COMP9444 Neural Networks and Deep Learning

Session 2, 2018

Project 2 – Recurrent Networks and Sentiment Classification

Due: Sunday 23 September, 23:59 pm
Marks: 15% of final assessment

Introduction

You should now have a good understanding of the internal dynamics of TensorFlow and how to implement, train and test various network architectures. In this assignment we will develop a classifier able to detect the sentiment of movie reviews. Sentiment classification is an active area of research. Aside from improving performance of systems like Siri and Cortana, sentiment analysis is very actively utilized in the finance industry, where sentiment is required for automated trading on news snippits and press releases.

Preliminaries

Before commencing this assignment, you should download and install TensorFlow, and the appropriate Python version. It is also helpful to have completed the Word Embeddings and Recurrent Neural Networks tutorials located on the TensorFlow website.

You will need to run TensorFlow 1.9 for this assignment. The best option is to run it on your own laptop, either in CPU or GPU mode. We expect that most students will be using CPU mode, and the assignment can definitely be completed in this mode. However, the training will run significantly faster if you have an appropriate GPU in your machine, and you manage to configure TensorFlow to make use of it. Instructions for setting up TF-GPU can be found here, and there is a useful forum post here.

Another option is to make use of google colabs. This will allow you to run jupyter notebooks online, but with the ability to hook into a K80 GPU.

If you are physically sitting at a CSE lab machine, you can access TensorFlow 1.9 after typing "run64". Note however that this may run quite slowly (because it is using a 64-bit emulator and in CPU mode).

Provided Files

Once you are ready to begin, download the necessary files in Assignment2.zip

Unzip this archive by typing

```
unzip Assignment2.zip
```

You should then see the following files:

```
data/
```

	Directory containing the training and evaluation datasets.
`implementation.py`	This is a skeleton file for your code. The assignment should be completed by modifying this file.
`runner.py`	This file is a wrapper around implementation.py and contains a large amount of pre-implemented functionality. An unedited version of this file must be used to generate your final model.
`glove.6B.50d.txt`	This is a text file containing the embedding vectors used in this assignment. Word embeddings have been shown to improve the performance of many NLP models by converting words from character arrays to vectors that contain semantic information of the word itself. In this assignment, we use GloVe embeddings, you can read more about them here.

Dataset

The training dataset contains a series of movie reviews scraped from the IMDB website. There are no more than 30 reviews for any one specific movie. The "data" directory contains two sub-directories, "train" and "validate". Each of these contains two sub-directories, "pos" and "neg". These directories contain the raw reviews in plain text form. The "train" directory contains 12500 positive and 12500 negative reviews; the "validate" directory contains 1000 positive and 1000 negative reviews. Each review is confined to the first line of its associated text file, with no line breaks.

For evaluation, we will run your model against a third dataset "test" that has not been made available to you. If contains additional reviews in the same format. For this reason you should be very careful to avoid overfitting – your model could report 100% training accuracy but completely fail on unseen reviews. There are various ways to prevent this such as judicious use of dropout, splitting the data into a training and validation set, etc.

Groups

This assignment may be done individually, or in groups of two students. Details about how to form groups will be posted shortly on the FAQ

Tasks

While the original, unchanged version of `runner.py` must be used for your final submission, you are encouraged to make changes to this file during development. You may want to track additional tensors in tensorboard, or serialize training data so that the word embedding is not called on each run.

The code to load the review files has already been written for you. However it is essential to perform some level of preprocessing on this text prior to feeding it into your model. Because the GloVe embeddings are all in lowercase, you should convert all reviews to lowercase, and also strip punctuation. You may want to do additional preprocessing by stripping out unnecessary words etc. You should examine any avenue you can think of to reduce superfluous data that you will feed into your model.

Attempt to train a model by running `python runner.py train`. This will create local copies of the data and embedding structures for faster subsequent loading, but will ultimately fail due to `implementation.py` being empty. The goal of this assignment is to add code to `implementation.py` in order to train and submit a neural network capable of classifying the sentiment of the provided reviews with a high level of accuracy. In this assignment, unlike assignment 1, the network structure is not specified, and you will be assessed based on the performance of your final classifier. There are very few constraints on your model – however for this assignment you may **NOT** make calls to the following parts of the API;

- `tf.contrib.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 result in us not being able to run your model 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.

During testing, we will load your saved network from a TensorFlow checkpoint (see: the TensorFlow programmers guide to saved models). To allow our test code to find the correct path of the graph to connect to, the following naming requirements must be implemented.

1. Input placeholder: `name="input_data"`
2. labels placeholder: `name="labels"`
3. accuracy tensor: `name="accuracy"`
4. loss tensor: `name="loss"`

If your code does not meet these requirements it cannot be marked and will be recorded as incomplete. You can verify your naming is correct by running `runner.py` in eval mode. If it succeeds it was able to find all relevant tensors.

Assessment

This assignment will be marked on functionality in the first instance. This table gives a general indication of what kind of mark you can expect, based on the model accuracy achieved for the withheld test set:

Accuracy	Mark
0.6	8 marks
0.65	9 marks
0.7	10 marks
0.75	11 marks

0.8	12 marks
0.85	14 marks
0.9	15 marks

Submissions failing to achieve 0.6 accuracy will be assessed by humans and assigned a mark between 0 and 7.

Code Overview

This section provides an outline to the structure of the provided code, however it is not exhaustive. Reading and understanding the functionality of the provided code is part of the assignment.

`runner.py`

This file allows for three modes of operation, "train", "eval" and "test". The first two can be used during development, while "test" will be used by us for marking.

"train" calls functions to load the data, and convert it to embedded form. It then trains the model defined in implementation.py, performs tensorboard logging, and saves the model to disk every 10000 iterations. These model files are saved in a created `checkpoints` directory, and should consist of a `checkpoint` file, plus three files ending in the extentions `.data-00000-of-00001`, `.index` and `.meta` It also prints loss values to stdout every 50 iterations. While an unedited version of this file must be used to train your final model, during development you are encouraged to make modifications. You may wish to display validation accuracy on your tensorboard plots, for example.

"eval" evaluates the latest model checkpoint present in the local `checkpoints` directory and prints the final accuracy to the console. You should not modify this code.

You may note that in both train and eval mode, the data is first fed through the `load_data()` method, which in turn calls the `preprocess(review)` function that you will define. This is to ensure your preprocessing is consistent across all runs. In otherwords, whatever transformations you apply to the data during training will also be applied during evaluation and testing.

Further explanation of the functionality of this file appears in comments.

`implemention.py`

This is where you should implement your solution. This file contains two functions: `preprocess()`, and `define_graph()`

`preprocess(review)` is called whenever a review is loaded from text, prior to being converted into embedded form. You can do anything here that is manipulation at a string level, e.g.

- removing stop words
- stripping/adding punctuation
- changing case

- word find/replace
- paraphrasing

Note that this shouldn't be too complex – it's main purpose is to clean data for your actual model, not to be a model in and of itself.

`define_graph()` is where you should define your model. You will need to define placeholders, for the input and labels, Note that the input is not strings of words, but the strings after the embedding lookup has been applied (i.e. arrays of floats). To ensure your model is sufficiently general (so as to achieve the best test accuracy) you should experiment with regularization techniques such as dropout. This is where you must also provide the correct names for your placeholders and variables.

There are two variables which you should experiment with changing. `BATCH_SIZE` defines the size of the batches that will be used to train the model in `runner.py` and `MAX_WORDS_IN_REVIEW` determines the maximum number for words that are considered in each sample. Both may have a significant effect on model performance.

Visualizing Progress

In addition to the output of `runner.py`, you can view the progress of your models using the tensorboard logging included in that file. To view these logs, run the following command from the source directory:

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

1. open a Web browser and navigate to `http://localhost:6006`
2. you should be able to see a plot of the loss and accuracies in TensorBoard under the "scalars" tab

Make sure you are in the same directory from which `runner.py` is running. For this assignment, tensorboard is an extremely useful tool and you should endeavor to get it running. A good resource is here for more information.

Submission

You need to submit six files:

- Your complete `implementation.py` file
- `report.pdf` A maximum single page document containing a description of the design choices you made and why.
- the (final) checkpoint files generated by an unedited `runner.py`

If these files are in a directory by themselves, you can submit by typing:

```
give cs9444 hw2 implementation.py report.pdf *.data* *.index *.meta checkpoint
```

It is your responsibility to ensure the model can be run. The easiest way to check is to ensure running `python runner.py eval` functions correctly. Models that cannot be run will receive very low marks. We will not manually review your code and attempt to correct errors.

You can submit as many times as you like – later submissions will overwrite earlier ones, and submissions by either group member will overwrite those of the other. You can check that your submission has been received by using the following command:

```
9444 classrun -check
```

The submission deadline is Sunday 23 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. You should check this page regularly.

If you believe you are getting an error due to a bug in the specification or provided code, you can post to the forums on the course Web page. 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

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!
