CS 5600/6600
Assignment 5

Vladimir Kulyukin
Department of Computer Science
Utah State University

October 5, 2019

Learning Objectives

1. ConvNets
2. Training and Testing ConvNets
3. Reinforcement Learning
4. Deep Q Learning

Introduction

For this assignment you'll need to install tensorflow (`tensorflow.org`), tflearn (`tflearn.org`), and Open AI Gym (`gym.openai.com/`). Actually, tflearn requires tensorflow. You can go directly to `tflearn.org` and install all the dependencies including tensorflow.

I included in this assignment `tf01.py`, `tf02.py`, `tf03.py`, and `tf04.py`. These are small Py programs I wrote when I was teaching myself tensorflow by going through the online documentation. If you haven't played with tensorflow before, I suggest you play with them for a few minutes to get more comfortable with this library. We went over the code in `tf01.py` in lecture 6. The other short programs are similar. They build simple computation graphs and run them in tensorflow sessions.

Problem 1 (3 points)

Use tflearn to train and persist 5 convnets. Each of your convnets should have several convolutional layers. Experiment with 2, 3, 4, and 5 layers. To safe you some typing, I created the following Py files with some starter code:

1. `mnist_convnet_1.py`;
2. `mnist_convnet_2.py`;
3. `mnist_convnet_3.py`;
4. `mnist_convnet_4.py`;
5. `mnist_convnet_5.py`.

Each of the above files has the function `build_tflearn_convnet_i()`, where $1 \leq i \leq 5$. I defined `build_tflearn_convent_1()` for you to follow as an example. You should define `build_tflearn_convent_2()`,

build_tflearn_convent_3(), build_tflearn_convent_4(), and build_tflearn_convent_5(). When you build your network correctly and run mnist_convnet_i.py, where $1 \leq i \leq 5$, you should see the following output in your command window. The output shows my training convnet 1 just for 2 epochs. Of course, you should train your nets for many more epochs.

```
---------------------------------
Run id: MNIST_ConvNet_1
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 50000
Validation samples: 10000
--
Training Step: 5000  | total loss: 0.16273 | time: 89.854s
| SGD | epoch: 001 | loss: 0.16273 - acc: 0.9524 |
val_loss: 0.12607 - val_acc: 0.9604 -- iter: 50000/50000
--
Training Step: 10000  | total loss: 0.09704 | time: 67.151s
| SGD | epoch: 002 | loss: 0.09704 - acc: 0.9691 |
val_loss: 0.06115 - val_acc: 0.9813 -- iter: 50000/50000
--
```

Record the physical time it took you to train all five networks. You don't have to be exact. An approximate number will do. After your training is completed, tflearn persists the trained model in a directory of your choice by creating three files: data, index, and meta. For example, if you persist your first network in /home/vladimir/research/mnist_convnets/hw06_my_net_01.tfl, you will see the following files created in it after the net is persisted:

1. hw06_my_net_01.tfl.data-00000-of-00001;
2. hw06_my_net_01.tfl.index;
3. hw06_my_net_01.tfl.meta.

Now go to test_mnist_convnets.py and define the five loading functions load_mnist_convnet_i, where $1 \geq i \geq 5$, that will load each of your persisted convnets. I defined load_mnist_convnet_1 for you to follow as an example. Change the variables net_path_1, net_path_2, net_path_3, net_path_4, net_path_5 accordingly. Here is how you can load and test a persisted ConvNet. The function test_convnet_model is implemented for you. Here is how you can load a trained network and test it against the mnist validation data (validX) and target (validY).

```
>>> net_model = load_mnist_convnet_1(net_path_1)
>>> test_convnet_model(net_model, validX, validY)
0.999
```

Of course, your accuracy may be different (but it should be above 80% for sure). I encourage you to experiment with different activation functions in your layers. For example, here is how you can feed an 28x28 input layer to a convolutional layer of 20 features with a 5x5 local receptive field and ReLU neurons.

```
input_layer = input_data(shape=[None, 28, 28, 1])
conv_layer_1  = conv_2d(input_layer,
                        nb_filter=20,
                        filter_size=5,
                        activation='relu',
                        name='conv_layer_1')
```

Here is how you can activate with tanhs and sigmoids.

```
conv_layer_1  = conv_2d(input_layer,
                        nb_filter=20,
                        filter_size=5,
                        activation='tanh',
                        name='conv_layer_1')

conv_layer_1  = conv_2d(input_layer,
                        nb_filter=20,
                        filter_size=5,
                        activation='sigmoid',
                        name='conv_layer_1')
```

Use the function `test_tflearn_convnet_model()` to compute the average accuracy of your ConvNets on the validation data, i.e., on `validX` and `validY`. Write a 1-page report with a table where you compare the average performance of different convnets and the physical times it took you to train them. You can put all your observations in one table. So, are ConvNets worth it? Are they better than ANNs on MNIST? Briefly argue why or why not based on your discoveries. Save your report in `hw06_report.pdf`.

Problem 2 (2 points)

Recall the mountain car game we discussed in lecture 6. The assignment zip contains the `mountain_car` folder with three files: `MCar.py`, `Memory.py`, and `Model.py`. After you install Open AI Gym, you can go to the command line and run this Deep Q learner right away as follows:

```
>>> python MCar.py
```

If everything is installed correctly, you should see a small car bouncing left and right on the screen (as I showed you at the end of lecture 6) and the following lines in the command window.

```
Episode 1 of 10
Step 1, Total reward: -1.0, Eps: 0.999901004949835
Step 2, Total reward: -2.0, Eps: 0.9998020197986801
Step 3, Total reward: -3.0, Eps: 0.9997030445455454
Step 4, Total reward: -4.0, Eps: 0.999604079189441
Step 5, Total reward: -5.0, Eps: 0.9995051237293776
Step 6, Total reward: -6.0, Eps: 0.9994061781643654
Step 7, Total reward: -7.0, Eps: 0.9993072424934148
Step 8, Total reward: -8.0, Eps: 0.9992083167155369
Step 9, Total reward: -9.0, Eps: 0.9991094008297421
...
```

After the RL training stops, the program displays two plots: the rewards plot and the maximum position plot. These plots are constructed from the two array slots in the `MCar` object: `self._reward_store` and `self._max_x_store`. You can use these arrays to compute the total award and the average x position reached by the car over all episodes. The number of episodes is defined in the variable `num_episodes` in the function `train_car()` in `MCar.py`.

`MCar.py` implements the temporal difference RL algorithm we discussed in lecture 6. You don't need to modify this file (other than modifying `num_episodes`). The file `Model.py` is where you can

define different NN models that will be trained by RL to drive the car. For example, the method `_define_model_1` builds a fully connected 2x50x50x3 network trained to drive the car. The input to this network is the current state of the mountain car (position and velocity). The output is a 3-element vector of actions: push left, no push, push right.

```
def _define_model_1(self):
    self._states = tf.placeholder(shape=[None, self._num_states],
                                  dtype=tf.float32)
    ## This is the Q(s, a) table.
    ## The number of columns is determined by the input fed to it
    ## The number of actions is 3. So, the size of _q_s_a is ? x 3,
    ## because there are 3 actions in the game.
    self._q_s_a = tf.placeholder(shape=[None, self._num_actions],
                                 dtype=tf.float32)
    # create two fully connected hidden layers
    self._fc1 = tf.layers.dense(self._states, 50, activation=tf.nn.relu)
    self._fc2 = tf.layers.dense(self._fc1, 50, activation=tf.nn.relu)
    self._logits = tf.layers.dense(self._fc2, self._num_actions)
    loss = tf.losses.mean_squared_error(self._q_s_a, self._logits)
    self._optimizer = tf.train.AdamOptimizer().minimize(loss)
    self._var_init = tf.global_variables_initializer()
```

Define four other methods in `MCar.py`: `_define_model_2()`, `_define_model_3()`, `_define_model_4()`, and `_define_model_5()` that construct different fullly connected NN architectures. Train each model for 50 episodes and compute for every NN model the total award and the average x position. Add more layers, experiment with different activation function in each layer. Write a 1 page report with a table where you briefly document your discoveries. Which architecture gave you the highest total award and the greatest average x postion? Save your report in `hw06_report.pdf`.

What to Submit

1. `mnist_convnet_1.py`;
2. `mnist_convnet_2.py`;
3. `mnist_convnet_3.py`;
4. `mnist_convnet_4.py`;
5. `mnist_convnet_5.py`;
6. `MCar.py`;
7. `Model.py`; you shouldn't modify this file but submit it for the sake of completeness;
8. `Memory.py`; you shouldn't modify this file but submit it for the sake of completeness;
9. `hw06_report.pdf`.

Happy Hacking!