Robot Learning and Vision for Navigation

HW1 – Imitation Learning

Andres S. Chavez Armijos

09/26/2020

# Network Design

## Data Loading:

In order to load the data provided by the expert a function that load the expert data was implemented. The data includes observation data with its corresponding label. This data is used for training of the neural network. A snapshot of the data loading function is shown in Figure 3.1.

## Training Function:

An epoch indicates the number of passes of the entire dataset when training occurs. This indicates the number of times that the training of different dataset batches occurs. A batch is a set of samples which size is smaller than the number of samples available in the dataset. The main purpose of using a batch set is to reduce the stress on the computer memory. Additionally, when using a batch approach, the possibility of the gradient straying in a region of local minima is reduced since a stationary point with respect to the error function for the whole data set will not be stationary of each of the data points contained in the batch set. The lines 43-48 of the training.py file, computes the forward pass of the training, then computes the loss corresponding to the previously computed estimation, and then the backpropagation is computed so the parameters with the least cost can be computed.

## Label Classification

The actions provided by the expert come in a vector format containing three values , However, the acceleration and braking parameters are independent and cannot be used together. Therefore, 9 classes are computed. These actions with their respective labels are listed in Table 3.1

## Architecture

The architecture defined for the system is defined in the Table 3.2

## Network Implementation

The architecture mentioned in Table 1.1 is implemented. Additionally, the input observation uses greyscale images instead of a 3 channel RGB image. The decision for greyscale over RGB is to reduce model complexity since color does not have significance over this classification.

In order to obtain higher accuracy and lower loss different hyperparameters were tested. These include the changing of the optimizer, changing total number of epochs, batch size, and learning rate. In order to evaluate the training, a set from the training data was used to compute the mean accuracy and loss using the trained parameters. The average testing accuracy was 0.998 and the average testing loss was 0.0095. Tuning the hyperparameters affect the accuracy, because each of the factors can contribute towards the avoidance of overfitting, local minima, and the improvement of performance. Nevertheless, despite the seemingly good accuracy and loss, when evaluating the model, the vehicle failed to take curves and incurred in a tendency to go straight. This resulted in an average reward of -16987.11. The loss and accuracy history are shown in Figures 3.2 and 3.3 respectively.

## Expert Dataset

The behavior produced by the previously mentioned neural network architecture was produced by class imbalance where the expert training dataset contains a higher number of forward data compared to the rest of classes. Therefore, other datasets with higher number of data can be used to improve the performance of the classification task. Thus, a good training dataset is one where the agent can learn how to recover from failures and is not only going straight. This means that the dataset needs to include the most variability.

# Network Improvements

## Observation

The extract\_sensor\_values function uses the observation input to compute the speed, angular rate, braking, and steering information. This information is encoded within the image environment on pixels x = 84:94. By integrating the usage of the sensors onto the network, the loss and accuracy training history is shown in Figures 3.4 and 3.5 respectively. It is possible to see that the model achieves higher accuracy faster despite the lower loss decrease. This may be the result of the inclusion of more information. The sensor information is added as a concatenation to the 4th layer, or first dense layer.

## Multiclass Prediction

In order to provide a multilabel classification, a different multi one-hot encoding classification was implemented. This includes a binary classification for {forward}, {left}, {right}, and {brake} actions. In this case, the classes are not independent from each other. For this purpose, the architecture is modified as shown in Table 3.2 where it can be seen that the last activation function was changed for a Sigmoid function. The training loss history is shown in Figure 3.6. It is possible to see that the loss is reduced at a higher rate since the likelihood of an image to belong to two different classes or more is higher. Additionally, the number of classes for classification is lower. Thus, reducing the complexity of the model. The average reward for the multiclass prediction task was 233.97

## Regression vs Classification

For the purpose of regression, the loss function used is the mean squared error, where the only purpose is to measure the distance from the truth. Both regression and classification use known datasets to make predictions. However, unlike classification, regression tries to estimate a mapping function that generates a continuous output of variables. This decreases model complexity since the loss function takes less computational resources. Additionally, the model prediction can create smoothness compared to the classification task where the output is discrete. However, when using regression, the estimates can produce erratic behavior that can translate into inputs that change drastically. Thus, creating vehicle instability. For our case of imitation learning, a regression model is doable, yet not reasonable unless corrected for smoothness. The overall. This is the reason for which the vehicle generated a shaking behavior and slower acceleration, obtaining an average reward of 133.02.

## Data Augmentation

The idea of using data augmentation is to create a wider sample dataset that can be used for training. When working with images, data augmentation can be accomplished by reusing existing data and distorting it either by adding noise to the image or rotating the image. For this task a new dataset including color jittering, random rotations, and random image flips were used. A sample image is shown in Figure 3.7. In general, the added dataset increased the training computation time. Additionally, it can be seen from Figure 3.8 that the loss takes more time being reduced. But the overall accuracy is higher than for the other models. The average reward for the run is 301.735 which is higher than the previously reported values

## Fine Tunning

In order to improve the performance of the network, dropout was included as part of several layers. The idea of dropout is to drop random features with probability of 0.3 to avoid overfitting. The equivalent network is shown in Table 3.4. Additionally, as part of class balancing, a different weight gain was added to the network reducing the weights for forward and do nothing by 0.2%. Additionally, in order to improve the model accuracy, a preprocessing function was written in order to ease the computation by smoothing, normalizing, and masking the image. The resulting average reward was 320.

# Appendix

Text

Description automatically generated

Figure 3.1 Data loading function snapshot

Table 3.1 Action Classification

|  |  |
| --- | --- |
| **Action** | **Label** |
| [0.0, 0.0, 0.0] | Straight |
| [0.0, 0.5, 0.0] | Accelerate |
| [1.0, 0.0, 0.0] | Right |
| [1.0, 0.0, 0.8] | Right/brake |
| [0.0, 0.0, 0.8] | Brake |
| [-1.0, 0.0, 0.8] | Left/brake |
| [-1.0, 0.0, 0.0] | Left |
| [1.0, 0.5, 0.0] | Right/accelerate |
| [-1.0, 0.5, 0.0] | Left/accelerate |

Table 3.1 Neural Network Architecture

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Feature Map** | **Output Channels** | **Activation** |
| Convolutional 2D | 1x96x96 | 2 | Leaky ReLU |
| Convolutional 2D | 2x48x48 | 4 | Leaky ReLU |
| Convolutional 2D | 4x24x24 | 8 | Leaky ReLU |
| Flatten | 8\*24\*24 | 1 | - |
| Dense | 8\*24\*24 | 64 | Leaky ReLU |
| Dense | 64 | 32 | Leaky ReLU |
| Dense | 32 | 9 | Softmax |

Chart, line chart

Description automatically generated

Figure 3.2 Initial Network Accuracy History

Chart, line chart

Description automatically generated

Figure 3.3 Initial Network Loss History

A picture containing chart

Description automatically generated

Figure 3.4 Accuracy history on architecture with sensor concatenation

Chart, line chart

Description automatically generated

Figure 3.5 Loss history on architecture with sensor concatenation

Table 3.3 Action Classification

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Feature Map** | **Output Channels** | **Activation** |
| Convolutional 2D | 1x96x96 | 2 | Leaky ReLU |
| Convolutional 2D | 2x48x48 | 4 | Leaky ReLU |
| Convolutional 2D | 4x24x24 | 8 | Leaky ReLU |
| Flatten | 8\*24\*24 | 1 | - |
| Dense | 8\*24\*24 | 64 | Leaky ReLU |
| Dense | 64 | 32 | Leaky ReLU |
| Dense | 32 | 4 | Sigmoid |

Chart

Description automatically generated

Figure 3.6 Multilabel classification loss training history

A picture containing graphical user interface

Description automatically generatedA picture containing clock, computer

Description automatically generated

Figure 3.7 Sample image distortion (image undistorted on the left, image distorted on the right)

Chart

Description automatically generated

Figure 3.8 Data Augmentation Accuracy History

Chart, line chart

Description automatically generated

Figure 3.9 Data Augmentation Loss History

Table 3.4 Neural Network Architecture

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Feature Map** | **Output Channels** | **Activation** |
| Convolutional 2D | 1x96x96 | 2 | Leaky ReLU |
| Dropout | 1x96x96 | 2 |  |
| Convolutional 2D | 2x48x48 | 4 | Leaky ReLU |
| Dropout | 2x48x48 | 4 |  |
| Convolutional 2D | 4x24x24 | 8 | Leaky ReLU |
| Flatten | 8\*24\*24 | 1 | - |
| Dropout | 8\*24\*24 | 1 |  |
| Dense | 8\*24\*24 | 64 | Leaky ReLU |
| Dropout | 8\*24\*24 | 64 |  |
| Dense | 64 | 32 | Leaky ReLU |
| Dropout | 64 | 32 |  |
| Dense | 32 | 9 | Softmax |