COMP9444 Neural Networks and Deep Learning

Session 2, 2017

Project 2 - Recurrent Networks and Sentiment Classification

Due Dates:
Stage 1: Sunday 8 October, 23:59 pm
Stage 2: Sunday 15 October, 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 andRecurrent Neural Networks tutorials located on the TensorFlow website.

You should download the following files from the directory hw2/src

`hw2.zip`	Python source code for Stage 1
`hw2sent.zip`	Python source code for Stage 2
`reviews.tar.gz`	25000 plain text movie reviews, split into positive and negative sets
`glove.6B.50d.txt`	GloVe word embeddings for Stage 2

Note that `reviews.tar.gz` should remain in its gzipped format. You may need to adjust the Preferences in your Web browser in order to prevent it from being gunzipped automatically (for example, with Mac Safari, go to Preferences, General and un-check "Open safe files after downloading").

Dataset

The training dataset contains a series of movie reviews scraped from the IMBD website. There are no more than 30 reviews for any one specific movie. You have been provided with a tarball (`reviews.tar.gz`) that contains two folders; "pos" and "neg". It contains the unchanged reviews in plain text form. Each folder contains 12500 positive and negative

reviews respectively. Each review is contained in the first line of its associated text file, with no line breaks.

You will need to extract these files, and load them into a datastructure you can feed into TensorFlow. It is essential to preform 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 unessesary words etc. You should examine any avenue you can think of to reduce superfluous data that you will feed into your model.

For the purposes of reducing training time, you MUST limit every review fed into the classifier at 40 words. This should occur after your preprocessing. The model must only accept input sequences of length 40. If a review is not 40 words it should be 0-padded. Some reviews are much longer than 40 words, but for this assignment we will assume the sentiment can be obtained from the first 40.

For evaluation, we will run your model against a test set that contains an additional 25000 reviews spit into positive and negative categories. For this reason you should be very careful about 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.

Tasks

There are two main tasks that must be implemented; word embedding and the classifier itself. Each task can be completed independently of the other, and the two tasks are to be submitted separately (as `hw2` and `hw2sent`).

Groups

This assignment may be done individually, or in groups of two students. Groups are defined by a number called hw2group. All students will initially be assigned a unique hw2group. When you decide that you want to work with a partner, send an email to `blair@cse.unsw.edu.au` indicating the names and student numbers of you and your partner, and I will modify hw2group accordingly.

Stage 1: Word Embeddings

Word embeddings have been shown to improve the performance of many NLP models by converting words from character arrays to vectors that contain semantic infomation of the word itself. In this assignment, you will implement a Continuous Bag of Words (CBOW) version of word2vec - one of the fastest and most commonly used embedding algorithms.

A good introduction to word embeddings can be found in the TensorFlow word2vec tutorial. This section of the assignment builds on that tutorial.

The aim of this task is to modify the code here so that it uses the continuous bag of words (CBOW) model instead of the skip-gram model. This should produce better embeddings, particularly when less data is available. Furthermore, implementing this change should give you a better understanding of both models, and the differences between them.

CBOW vs Skip-gram

Input-Output

The main difference between the skip-gram and CBOW model, is the way training data is presented.

With the skip-gram model, the input is the word in the middle of the context window, and the target is to predict any context word (word that is `skip_window` words to the left or the right) for the given word.

With CBOW, the input is all the words in the context besides the middle word, and the target is to predict the middle word, that was omitted from the context window.

For example, given the sentence fragment "the cat sat on the", the following training examples would be used by skip-gram, with parameters skip_window=1, num_skips=2 - in the form: [words in context window]: (input, target)

[the cat sat]: (cat, the), (cat, sat)
[cat sat on]: (sat, cat), (sat, on),
[sat on the]: (on, sat), (on, the)

While for CBOW the input-output pairs are (note that the inputs now contain more than one word):

[the cat sat]: ([the sat], cat),
[cat sat on]: ([cat on], the),
[sat on the]: ([sat the], on)

Of course, as is explained in the tutorial, the words themselves aren't actually used, but rather their (integer) index into the vocabulary (dictionary) for the task.

CBOW Input: Mean of Context Words Embeddings

In the skip-gram model there is just a single word as the input, and this word's embedding is looked up, and passed to the predictor.

In the CBOW, since there's more than one word in the context we just take the mean (average) of the embeddings for all context words (hint `tf.reduce_mean(?, axis=?)`).

Task

Download `hw2.zip` and `reviews.tar.gz` from the directory hw2/src. When unzipped, a directory `hw2` will be created with the following files:

`word2vec_fns.py`	skeleton code for your word2vec implementation
`word2vec_cbow.py`	code to train your word2vec model
`imdb_sentiment_data.py`	helper functions for loading the sentiment data, used by word2vec_cbow
`plot_embeddings.py`	to visualise embeddings

The file `word2vec_fns.py` is the one you will need to modify and submit. It contains two functions:

1. `generate_batch(...)` which is initially identical to the function in `https://github.com/tensorflow/tensorflow/blob/r1.3/tensorflow/examples/tutorials/word2vec/word2vec_basic.py`, with just one change, the `num_skips` parameter has been removed as it is not needed in the CBOW regime.
2. `get_mean_context_embeds(...)`

You are to:

1. modify `generate_batch` so batch is a vector of shape (batch_size, 2*skip_window), with each entry for the batch containing all the context words, with corresponding label being the word in the middle of the context
2. implement `get_mean_context_embeds` so that it returns `mean_context_embeds` - the mean of the embeddings for all context words for each entry in the batch, see the docstring in the function for more details.

You can run the code that does the embeddings with:

```
python3 word2vec_cbow.py
```

If this completes without error, you should see a file called `CBOW_Embeddings.npy` in the current directory, which you will need to submit along with your version of `word2vec_fns.py`

Additionally, if you run

```
python3 plot_embeddings.py
```

you should be able to see a low dimensional visualisation of the embeddings created with TSNE. Don't worry if you are unable to get the visualisation running. (Note: If you are working on the CSE Lab machines, you may need to use pip3 in order to get matplotlib installed correctly).

Hints:

1. `[i for i in range(5) if i != 2]` produces the list [0,1,3,4], something like this will be useful for extracting the CBOW input from the full `context_window`
2. `generate_batch` should be slightly simpler than the skip-gram version, since only one (input, output) pair needs to be created for each context window.
3. `get_mean_context_embeds(...)` should only require about 2 lines of TensorFlow code.

Submitting Stage 1

You should submit Stage 1 by typing:

```
give cs9444 hw2 word2vec_fns.py CBOW_Embeddings.npy
```

The submission deadline for Stage 1 is Sunday 8 October.

Stage 2: Sentiment Classifier

Download `hw2sent.zip` from the `hw2` directory and unzip it to produce a directory `hw2sent` containing these files:

`implementation.py`	this is a skeleton file for your RNN classifier implementation
`train.py`	file that calls implementation.py and trains your sentiment model

If you are running on your own machine, you will also need to download the file `glove.6B.50d.txt.gz` and gunzip it. If you are running on the Lab machines, you could use the copy of `glove.6B.50d.txt` in the class account by uncommenting this line in `implementation.py` (and commenting out the line above it)

```
data =
open("/home/cs9444/public_html/17s2/hw2/glove.6B.50d.txt",'r',encoding="utf-8")
```

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 - it must only use some form of recurrent unit (tanh, LSTM, GRU etc.) and be trained by the code provided.

So that no one has an advantage due to access to better hardware, we have provided the `train.py` file. On submission, you should train your model using an unedited version of train.py (this ensures the same number of data presentations for everyone's model). While an unchanged file must be used during 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. It would also be highly advisable to split the data into a training and validation set to ensure your model does not overfit.

You must make use of recurrent network elements in your final solution. Aside from the fact that this is the type of network this assessment aims to assess, for text classification

some recurrency will be important. Consider the review fragment; "I really thought this was a great example of how not to make a movie.". A naive classifier (e.g. a feed forward network trained on word counts) would be unable to correctly identify the sentiment as it depends on the tail end of the review being understood in the context of the "not" negation. Recurrent units allow us to preserve this dependency as we parse the review.

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.

Code Structure

`train.py`

This file contains the training code for the model to be defined in `implementation.py`
It calls functions to load the data, convert it to embedded form, and define the model structure. It then runs 100000 training iterations, with whatever batch size is defined in `implementation.py`.
Accuracy and loss values are printed to the terminal every 50 iterations, and are also saved as tensorboard summaries in a created `tensorboard` directory. The model is saved 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`.

This is the model that must be submitted, and will be used for marking.

`implemention.py`

This is where you should implement your solution. This file must contain three functions: `load_glove_embeddings()`, `load_data()` and `define_graph()`

`load_glove_embeddings()` should load the word embedding vectors found in `glove.6B.50d.txt` and return the vectors in array form, and a dictionary matching words to index's in that array. The array should contain one vector per row, and the dictionary should have words in string form as keys, and index's as values. For example, in the provided file, the first word vector is for "the". On loading this first vector, the values themselves (0.418 0.24968 -0.41242 0.1217 etc.) should be put in the first row of

the array. A new entry in the dictionary should then be added: {"the":0}. This tells us that the vector values for the word "the" are located in the first row of our embeddings array. There should also be a 0 vector as the first entry, with it's associated key being 'UNK' - this will be used for unknown words.

`load_data(glove_dict)` should load the training data found in `reviews.tar.gz` into a numpy array that can be used for training. Reviews should take up one row of the array, which must be capped at 40 colums. Each element should contain the index of the word from that review in the generated embeddings array. If reviews are shorter than 40 words they should be 0-padded.

`define_graph(glove_embeddings_arr, batch_size)` This is where you should define your model. The embedding array will be required to perform an embedding lookup (`tf.nn.embedding_lookup()`). You are required to make use of at least one recurrent unit - either an LSTM, GRU or tanh. 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 is also a global variable, `batch_size` which you should experiment with changing. This defines the size of the batches that will be used to train the model in `train.py` and may have a significant effect on model performance.

Visualizing Your Progress

In addition to the output of `train.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 src directory:

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

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

```
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 `train.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 should submit Stage 1 by typing

```
give cs9444 hw2 word2vec_fns.py CBOW_Embeddings.npy
```

For Stage 2, you need to submit four files:

- Your complete `implementation.py` file
- the (final) checkpoint files generated by `train.py`

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

```
give cs9444 hw2sent implementation.py *.data* *.index *.meta
```

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 for Stage 1 is Sunday 8 October, 23:59.
The submission deadline for Stage 2 is Sunday 15 October, 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 MessageBoard, 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

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!