ENPM808A - Introduction to Machine Learning

FINAL PROJECT

Sahruday Patti | 118218366 | 08/16/2021

Introduction

Robot navigation means the robot's ability to determine its own position in its frame of reference and then to plan a path towards some goal location. Path planning is one of the most prominent and essential parts of autonomous mobile robot navigation. Path planning involves the creation of a map of the environment by analyzing various types of sensor data, which involves a lot of computation processing. To reduce these computations, this project presents a data-driven motion planner. The ground robot in the project is provided demonstrations on how to navigate in a Corridor & a Box environment with obstacles produced by a path planning algorithm called A*. During navigation, the goal is not only to replicate the provided expert demonstrations in one specific scenario, but rather be able to learn collision avoidance strategies and replicate the results in previously unseen environments.

The data-Driven approach does not require a global map to navigate. Given the sensor data and the relative final and local goal positions, the robot will be able to navigate to the local goal keeping in mind the final goal while also avoiding obstacles.

Problem Statement

The goal is to develop a model, which provides action commands in the form of a translational velocity command (v) and a rotational velocity command (w), for a certain laser data having moving obstacles given the model is trained on two environments (A corridor scenario with moving obstacles & an open box/hall environment with moving obstacles).

The tasks involved in solving the problem statement are:

- Download and preprocess the data
- Development of a loss function to optimize i.e., the error measure
- Model Selection by use of Cross-Validation which optimizes the loss function
- Testing the model on test data and get an estimate of out of sample error

Data Preprocessing

1) The data is organized as follows:
    - The first 1080 columns represent the laser range data.
    - Next 4 columns represent final goal information.
    - Next 4 columns represent the local goal information.
    - Next 4 columns represent the robot's current position and pose information.

    - Final 2 columns represent the commanded action.

2) Feature Extraction:
    - Each Column of the laser data consists of 0.25-degree information. So, to reduce the number of dimensions laser data for the 15-degree range is considered. This is extracted by calculating the mean of 30columns. This reduced the number of dimensions of laser data from 1080 to 18.
    - The relative distance between the local, final goal positions and orientations with respect to robots local and orientation is considered. This reduces the goal and orientation information from 12 columns to 8.
    - To calculate the relative Orientation, calculating the cos angle between the orientation vectors is considered. But the output has been satisfactory, hence didn't use this feature.
    - By doing this, the effective number of dimensions has been reduced to 28.

3) Data Scaling:
    - The given data has already been scaled. But tried L1 Normalizing and Standard scalar functions also and used on the model just to see how the model predicts.
    - By doing this helped in understanding the differences between normalizing and scaling.
4) Data Separation:
    - The data has been separated for Training and Validation with a split of 0.2
5) The above model has used only a fraction of the data set that has been provided due to the memory constraints.

Loss Function

Our model aims to predict suitable steering commands **U (v, w)**, given the sensor data, goal & orientation information. So, we try to find an equation,

$$U = F_\theta(y, g)$$

Which directly maps a vector of sensor data y and goal information g to the desired steering commands U. This function is parameterized by a vector θ. During supervised training, we find the function parameters θ that best explains a set of training data. The optimization criterion is based on $|F_\theta(y, g) - u|$, the difference between the predicted steering commands and the ones provided by the expert operator. This is the mean absolute error function.

Model Selection

The type of problem here is Regression, so used regression models to evaluate:

- Linear Regression: Trained the linear regression and calculated the in-sample error & validation error, which is the mean absolute error.
- Decision Tree Regressor: Trained the Decision Tree Regressor and in-sample error & validation error has been calculated & saved.
- Random Forest Regressor: Used a grid search with 10-fold cross-validation to find the model with a low mean validation error and saved the number of estimators for the best model and mean validation error.
- Later used a Neural Network and the following Hyperparameters have been tuned to optimize the classifier:
    - ♦ Training Length (number of epochs)
    - ♦ Batch size
    - ♦ Number of Layers
    - ♦ Learning rate
    - ♦ Regularization
    - ♦ Activation Layer Types
    - ♦ Optimizer
    - ♦ Weights Initialization

Initially, a single hidden layer of neural network with l1 regularization of 0.1 and an output layer of 2 nodes was built and trained for a certain number of epochs and plotted the learning and validation loss curves. The validation loss was much higher than the learning loss. So, presumed the model is underfitting. By interpreting the learning curve, the curve becomes a straight line too quickly.

Later increased the number of layers and nodes with different optimizers namely SGD & Adam, different activation functions like relu and leaky relu, different weight initializers like Xavier, and different regularization coefficients and plotted the curves.

By Observing the learning curve, realized models were having a constant learning curve after very few epochs, and the validation curve was plotted against it, the validation curve loss is still above the training less. Thought, it was a local minimum, wrote an exponential decay function with a high learning rate for SGD. But curves remained the same.

Later, used a Huber Loss function to optimize without good results. So, learning from the curves that the model is underfitting increased the number of layers and nodes of each layer, and decreased the regularization parameter to 0.001, the validation curve was a lot closer to the training loss but was having the same problem that after a very few epochs both the curves are just a straight line, and the validation loss is higher. This is also a case for underfitting.

Finally, to try and fit the data, the regularization term was reduced to 0.0001 and lower and this showed results in the curves. The validation and training loss curves are closer. Essentially realizing the coefficient of regularization was very low, So, dropped regularization from the layers to help the model fit better.

After Dropping regularization, the model was fitting better, and the curves are much closer & the training loss decreased after each epoch and the validation loss decreased with it. To give a better representation of the training data while training started tuning the batch size and random split values, by looking at the learning curve, which brought the curves closer and closer.

Finally, the model with 5 layers (including output layer), with optimizer Adam (which has Exponential decay learning rate) without the regularization with 100 epochs & a batch size of 128 tracked the validation loss smoothly and the validation loss was better than any other model till now.

Up until now, the values to be predicted were both velocity and rotational velocity commands together. So, tried to develop two individual neural networks to predict the values separately. Also tried Convolutional Neural Network first to classify the laser range data and then pass the output of CNN to a neural network to predict the commands. But couldn’t go past the errors of combining the output of one layer to another.

Validation Curve:

The validation curve was used to tune the Hyperparameters of the model. After each change in the hyperparameter, the validation curve & the learning curve was plotted and see how the model is fitting. The curve helped in understanding how the model is fitting the data and what effect the change in the hyperparameter had on the model which helped in making decisions to improve the model.

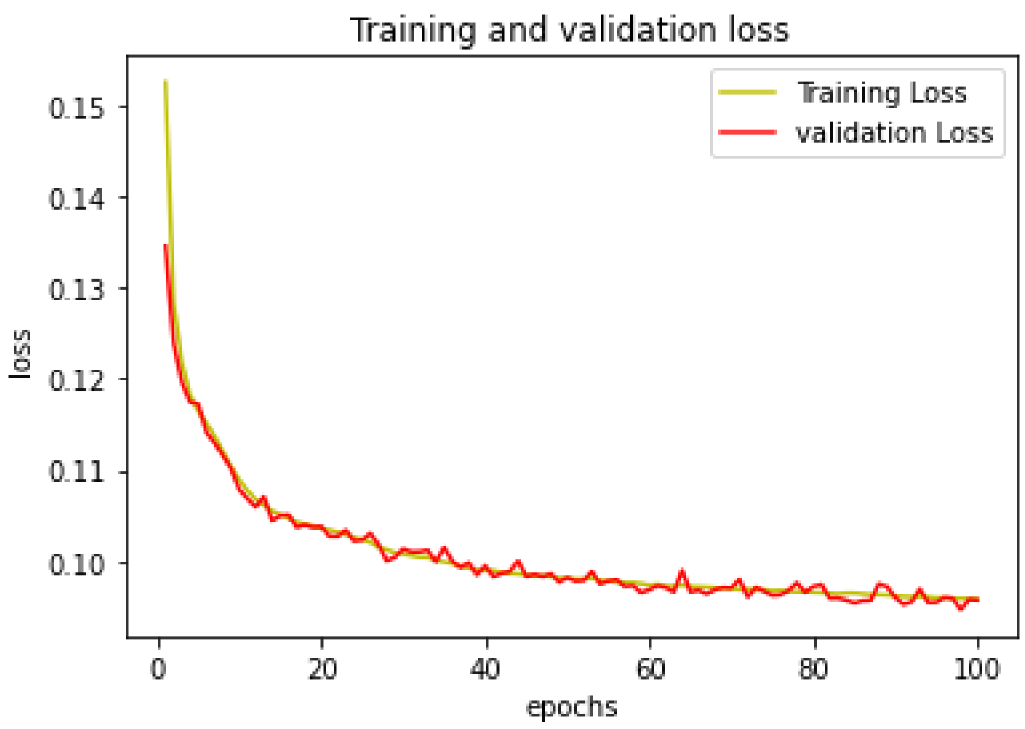
Regularization:

We are using a mean absolute error loss function to optimize so constraint on the weights based on the 'absolute value of magnitude seemed more logical. But the data doesn't have any inherent noise as seen in the model selection process. Hence, decided not to put any constraint on the weights.

But to be sure, tried l2 regularization too and saw the validation curve which showed similar results. Hence, did not use any regularization on the final model.

Final Model Evaluation

The final model is chosen from all the above models which had the least validation error that is a neural network with 5 layers. Below is the Training loss plotted against the validation error of the model. If we ran for 200 epochs the in-sample error & validation will continue to decrease but the in-sample R2 score is also decreasing, hence stopping at 100 epochs. Also, early stopping can be done at 250 epochs where the validation curve starts to diverge from the training loss.



After selecting this model from validation, the model was trained with the whole dataset i.e., including the validation dataset, and the final model was used for Test Estimate.

After, training the model on the whole data set, the in-sample error was found to be:

The In-sample Error of the above model is: 0.0957

Metrics of Test data:

The mean absolute error of the model on test data was: 0.0995

R2 Score: R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable or variables in a regression model.

The residual error, the difference between the original commands & the predicted commands squared is the residual error. The residual error needs to be low for our model to be able to generalize well in real scenarios.

The R2 score is calculated as 1 – RSS/TSS. Where RSS is the Residual sum square and TSS is the difference between the mean of the data points & the data points squared.

The R2 score for the final model is found to be: 0.4124

This is a very bad score. The R2 score should be close to one to be able to generalize well. Although the in-sample loss is a little low, the R2 score is very bad.

This model will not be able to make precise motor commands required for motion planning in real life scenario.

Estimate for the out of sample error

Evaluation on the test, simple Hoeffding inequality holds. The total number of testing examples given is N=389817. The effective number of hypotheses becomes 1 in the test set. As simple hoeffding inequality holds, there is no need of VC dimension of the model.

We know, $E_0 \leq E_I + \varepsilon$ where, E_0 = Eout and E_I = in sample error. In this case a test error.

And $\varepsilon = \sqrt{\frac{1}{2N} \ln \frac{2}{\delta}}$. For a $\delta \ = \ 0.05$ the out of sample estimate is given by,

$$E_0 \leq E_T + 2.174 \times 10^{-3}$$

$$E_0 \leq 0.1016$$

This the bound on the out of sample error.

With a probability of 95%, out of sample error of the final model will be less than or equal to 0.1016.

Improvements

- Hypothesis set should be increased, to find a better model.
- Development of the architecture of the CNN model can be approached as a classification problem first, and then as a regression problem.
- The R2 score can be increased when the model overfits the data. The regularization helps to have a smooth curve. Understanding about these two is one of the major learnings.