Introduction to Artificial Intelligence: Assignment 2

Vladimir Ivanov, Neelesh Kumar and Rahul Shome

August 7, 2018

Please ensure compliance with Rutgers academic integrity policies [1] in course of completing this assignment. Only one member of the group has to submit the completed assignment. Only electronic submissions will be accepted. Pictures of handwritten submissions will be penalized. Please make sure that the assignment submission includes the name of *every member of your group.*
Due date: August 12, 11:55pm.

1 Introduction

In this assignment you will be building a neural network in tensorflow to classify hand written digits. To train your network you will be using the famous MNIST training dataset. It is already built into the assignment code file. Here are the key deliverables that should be part of a report on sakai, along with the coded script files:

1. Build a basic neural network model that learns above 90 percent accuracy. Answer theoretical questions in report. [40 pts.]
2. Generate an accuracy plot. Discuss model behaviour. [20 pts.]
3. Integrate the neural network model into your existing robotics A* search code. [40 pts.]
4. EXTRA CREDIT: Improve your model to achieve above 97 percent accuracy. [50 pts]

   `NOTE:` You are free to use resources available readily on the internet. However, you will be asked to explain what you did and your design choices during the demonstration.

2 Getting started

To start, download the following files from Sakai(or refer to the files in the `$PRACSYS_PATH/prx_core/sensing` directory of the repository explained in Section 6):

1. `tf_train_model_ASSIGNMENT_FILE.py`
2. `tf_evaluate_model_code.py`

Place them into the same folder. This directory will be where all your tensorflow models will be stored. To build your model, you will need to open the first file above and fill in blocks of missing code labeled as:

1. `################### YOUR MODEL GOES HERE ###################################`
2. `############## YOUR LOSS AND ACCURACY FUNCTIONS GO HERE ##################`
3. `############## YOUR TRAINING FUNCTION GOES HERE ###########################`
4. `############## YOUR TRAINING LOOP CODE GOES HERE ##########################`
5. `############## YOUR MODEL ACCURACY PLOT CODE GOES HERE ####################`

[1] http://academicintegrity.rutgers.edu/

In order to understand how to fill in the above parts it is advised to go through the basic tensorflow tutorial. Specifically, take a look at the **Tensorflow Core Tutorial** and the **tf.train API** sections in:
https://www.tensorflow.org/versions/r1.0/get_started/get_started

And for MNIST classification:
https://www.tensorflow.org/versions/r1.0/get_started/mnist/beginners

3 Specifics On Each Code Block

3.1 MODEL BLOCK

3.1.1 Include in Code

This part of the code is where you will construct your main model. It will take in a 784 (28*28) length array as input. Hint: use placeholder for input. Name the placeholder as:

```
ph_x
```

Also, make a placeholder for data labels to be inputed, named as:

```
ph_y_
```

This will be the MNIST hand written digit array. MNIST dataset is already built into the code file. All that needs to be done is to feed it into your model during training and evaluation using a feed dictionary as shown in the comments section of the code file.

The output must be an array of length 10. Each index in the array corresponds to a digit 0 to 9. The real output value of each index must represent the probability of the input image being the digit represented by that index. As an example, if we feed an input array representing the digit 0, then the output index with the greatest real value in the output should be index 0. This will constitute the correct classification. As a hint, to get this probability output, use a softmax neuron for the output layer.

Possible ways of improving your model in order to achieve 99 percent accuracy are:

1. Multiple neuron layers.
2. Convolutional layers with max pooling.
3. Better weight(parameter) initialization.

3.1.2 Include in Report

Explain the structure of you model. Specifically:

1. How many layers you used.
2. What type of layers you used.
3. What nonlinearities you used.
4. What is the input and output of your model

3.2 LOSS AND ACCURACY BLOCK

3.2.1 Include in Code

This section should be no longer than 2 lines of code. First line should be your loss function. Loss function measures the difference or distance between the output your model gives and the correct output provided by the dataset. Since output is a vector, one common metric to use as loss is the Euclidian distance between the two vectors. This however is not always optimal. Try different loss functions to see which one works best.

Second line is the accuracy function which is used to check how well you model is performing on the validation and test datasets. Accuracy is usually evaluated every x number of training batches to see how much learning the model has achieved. In your model, name your accuracy function as:

```
op_accuracy
```

You will need to implement this in your training loop in order to generate the accuracy plot.

3.2.2 Include in Report

Write out the formula you used for your loss function. Explain what is the purpose of the loss function in neural network models. Does the loss function have a significant effect on ability of model to achieve high accuracy. Write out the formula you used to evaluate the accuracy function. Explain what the accuracy function does. What is its output?

3.3 TRAINING FUNCTION BLOCK

3.3.1 Include in Code

This block is one line of code. It defines the training algorithm that will make updates too all the weights(parameters) of your neural network model. One common algorithms is the Adam optimizer.

3.3.2 Include in Report

Specify the optimizer you are using and the learning step you used. Comment on the relationship between learning step size and training of the model in terms of speed of learning and ability to converge to a local minima.

3.4 TRAINING LOOP BLOCK

3.4.1 Include in Code

This is where you implement a loop to go through the MNIST dataset. On each pass of the loop, you should run your training function. The training optimization algorithm you coded in the previous code block will do the following:

1. Run MNIST data through your model.
2. Use the MNIST labels to evaluate the error of your model.
3. Change model parameters according to the errors computed.

Either on each pass or after some number of passes, you should run an accuracy measure on the validation MNIST dataset. You should find a way to store the output of these validation evaluations in order to produced the plot in the next code block.

3.4.2 Include in Report

Neural network models are very data intensive. This means that every bit of additional data can make a big difference in your model performance. In your report, explain the rational and purpose of breaking up your training dataset into training, validation, and test sets hence effectively reducing your training data size. Specifically, explain the tradeoff or dilemma that machine learning engineers face in this case and why it makes sense to still break down your data set into the above mentioned 3 subsets.

3.5 ACCURACY PLOT BLOCK

3.5.1 Include in Code

This code block is supposed to produce and save the training accuracy plot of your model. The x axis should be training iterations. The y axis should be the accuracy measure. You should evaluate the model accuracy every 100 training iterations. Accuracy evaluation should be run on a batch of training data, validation data, and test data. Your plot should plot the evolution of all 3 accuracies as the model trains.

3.5.2 Include in Report

Show the plot in your report. Comment on any relationships you see between the 3 different accuracies. Can you explain why this is happening? Regardless of what your experimental results produce, what kind of relationship do you expect to see from a theoretical perspective.

4 Launching The Training File

Once you have filled in all the necessary code blocks, you can launch the model training. To do so, open a python terminal, navigate to your tensorflow directory, and input the following commands:

```
from YOUR_TENSORFLOW_DIRECTORY import tf_train_model_code as tm

model_train = tm.build_train()
model_train.build_train_network(model_version=YOUR_MODEL_VERSION_NUMBER)
```

5 Training Complete - Next Steps

Once you have built your model and trained it using the assignment code file, it will save your model in 3 files with endings:

1. .data
2. .index
3. .meta

You need all three files to be able to integrate your model into the previous assignment. To check that it works, use the second file in the Sakai download to run an evaluation check on your model. Make sure you are getting approximately the same accuracy reading as during training. To launch the second file, run the following commands in python terminal in your tensorflow directory.

```
from YOUR_TENSORFLOW_DIRECTORY import tf_evaluate_model_code as emc

model_eval = emc.evaluate_model()
model_eval.evaluate_model(model_version=YOUR_MODEL_VERSION_NUMBER)
```

6 Integration

Version control is a way for large scale projects to allow multiple programmers to collaborate in an organized fashion. It is widely used across academia and industry to keep track of changes made to such projects. These projects are called repositories, and in Assignment 1, you downloaded the repository corresponding to https://rahulshome@bitbucket.org/rahulshome/cs_440_assignment.

You need to sync your repository from bitbucket. This is related to version control rules (https://en.wikipedia.org/wiki/Version_control). The code you downloaded for Assignment 1, has to be updated to reflect changes that have been made by us on *bitbucket*. Doing so will attempt to reconcile any changes you have made to the code on your local machine, and any changes that have been made to the repository on *bitbucket*. Please note that, if you have made additional changes to the code in Assignment 1, a conflict may happen(i.e., the same line of code you changed was also changed on the repository so it is undefined, which change persists after the update). Make sure at the end of the following steps you have updated all the changes from the repository, but your own changes related to $A*$ still persist on your local copy.

```
cd $PRACSYS_PATH
hg pull
hg up
```

If there are no conflicts, the above three commands should ensure the update has happened properly. If there is a conflict, that indicates that you need to manually fix the files. While there are many ways to do this, one way would be to go to https://bitbucket.org/rahulshome/cs_440_assignment/src/default/ and view our changes. You can manually edit the your local copies of the files where the conflict arises to ensure they have both our and your changes.

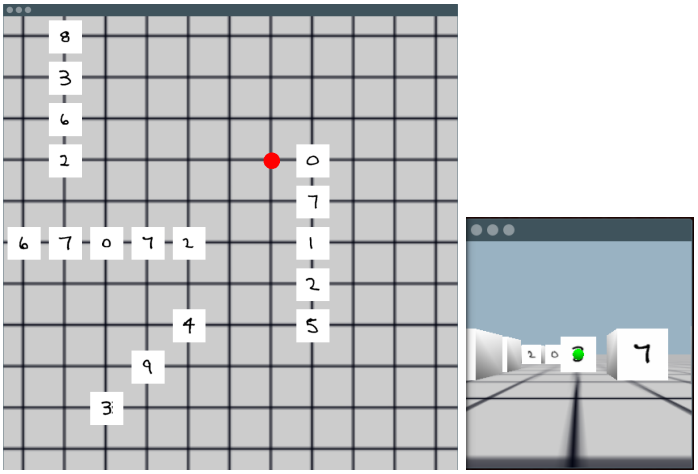

Figure 1: The image on the left represents the top-down view of the grid maze. The numbers in the top-down view represent the **id** associated with the location of the agent. The image on the right represents the simulated camera attached to the red agent, that always points to the right of the red agent. The number visible to the camera at a goal(green dot) represent the **id**(or top down number) of the correct next goal. This means that the red agent moves to the location beside the **id** sensed from the simulated camera image. Any motion needs to still be collision free, as in Assignment 1.

6.1 Sensing

In Assignment 1, you were tasked with designing a planning subroutine for an agent in a grid maze environment. In this problem, you will be tasked with designing a sensing subroutine for the agent as well. As the agent moves around the maze, it will see the walls of the maze around it, from its perspective.

Execute the command

```
roslaunch prx_core assignment_2.launch \
env:="$PRACSYS_PATH/prx_core/launches/test_maze.txt"
```

to see the windows similar to Fig 1.

The objective of the red agent is to perceive the environment through the simulated camera screen in Fig 1(*right*), and determine the next checkpoint. The next checkpoint or goal(green dot) would be placed beside a block matching the sensed number. If the sensing is incorrect, the sequence of checkpoints will diverge from the valid sequence. This can be empirically checked if you with your sensing model by running the code to determine whether the agent goes to the checkpoint(green dot) beside what was on the camera screen at the previous checkpoint.

You are asked to fill in the function `predict()` in the script that is already invoked from the code in `demo_application_t::sense()` with the path to the image file captured by the camera.

`$PRACSYS_PATH/prx_core/sensing/sense_environment.py`

The function `predict` must return the digit corresponding to the array being passed as input to it. The array corresponds to the flattened binary representation of the image. (*Hint: This array needs to be converted to a form that you can use with your trained model. Once you also have access to the model, you need to derive the best estimate out of it, and return the class it corresponds to.*)

You will be asked to demonstrate the integrated code during the demo and your grade will depend on how accurately the agent follows the checkpoint sequence in a randomly generated maze.

In your report include the following:

1. Do you think the accuracy of your model is as high as what you recorded in the accuracy plots?
2. What do you think explains the difference? Is this a problem symptomatic of machine learning?

6.2 Extra Credit

6.3 Include in Code

Now that you have trained a basic neural network model on MNIST, try to expand your model to improve upon your current model's accuracy measure. To do so, consider adding the following to your model:

1. Adding additional layers.
2. Adding convolutional max pooling layers.
3. Changing weight and bias initialization schemes.
4. Adding dropout to training.
5. Longer training and different batch size

A good guide on achieving 99 percent accuracy on MNIST dataset is the following page: https://www.tensorflow.org/versions/r1.0/get_started/mnist/pros

6.4 Include in Report

Comment on what accuracy you were able to achieve. Which additions to your model or changes to your training appeared to be most beneficial. Which ones did not help improve accuracy. Does increasing duration of training continuously keep improving the accuracy?