the functions in the hypothesis set gives a best hypothesis function matching the target function.

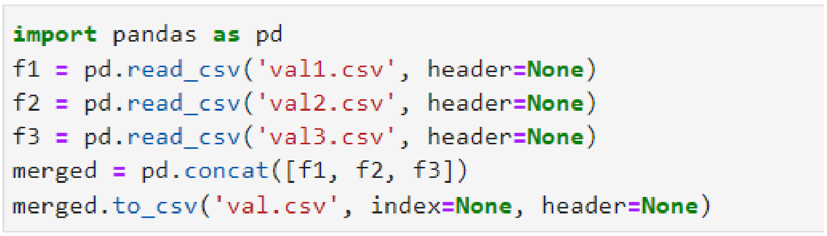


2.PROBLEM STATEMENT:

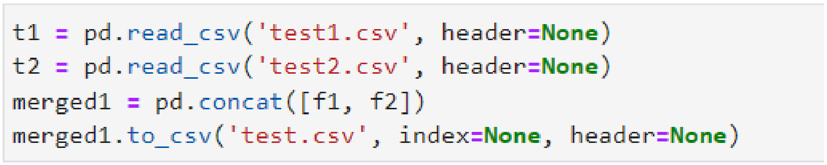
A car like robot is given where v and w are the translational velocity and rotational velocity of the robot. The robot is operated in various environments like corridor and open hall and the LIDAR outputs of the surroundings are collected. Besides the laser range data, the positional data of the data is also collected. The action commands for the robot to navigate are provided in the form of translational velocity (v) and rotational velocity (w). All the above-mentioned data are collected for months and through an efficient machine learning algorithm, the action commands for the robot to move around should be predicted.

3.DATA PRE-PROCESSING:

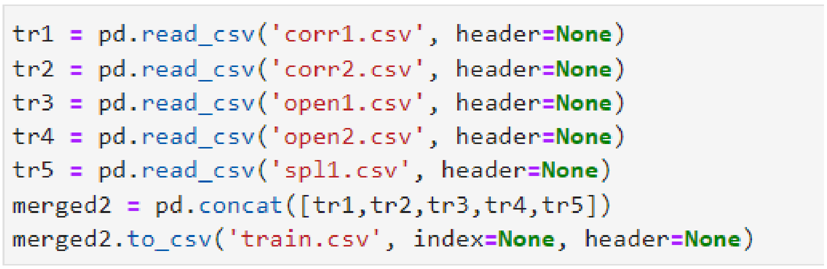
The laser range data from the LIDAR is a 1D array of 1080 values which is the range recorded at each 0.25deg within a 270deg field of view with the 540th element being the range corresponding to directly in front of the robot. The robot current position, the local and final goal information are given in 3D cartesian coordinates (x; y; z) for position and a quaternion (q = qr + qi*i + qj*j + qk*k) for the orientation. The current work only considers the horizontal plane i.e, x and y and qr and qk in quaternion.



```python
import pandas as pd
f1 = pd.read_csv('val1.csv', header=None)
f2 = pd.read_csv('val2.csv', header=None)
f3 = pd.read_csv('val3.csv', header=None)
merged = pd.concat([f1, f2, f3])
merged.to_csv('val.csv', index=None, header=None)
```



```python
t1 = pd.read_csv('test1.csv', header=None)
t2 = pd.read_csv('test2.csv', header=None)
merged1 = pd.concat([f1, f2])
merged1.to_csv('test.csv', index=None, header=None)
```

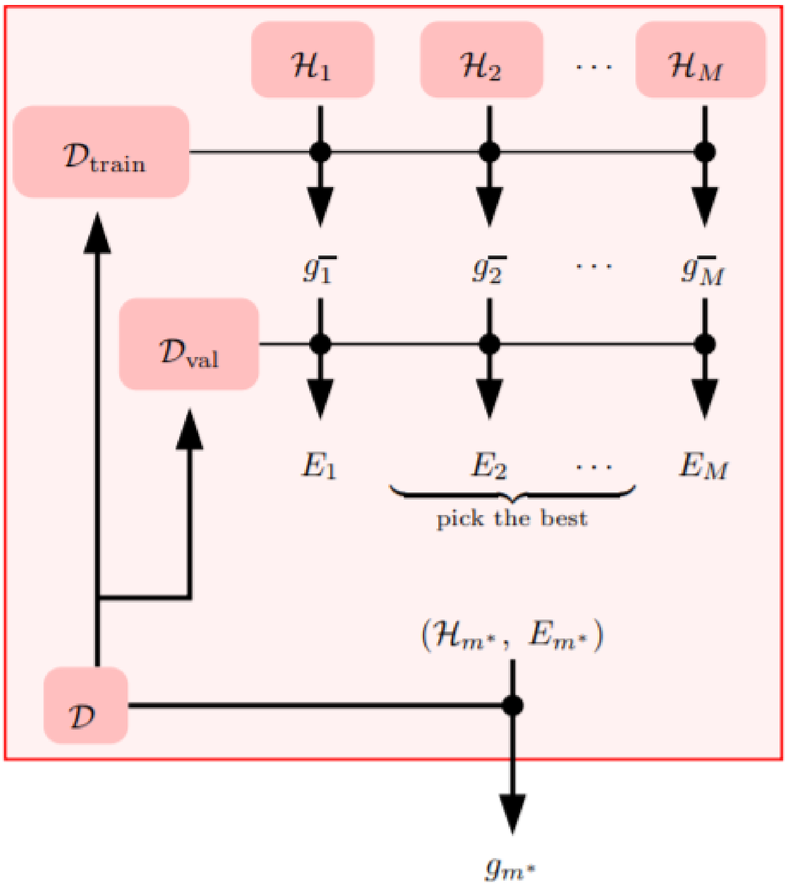


```python
tr1 = pd.read_csv('corr1.csv', header=None)
tr2 = pd.read_csv('corr2.csv', header=None)
tr3 = pd.read_csv('open1.csv', header=None)
tr4 = pd.read_csv('open2.csv', header=None)
tr5 = pd.read_csv('spl1.csv', header=None)
merged2 = pd.concat([tr1,tr2,tr3,tr4,tr5])
merged2.to_csv('train.csv', index=None, header=None)
```

For the data processing, a field of view of 30 degrees is taken as one value and hence the laser data is brought down to 9 columns. In the positional data, the quaternion sum q is taken adding qr and qk. Hence, the current robot position, the local goal and the final goal positional data has x, y and q values contributing 3 columns each. The input features have become into 18 columns. A testing dataset, a training dataset and a validation dataset are segregated from the given data set for further learning.

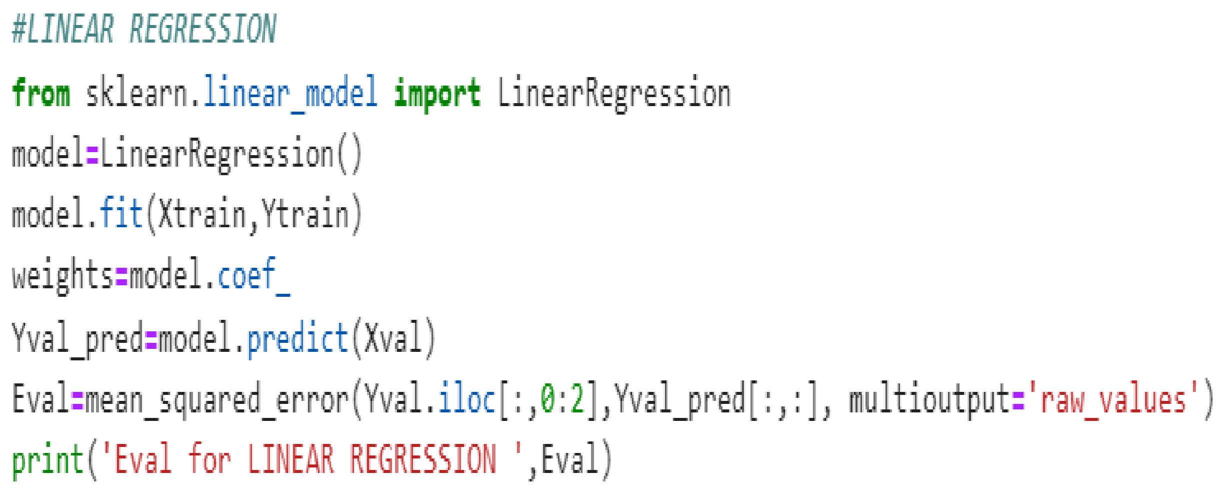
4.MODEL SELECTION AND VALIDATION:

After the data processing is done, the merged versions of the validation data, the training data and the testing data are used for fitting, prediction, and testing. For this project, LINEAR REGRESSION FROM SKLEARN and NEURAL NETWORKS FROM KERAS TENSORFLOW are taken and compared. Both the models are trained with training data and fitted with validation data. Then, the validation error E_val is calculated for both models. In general, validation is used to as an estimate of E_out and is very helpful in giving an idea of the efficiency of the model in the client point of view. As a leap of faith, the model with better E_val is considered to be the best model to implement and test out with the testing data. This is how the model selection is done in this project.

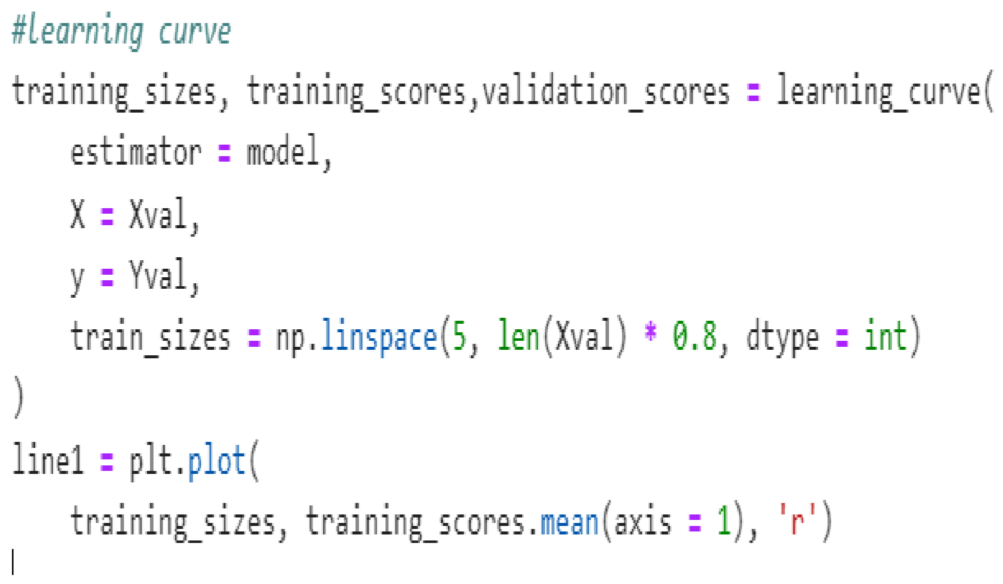


5. SELECTION OF THE PREDICTION MODEL:

Firstly, the linear regression model is taken from sklearn.

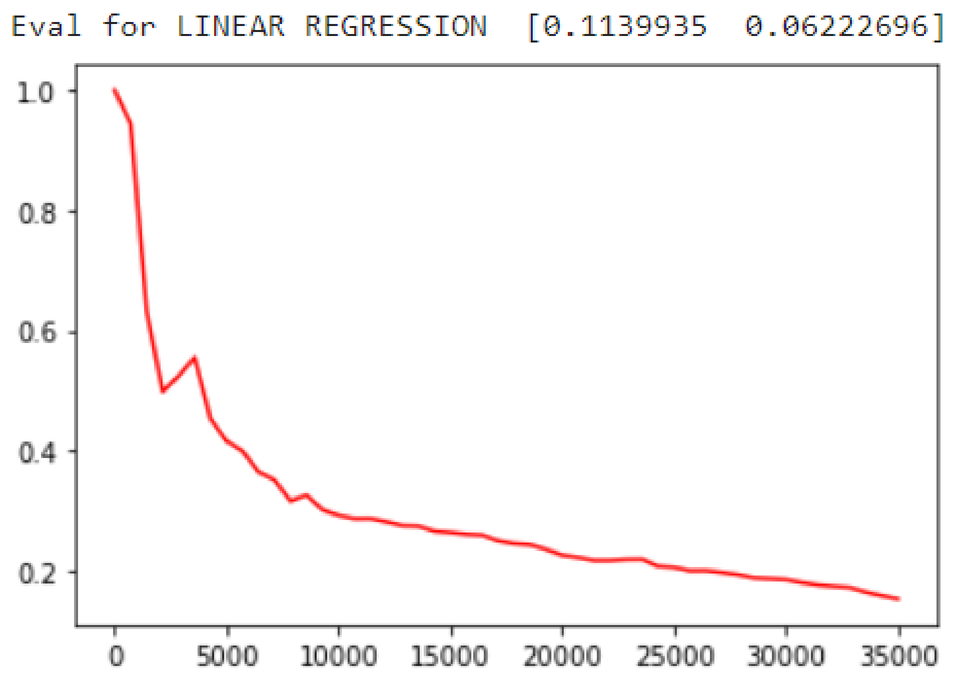


```
#LINEAR REGRESSION
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(Xtrain,Ytrain)
weights=model.coef_
Yval_pred=model.predict(Xval)
Eval=mean_squared_error(Yval.iloc[:,0:2],Yval_pred[:,:], multioutput='raw_values')
print('Eval for LINEAR REGRESSION ',Eval)
```



```
#learning curve
training_sizes, training_scores,validation_scores = learning_curve(
    estimator = model,
    X = Xval,
    y = Yval,
    train_sizes = np.linspace(5, len(Xval) * 0.8, dtype = int)
)
line1 = plt.plot(
    training_sizes, training_scores.mean(axis = 1), 'r')
```

The Eval is calculated between Yval data and Ypredicted data. The learning curve is also plotted.



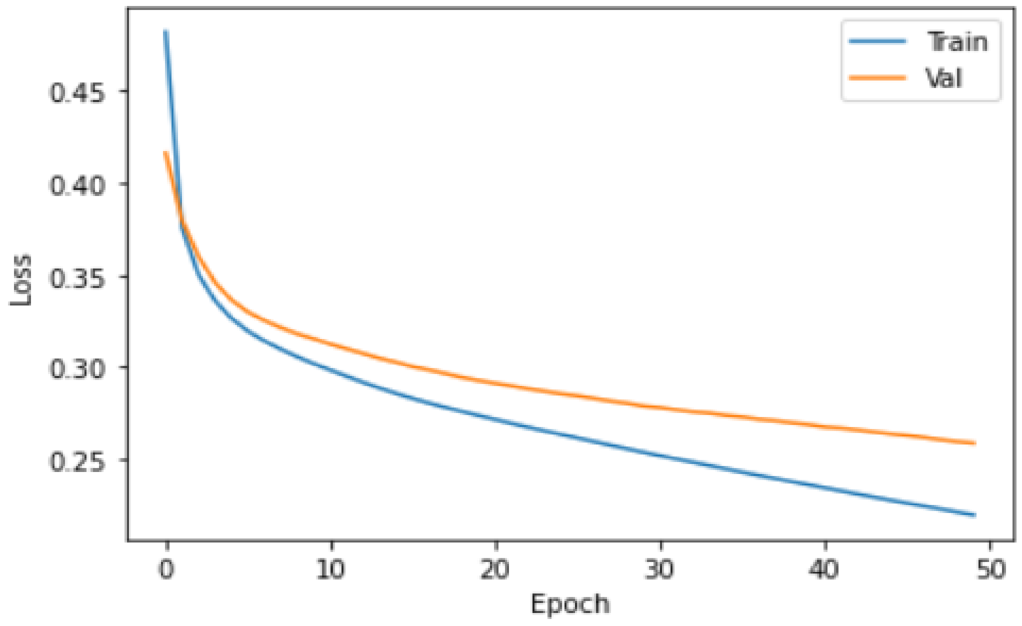
```
Eval for LINEAR REGRESSION  [0.1139935  0.06222696]
```

Then, the neural network model is taken from keras TensorFlow. The learning curve is plotted and the Eval is also evaluated for both translational and rotational velocity. The keras sequential model i.e., a model defined layer by layer, has which is the hyper first and second layer dense as 18 in this project. After several trial and error the below mentioned hyper-parameters are used in the neural network. One of the activation types used here is tanh which is the hyperbolic function that takes any real value as input and gives output in the range of -1 to 1.

The sigmoid activation type gives out output ranging from 0 to 1. This activation is given to the output dense layer of the neural network. After this step, the configuration of the model should be done with the function model.compile. Here, the stochastic gradient descent optimizer is chosen. The loss function for outputs take values 1 0r 0 in binary cross entropy.

```
import tensorflow
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense

neural = Sequential([
    Dense(18, activation='tanh'),
    Dense(18, activation='tanh'),
    Dense(2, activation='sigmoid')
])
neural.compile(optimizer='sgd',
              loss='binary_crossentropy')

#for translational velocity

hist = neural.fit(Xtrain, Ytrain, batch_size=1000, epochs=50,validation_data=(Xval, Yval))

Yval_pred1=neural.predict(Xval)
Eval1=mean_squared_error(Yval.iloc[:,0:1],Yval_pred1[:,0], multioutput='raw_values')
print('Eval of translational velocity for Neural Network',Eval1)
Eval2=mean_squared_error(Yval.iloc[:,1:2],Yval_pred1[:,1], multioutput='raw_values')
print('Eval of rotational velocity for Neural Network',Eval2)

#learning curve
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')

plt.show()
```

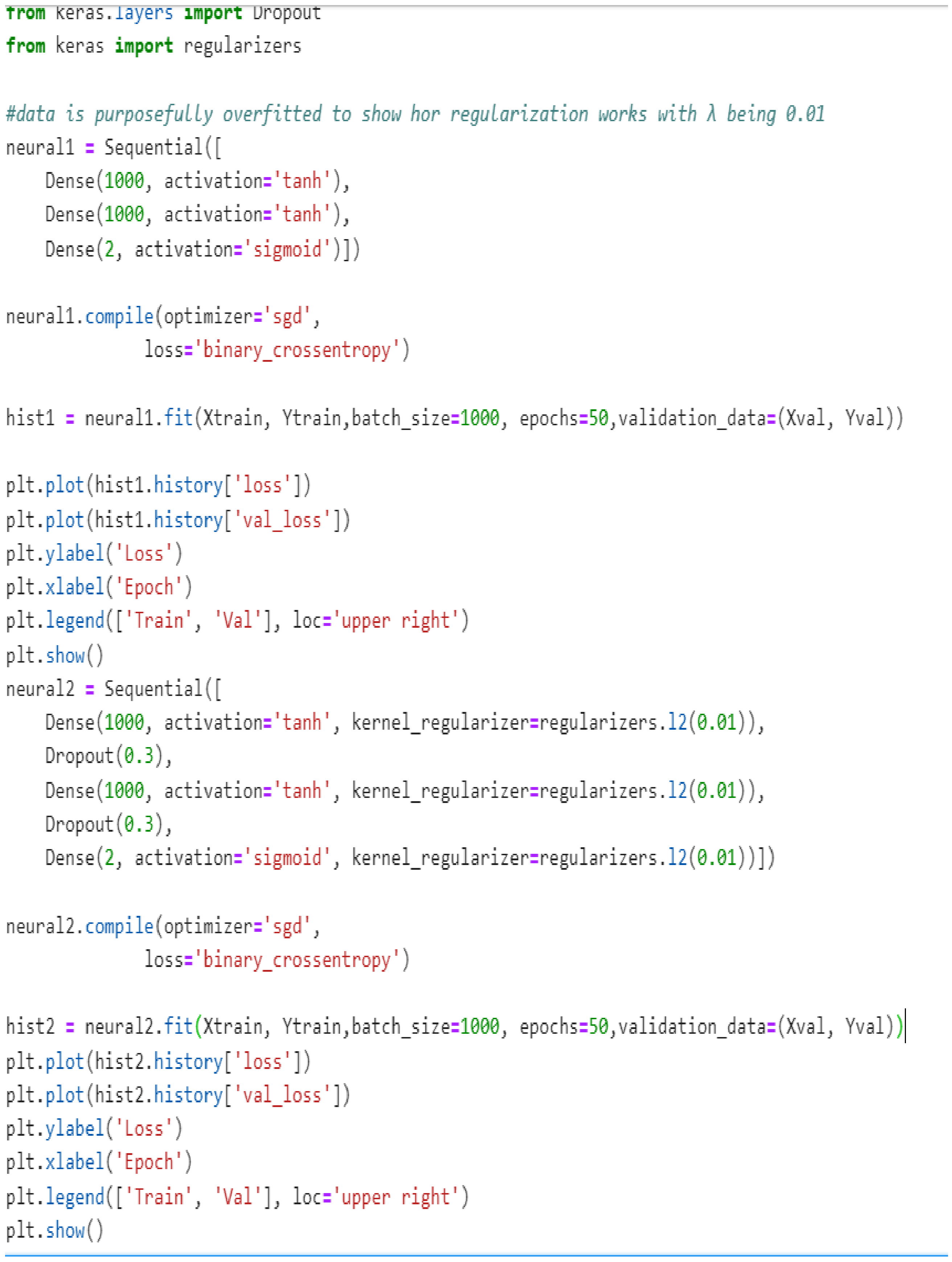
```
Eval of translational velocity for Neural Network [0.10823153]
Eval of rotational velocity for Neural Network [0.0638172]
```



On comparing the Eval values of both the learning models, the NEURAL NETWORK model is chosen for further analysis.

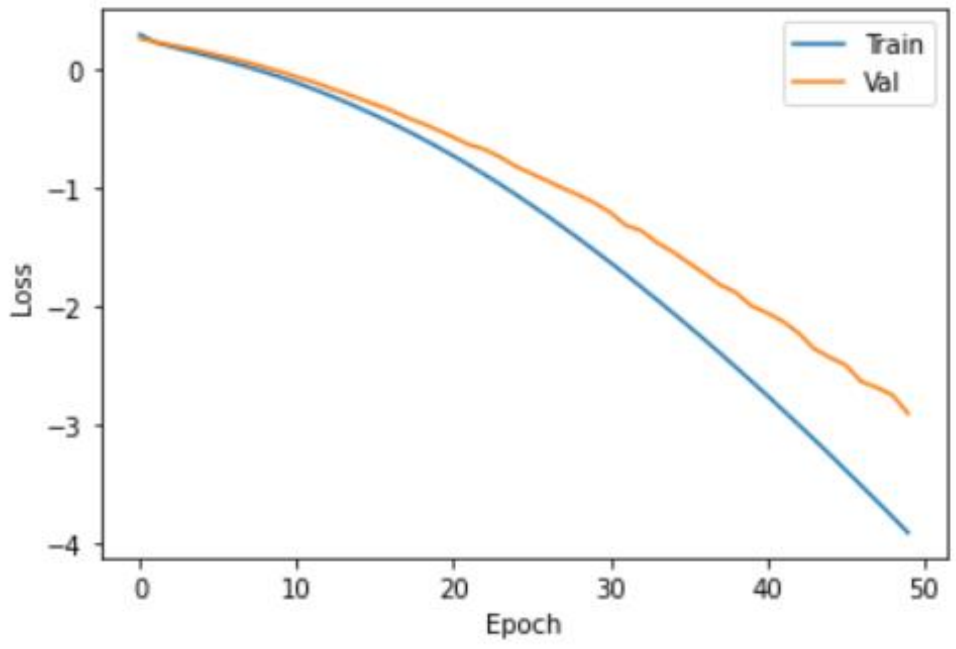
5.1 <u>REGULARIZATION :</u>

Regularization is done to prevent overfitting of the training dataset. Here the dense values for layer one and two are taken as 1000. This leads to the learning and validation curve take negative values. To prevent this, the regularization parameter is taken as 0.01. The l2 regularization tells Keras to include the squared values of those parameters in our overall loss function, and weight them by 0.01 in the loss function. The change in the output plot after regularization can be observed below.

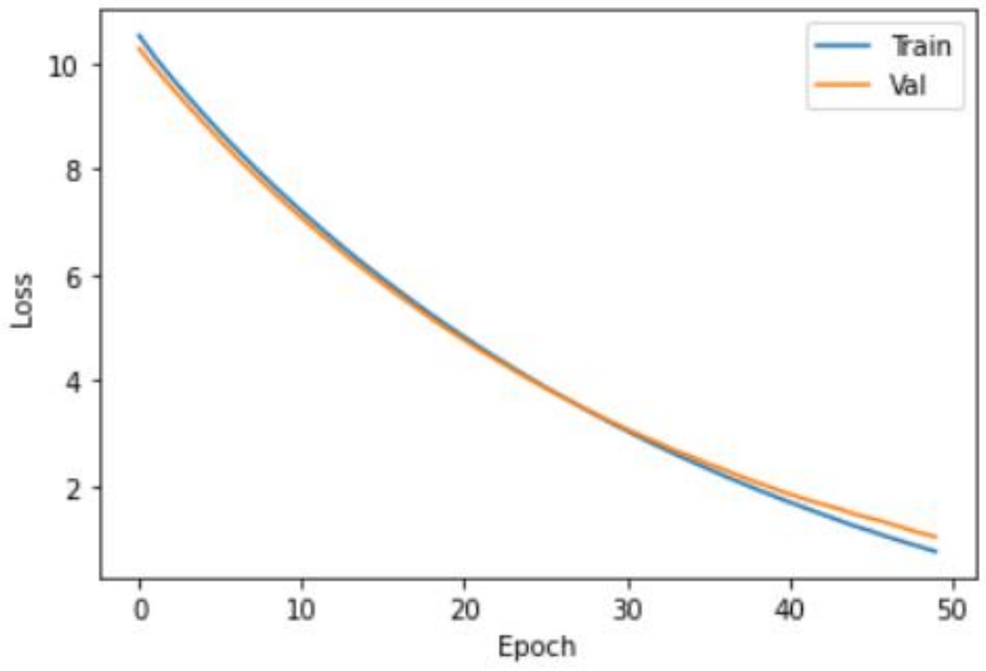


```python
from keras.layers import Dropout
from keras import regularizers

#data is purposefully overfitted to show hor regularization works with λ being 0.01
neural1 = Sequential([
    Dense(1000, activation='tanh'),
    Dense(1000, activation='tanh'),
    Dense(2, activation='sigmoid')])

neural1.compile(optimizer='sgd',
              loss='binary_crossentropy')

hist1 = neural1.fit(Xtrain, Ytrain,batch_size=1000, epochs=50,validation_data=(Xval, Yval))

plt.plot(hist1.history['loss'])
plt.plot(hist1.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
neural2 = Sequential([
    Dense(1000, activation='tanh', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(1000, activation='tanh', kernel_regularizer=regularizers.l2(0.01)),
    Dropout(0.3),
    Dense(2, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01))])

neural2.compile(optimizer='sgd',
              loss='binary_crossentropy')

hist2 = neural2.fit(Xtrain, Ytrain,batch_size=1000, epochs=50,validation_data=(Xval, Yval))
plt.plot(hist2.history['loss'])
plt.plot(hist2.history['val_loss'])
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Val'], loc='upper right')
plt.show()
```

The output of overfitted data with poor learning curve and validation curve.



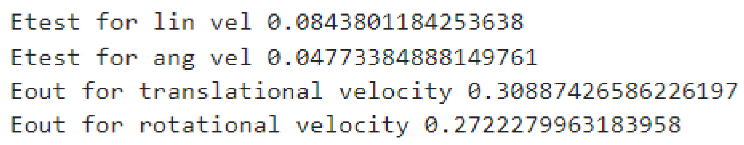
The output after regularization.



6. OUT-OF-SAMPLE ERROR FROM TEST DATA:

The selected prediction model, neural networks learning algorithm is used to test the data and evaluate the out of sample error.

```
#to_test
Xtest=test1.iloc[:,0:18]
Ytest=test1.iloc[:,18:20]
Ytest_pred=neural.predict(Xtest)

Etest_linvel=mean_squared_error(Ytest.iloc[:,0:1],Ytest_pred[:,0])
print('Etest for lin vel',Etest_linvel)
Etest_angvel=mean_squared_error(Ytest.iloc[:,1:2],Ytest_pred[:,1])
print('Etest for ang vel',Etest_angvel)
dvc=19
tol=0.1

E_out = Etest_linvel + np.sqrt((8/35861)*np.log((2*35861)**(dvc+1)/tol))
print('Eout for translational velocity',E_out)
E_out1 = Etest_angvel + np.sqrt((8/35861)*np.log((2*35861)**(dvc+1)/tol))
print('Eout for rotational velocity',E_out1)
```



```
Etest for lin vel 0.0843801184253638
Etest for ang vel 0.04773384888149761
Eout for translational velocity 0.30887426586226197
Eout for rotational velocity 0.2722279963183958
```

7. CONCLUSION:

Both the learning models are analysed and the best model for the given problem statement has been chosen as NEURAL NETWORK learning algorithm.