COMP9444 Neural Networks and Deep Learning

Session 2, 2017

Project 1 - Digit Recognition

Due: Sunday 3 September, 23:59 pm
Marks: 10% of final assessment

Introduction

In this assignment, 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.

Preliminaries

Before commencing this assignment, you should download and install TensorFlow, and the appropriate python version. It is also helpful to complete the 'MNIST for beginners' tutorial located on the TensorFlow website https://www.tensorflow.org/get_started/mnist/beginners

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.

Getting Started

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`

Now run `train.py` by typing

```
python3 train.py
```

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.)

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.

The file `train.py` calls functions defined in `hw1.py` and should not be modified during the course of the

assignment. A submission that does not run correctly when `train.py` is called will lose marks. The only situation where you should modify `train.py` is when you need to switch between different network architectures. This can be done by setting the global variable on line 7:

```
network = "none"
```

to any of the following values:

```
network = "onelayer"
network = "twolayer"
network = "conv"
```

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.

Stage 0: Provided Code

The functions `input_placeholder()` and `target_placeholder()` specify the inputs and outpus 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 `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 (3 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

Change line 7 of `train.py` to `network = "onelayer"` and test your network on the MNIST dataset by typing

```
python3 train.py
```

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 (3 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

Change line 7 of `train.py` to `network = "twolayer"` and test your network on the MNIST dataset by typing

```
python3 train.py
```

Note: if you are using the CSE Lab machines and are running out of memory, you may like to remove the files in the `summaries` directory from previous training runs.

Stage 4: Convolutional Network (4 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

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

Change line 7 of `train.py` to `network = "conv"` and test your network on the MNIST dataset by typing

```
python3 train.py
```

It may take several minutes to run, 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.

TensorFlow provides multiple API's, at various levels of abstraction. For the specified functionality in this assignment, there are generally high level TensorFlow library calls that can be used. As we are assessing TensorFlow, functionality that is technically correct but implemented manually, using a library such as `numpy`, will fail tests. If you find yourself writing 50+ line methods, it may be a good idea to look for a simpler solution.

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:

```
python3 -m tensorflow.tensorboard --logdir=./summaries
```

Depending on your installation, the following command might also work:

```
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

```
python3 train.py
```

Once submissions are open, you should submit by typing

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

(Your file can have any name, but must end in `.py`)

When you submit, you will see some feedback for Stage 1, which you can use 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 3 September, 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.

Questions relating to the project can also be posted to the Forums on the course Web page.

If you have a question that has not already been answered on the FAQ or the Forums, you can email it to `alex.long@student.unsw.edu.au`

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. 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 Honesty and Plagiarism if you require further clarification on this matter.

Good luck!
