Deep Learning laboratory Report

Assignment 3

Mohamed AbouHussein
Louay Abdelgawad

Department of Machine Learning
University of Freiburg
November 2016

Summary

This report is related to the work of implementing a convolutional neural network using `tensorflow as tf` and `keras` package for motion planning. The report is divided into 4 sections as follows: Section 1 represent a simple explanation of the implementation, section 2 represents the designed convolutional neural network, section 3 represents the results and discussion and section 4 mentions several possible improvements.

Report

1 Convolutional Neural Network Implementation

The algorithm begins by loading the `get_data.py` and saving the input and their respective output locally in files. After that we train the agent. In file `train_agent.py`, we initialize the `Simulator` and the `TransitionTable`. The functions of both objects will be used later. We begin by saving the input data and the test labels `x` and `y` as well as the validation data and label `x_val` and `y_val`. The `model` is initialized using `keras` package that acts over `tensorflow`. The functions we utilized from `keras` are `Convolutional2D`, `Activation`, `MaxPooling2D`, `Flatten`, `Dense` and `Dropout`. The exact CNN architecture will be illustrated in the next section.

2 Designed Convolutional Neural Network

For the design of the conv network, the procedure utilized was to create a single convolutional layer and pool layer at a time. The layer parameters was randomly tried out from a range of values. The best estimated random values was the one implemented. Afterwards, another layer is added and the same procedure is tried out. In addition, the hypermaraters is also set in similar manner. The previous procedure has resulted in the following: The first conv layer included 64 `Covolutional2D()` filters, 3x1 `filter_size`. The activation function for the first layer was `relu`. Another `Convolutional2D()` layer is added that has the same specs and activation function but of size 32 instead. After that, a `MaxPooling2D()` of frame size 1x2 is added. Finally, the data is `Flatten()`ed and `Dense()`d and the `softmax` function is used on the output layer. Stochastic gradient descent `sgd` is selected as the optimization algorithm for gradient descent. The `epoch_size` and the `batch_size` selected was 20 and 64 respectively. The results of the conv network is discussed in section 3.

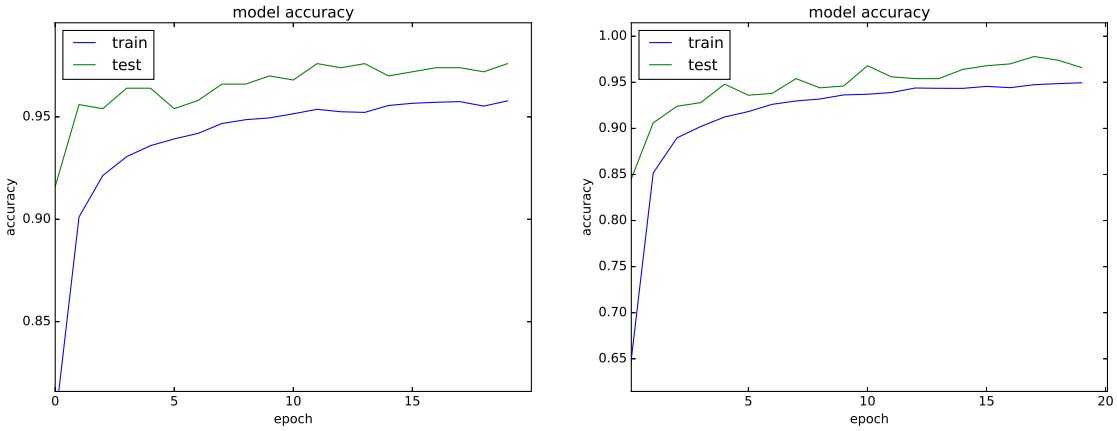
3 Results and Performance Evaluation

3.1 Local view

From our point of view, the trained agents reaches the goal in all occasions, which means it always gets an accuracy of 100%. That was concluded after running the `test_agent` several times and observing the results. However, in some rare cases the results weren't

optimal as the agent may take two or three extra steps more than that of A* algorithm.

3.2 Changing History length



(a) History length = 4 **(b)** History length = 10

Figure 1: Training and Test accuracies for different History lengths

In contrary to our expectations, the effect of increasing the history length was a negative effect, which is noted only when running the `test_agent`, as the agent tends to make less optimal solutions after increasing the history length from 4 to 10. In addition, as seen in fig 1, the train and test error are nearly the same except that the `test_error` in that of history length 4 is slightly higher than that of history length 10, 97.6% and 96.8% respectively.

3.3 Changing Target location or Map

As expected, changing either the Target location or the Map would lead to a huge error, due to the fact that the agent is trained in a specific map with a specific target location. That means that the agent will always try to reach the target position specified during training. Solutions for such problem are provided in section 4

4 Improvements

In order to improve our agent to be able to solve any Map and reach any Target location, two improvements are suggested. Firstly, instead of training on just one target location, training is done instead by considering all the locations in the map as target locations, and with different map shapes. After that, the training locations as well as the maps are augmented in the training dataset of the agent. However, the downside of this problem that the size of the training dataset will increase exponentially. The second suggested method

and the more efficient one is by using reinforcement learning which is done by making the agent train itself using rewards, and using the Q-values method to construct the Deep Q network, and this is different than the method used in this report as it doesn't use the A* algorithm to get the training dataset, instead it just keeps running to a specified number of episodes while improving itself in each run until reaching a satisfactory result.