COMP9444 Neural Networks and Deep Learning

Session 2, 2018

Project 1 – Basic TensorFlow and Digit Recognition

Due: Sunday 26 August, 23:59 pm
Marks: 10% of final assessment

Introduction

This and all following assignments will use the Python API for the TensorFlow library. TensorFlow (TF) is an opensource library primarily used to construct, train and evaluate machine learning models. TF allows rapid development and supports automatic differentiation – meaning backprop is able to be done automatically for any model adequately defined. TF also abstracts away much of the low-level code required to set up training on GPU's; in many cases TF will automatically detect and utilize your computer's GPU if it has one. Central to the design of TF is the concept of a 'graph' – a low level representation of a model consisting of nodes and tensors. Broadly, implementing a TF model can be broken down into two sections; creating the graph, and training/testing it. This assignment is mainly concerned with graph creation. You can read more about the general structure of TensorFlow here.

This assignment is split into two parts. Part 1 contains basic introductory exercises using TensorFlow's core API. In Part 2, you will be implementing a single layer network, a two layer network and a convolutional network to classify handwritten digits. We will work with the MNIST dataset, a common dataset used to evaluate Machine Learning models.

Be sure to read this document in it's entirety before starting the assignment.

Notes/Common Problems

1. Tensorflow has several levels of abstraction in it's API, with some sections designed for rapid prototyping (Keras, Estimators), others for more mid-level functionality (`tf.layers,tf.nn`), and some exposing base functionality. For this course we will mostly be focusing on the core and mid-level APIs. For this Assignment you may **NOT** make calls to the following parts of the API;
   - `tf.contrib` (including eager). By extension: do not follow any example that uses the following call at any point in the code. `tf.enable_eager_execution()`.
   - `tf.keras`
   - Anything to do with estimators (`tf.estimator`)

   Calls from these sections of the API will cause your code to fail the autotests and for you to receive a mark of 0 for that section of the assignment. It is suggested to call functions from:
   - `tf.layers` (this is mostly wrappers for `tf.nn` but with nicer parameter structuring)
   - `tf.nn`

   Calls from other parts of the API such as `tf.math` are allowed but discouraged – there is probably an easier way to do what you are trying to do.

2. Tensorflow is under extremely active development, and many Google searches will point to answers that use old versions of the API, or older version of the docs. Ensure you are always viewing information relevant to 1.9
3. There a great many tutorials dealing with MNIST and TensorFlow. Because of the above point their utility is limited for this assignment. It is suggested to read and understand the docs themselves as opposed to blindly following tutorials.
4. In general, you should never call functions that have depreciation warnings in the documentation (e.g. `tf.nn.softmax_cross_entropy_with_logits()`)

Preliminaries

Before commencing this assignment, you should download and install TensorFlow (instructions here) and the appropriate python version (3.5 – 3.7 are all ok, do not use 2.7x). It is also helpful to complete the 'Low level Introduction' tutorial located on the TensorFlow website here.

Copy the archive `src.zip` into your own filespace and unzip it. Then type

```
cd src
```

You will see two files: `train.py` and `hw1.py`

Part I [3 marks]

Implement the following functions within `hw1.py`. You do not need to use `train.py` in Part I. Note that the example function `add_consts()` has been completed for you.

1. `add_consts()` (*already implemented): Construct a TensorFlow graph that declares 3 constants, 5.1, 1.0 and 5.9 and adds these together, and returns the resulting tensor.
2. `add_consts_with_placeholder() [1 mark]`: Construct a TensorFlow graph that constructs 2 constants, 5.1, 1.0 and one TensorFlow placeholder of type `tf.float32` that accepts a scalar input, and adds these three values together, returning as a tuple (and in the following order), the final sum and created placeholder.
3. `my_relu() [1 mark]`: Implement a ReLU activation function that takes a scalar `tf.placeholder` as input and returns the appropriate output. A ReLU is one of the simplest activation functions, simply setting all negative values passed into it to 0, and passing positive values through unchanged. For more information, see here.
4. `my_perceptron() [1 mark]` Implement a single perceptron that takes `x` inputs and produces one output, using the RelU activation function you defined previously.: Specifically, implement a function that takes an argument `x`, and creates a `tf.placeholder` of length `x`. Then create a trainable TF variable for the weights `w` (*hint: look at tf.get_variable() and the initalizer argument.*). Ensure this variable is set to be initialized as all ones. Multiply and sum the weights and inputs following the peceptron outlined in the lecture slides. Finally, call your ReLU activation function. Return the placeholder and output in that order as a tuple. The code will be tested using the following init scheme

```
# graph def (your code called)
init = tf.global_variables_initializer()
self.sess.run(init)
```

```
                # tests here
```

Part II [7 marks]

Stage 0: Provided Code and Getting Started

Now run `train.py` by typing

```
python train.py your_student_id onelayer
```

The student number argument is not actually important for local testing - it's use is for automarking. When run for the first time, `train.py` should create a new folder called `data` and download a copy of the MNIST dataset into this folder. All subsequent runs of `train.py` will use this local data. (Don't worry about the `ValueError` at this stage. Deprecation warnings can also be ignored).

The file `train.py` contains the TensorFlow code required to create a session, build the graph, and run training and test iterations. It has been provided to assist you with the testing and evaluation of your model. While it is not required for this assignment to have a detailed understanding of this code, it will be useful when implementing your own models, and for later assignments.

IMPORTANT: `train.py` should not be modified at any point. Only `hw1.py` requires modification. A submission that does not run correctly with an unaltered `train.py` will lose marks.

The file `hw1.py` contains function definitions for the three networks to be created. You may also define helper functions in this file if necessary, as long as the original function names and arguments are not modified. Changing the function name, argument list, or return value will cause all tests to fail for that function. Your marks will be automatically generated by a test script, which will evaluate the correctness of the implemented networks. For this reason, it is important that you stick to the specification exactly. Networks that do not meet the specifications but otherwise function accurately, will be marked as incorrect.

The functions `input_placeholder()` and `target_placeholder()` specify the inputs and outputs of your networks in the TensorFlow graph. They have been implemented for you.

In addition, there is a function `train_step()` that passes batches of images to the constructed TensorFlow Graph during training. It's implementation should help you understand the shape and structure of the actual data that is being provided to the model.

Unless otherwise specified, the underlying type (`dtype`) for each TF object should be `tf.float32`. `INPUT_SIZE`, where it appears in comments, refers to the length of a flattened single image; in this case 784. `OUTPUT_SIZE`, where it appears in comments, refers to the length of a one-hot output vector; in this case 10.

In the provided file `hw1.py`, detailed specifications are provided in the comments for each function.

Stage 1: Single-Layer Network [2 marks]

Write a function `onelayer(X, Y, layersize=10)` which creates a TensorFlow model for a one layer neural network (sometimes also called logistic regression). Your model should consist of one fully connected layer with weights `w` and biases `b`, using softmax activation.

Your function should take two parameters `X` and `Y` that are TensorFlow placeholders as defined in `input_placeholder()` and `target_placeholder()`. It should return varibles `w, b, logits, preds, batch_xentropy` and `batch_loss`, where:

- `w` and `b` are TensorFlow variables representing the weights and biases, respectively
- `logits` and `preds` are the input to the activation function and its output
- `xentropy_loss` is the cross-entropy loss for each image in the batch
- `batch_loss` is the average of the cross-entropy loss for all images in the batch

Note that to return these variables you cannot simply call `tf.layers.dense`. A good approach might be to define your own dense layer using TF primitives.

Test your network on the MNIST dataset by typing

```
python train.py student_id onelayer
```

It should achieve about 92% accuracy after 5 epochs of training.

It is a good idea to submit your code after completing Stage 1, because the submit script will run some simple tests and give you some feedback on whether your model is correctly structured.

Stage 2: Two-Layer Network [2 marks]

Create a TensorFlow model for a Neural Network with two fully connected layers of weights `w1, w2` and biases `b1, b2`, with ReLU activation functions on the first layer, and softmax on the second. Your function should take two parameters `X` and `Y` that are TensorFlow placeholders as defined in `input_placeholder()` and `target_placeholder()`. It should return varibles `w1, b1, w2, b2, logits, preds, batch_xentropy` and `batch_loss`, where:

- `w1` and `b1` are TensorFlow variables representing the weights and biases of the first layer
- `w2` and `b2` are TensorFlow variables representing the weights and biases of the second layer
- `logits` and `preds` are the inputs to the final activation functions and their output
- `xentropy_loss` is the cross-entropy loss for each image in the batch
- `batch_loss` is the average of the cross-entropy loss for all images in the batch.

To test, run;

```
python train.py student_id twolayer
```

Stage 3: Convolutional Network [3 marks]

Create a TensorFlow model for a Convolutional Neural Network. This network should consist of two convolutional layers followed by a fully connected layer of the form:

conv_layer1 → conv_layer2 → fully-connected → output

Note that there are **no pooling** layers present in this model. Your function should take two parameters `X` and `Y` that are TensorFlow placeholders as defined in `input_placeholder()` and `target_placeholder()`. It should return varibles `conv1, conv2, w, b, logits, preds, batch_xentropy` and `batch_loss`, where:

- `conv1` is a convolutional layer of `convlayer_sizes[0]` filters of shape `filter_shape`
- `conv2` is a convolutional layer of `convlayer_sizes[1]` filters of shape `filter_shape`
- `w` and `b` are TensorFlow variables representing the weights and biases of the final fully connected layer
- `logits` and `preds` are the inputs to the final activation functions and their output
- `xentropy_loss` is the cross-entropy loss for each image in the batch
- `batch_loss` is the average of the cross-entropy loss for all images in the batch

Hints:

1. use `tf.layer.conv2d`
2. the final layer is very similar to the `onelayer` network, except that the input will be from the `conv2` layer. If you reshape the `conv2` output using `tf.reshape`, you should be able to call `onelayer()` to get the final layer of your network.

To test your implementation run;

```
python train.py student_id conv
```

It may take several minutes to train, depending on your processor.

Notes

All TensorFlow objects, if not otherwise specified, should be explicity created with `tf.float32` datatypes. Not specifying this datatype for variables and placeholders will cause your code to fail some tests.

The majority of this assignment can be implemented extremely compactly. If you find yourself writing 50+ line methods, it may be a good idea to look for a simpler solution. You do not need to make any additional imports, and code that makes use of external libraries such as `numpy`, will fail tests.

Visualizing Your Models

In addition to the output of `train.py`, you can view the progress of your models and the created TensorFlow graph using the TensorFlow visualization platform, TensorBoard. After beginning training, run the following command from the src directory:

```
tensorboard --logdir=./summaries
```

1. open a Web browser and navigate to `http://localhost:6006`
2. you should be able to see a plot of the train and test accuracies in TensorBoard
3. if you click on the histogram tab you'll also see some histograms of your weights, biases and the pre-activation inputs to the softmax in the final layer

Make sure you are in the same directory from which train.py is running. Don't worry if you are unable to get TensorBoard working; it is not required to complete the assignment, but it can be a useful tool to monitor training, so it is probably worth your while becoming familiar with it. Click here for more information:

Submission

You can test your code by typing

```
python train.py
```

Once submissions are open, you should submit by typing

```
give cs9444 hw1 hw1.py
```

When you submit, you will see some feedback for Stages 1 and 2. This is a series of Unit-tests that verify we can train your implementation. Passing all Part 1 tests on submission means you will receive all 3 marks, we will not use any additional tests for this part. For Part 2, clearing the unit-tests simply means you model is able to be evaluated, not that it has been. These tests are intended to be checks only, however you may find the error messages somewhat informative. You can make use of these unittests to check that you are structuring your code correctly.

You can submit as many times as you like - later submissions will overwrite earlier ones. You can check that your submission has been received by using the following command:

```
9444 classrun -check
```

The submission deadline is Sunday 26 August, 23:59.
15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

Additional information may be found in the FAQ and will be considered as part of the specification for the project. You should check this page regularly.

Note that the tests run at submission time will be using TensorFlow 1.3. Since we are using only the core API, this should not cause any problems. However, if you believe you are getting an error due to version incompatibility, or if you feel there is genuine ambiguity or an error in the specification or provided code, you can post to the Forums on the course Web page. Due to the higher than expected enrollments for this course, please only post queries after you have made a reasonable effort to resolve the problem yourself. If you have a generic request for help you should attend one of the lab consultation sessions during the week.

This assignment will be marked on functionality in the first instance. You should always adhere to good coding practices and style. In general, a program that attempts a substantial part of the job but does that part correctly will receive more marks than one attempting to do the entire job but with many errors.

Plagiarism Policy

Group submissions will not be allowed for this assignment. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise and serious penalties will be applied, particularly in the case of repeat offences.

DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE

Please refer to the UNSW Policy on Academic Integrity and Plagiarism if you require further clarification on this matter.

Good luck!