COMP9444 Neural Networks and Deep Learning

Term 3, 2019

Project 2 - Recurrent Networks and Sentiment Classification

Due: Sunday 24 November, 23:59 pm
Marks: 24% of final assessment

NOTE: READ THIS DOCUMENT IN ITS ENTIRELY PRIOR TO STARTING THE ASSINGNMENT.

This assignment is divided into three parts:

- Part 1 contains simple PyTorch questions focused on reccurent neural networks designed to get you started and familiar with this part of the library.
- Part 2 involves creating specific recurrent network structures in order to detect if a movie review is positive or negative in sentiment.
- Part 3 is an unrestricted task where marks will be assigned primarily on final accuracy, and you may implement any network structure you choose.

Provided Files

Copy the archive `hw2.zip` into your own filespace and unzip it. This should create an hw2 directory with three skeleton files `part1.py`, `part2.py` and `part3.py` as well as two subdirectories: `data` and `.vector_cache`

Your task is to complete the skeleton files according to the specifications in this document, as well as in the comments in the files themselves. Each file contains functions or classes marked `TODO:` which correspond to the marking scheme shown below. This document contains general information for each task, with in-code comments supplying more detail. Parts 1 and 2 in this assignment are sufficiently specified to have only one correct answer (although there may be multiple ways to implement it). If you feel a requirement is not clear you may ask for additional information on the course forum.

There is also an additional file, `imdb_dataloader.py`. This is used to load the dataset provided to you in `./data` for parts 2 and 3. It will also be used in our testing. Do not modify this file.

Marking Scheme

All parts of the assignment will be automarked. Marks are assigned as follows.

Part 1:	1. [0.5]	RnnCell
	2. [0.5]	Rnn
	3. [1]	RnnSimplified
	4. [1]	Lstm
	5. [1]	Conv
Part 2	1. [3]	LSTM
	2. [3]	CNN
	3. [1]	Loss
	4. [1]	Measures
Part 3	[12]	Full Model

Similarly to the first assignment, when you submit your files through give, simple submission tests will be run to test the functionality of part 1, and to check that the code you have implemented in parts 2 and 3 is in the correct format and that we can test your models. The tests you see on submission are the only tests we will run for part 1 - so if you pass these you know you will receive full marks for part 1. After submissions

have closed, we will run the final marking scripts, which will assign marks for each task. For part 2 this will test the correctness of the networks, and for part 3 this will be inference on the full test dataset. We will not release these final tests, however you will be able to see basic information outlining which sections of code were incorrect (if you do not receive full marks) when you view your marked assignment.

Groups

This assignment may be done individually, or in groups of two students. Groups are determined by an SMS field called hw2group. Every student has initially been assigned a unique hw2group which is "h" followed by their studentID number, e.g. `h1234567`. If you plan to complete the assignment individually, you donâ€™t need to do anything (but, if you do create a group with only you as a member, thatâ€™s ok too). If you wish to form a group, go to the COMP9444 WebCMS page and click on "Groups" in the left hand column, then click "Create". Click on the menu for "Group Type" and select "hw2". After creating a group, click "Edit", search for the other member, and click "Add". WebCMS assigns a unique group ID to each group, in the form of "g" followed by six digits (e.g. g012345). We will periodically run a script to load these values into SMS.

Setting up your development environment

You should follow the instructions from Assignment 1, or use the environment you have already created there. In this assignment we will be using an additional library which you must install.

1. Activate your environment (not nessesary if you are not using virtual envs):

   ```
   conda activate COMP9444
   ```

2. Install torchtext:

   ```
   conda install torchtext
   ```

For this assignment a GPU will speed up computation, which may be helpful for part 3. For this reason you may wish to look into Google Colabs, which is a free service from google that allows development in hosted notebooks that are able to connect to GPU and TPU (Google's custom NN chip - faster than GPU's) hardware runtimes. This is not necessary to complete the assignment but some students might find it helpful.

More information and a good getting started guide is here.

It is important to note this is just an option and not something required by this course - some of the tutors are not familiar with colabs and will not be able to give troubleshooting advice for colab-specific issues. If you are in doubt, develop locally.

Part 1 [4 marks]

For Part 1 of the assignment, you should work through the file `part1.py` and complete the functions where specified.

Part 2 [8 marks]

For Part 2, you will develop several models to solve a text classification task on movie review data. The goal is to train a classifier that can correctly identify whether a review is positive or negative. The labeled data is located in `data/imdb/aclimdb` and is split into train (training) and dev (development) sets, which contain 25000 and 6248 samples respectively. For each set, the balance between positive and negative reviews is equal, so you don't need to worry about class imbalances.

You should take at least 10 minutes to manually inspect the data so as to understand what is being classified. In the entire collection, no more than 30 reviews are allowed for any given movie because reviews for the same movie tend to have correlated ratings. Further, the train and dev sets contain a disjoint set of movies, so no significant performance is obtained by memorizing movie-unique terms and their association with

observed labels. In the labeled train/dev sets, a negative review has a score <= 4 out of 10, and a positive review has a score >= 7 out of 10. Thus reviews with more neutral ratings are not included.

The provided file `part2.py` is what you need to complete. This code makes heavy use of `torchtext`, which aims to be the NLP equivelent to `torchvision`. It is advisable to develop a basic understanding of the package by skimming the documentation here, or reading the very good tutorial here.

Since this is not an NLP course, the following have already been implemented for you:

- Dataloading: a dataloader has been provided in `imdb_dataloader.py`. This will load the files into memory correctly.
- Preprocessing: review strings are converted to lower case, lengths of the reviews are calculated and added to the dataset. This allows for dynamic padding.
- Tokenization: the review strings are broken into a list of their constituent words.
- Vectorization: words are converted to vectors. Here we use 50-dimensional GloVe embeddings.
- Batching: We use the `BucketIterator()` provided by torchtext so as to create batches of similar lengths. This isn't necessary for accuracy but will speed up training since the total sequence length can be reduced for some batches.

Glove vectors are stored in the `.vector_cache` directory.

You should seek to understand the code provided as it will be a good starting point for part 3. Additionally, the code is structured to be backend-agnostic. That is, if a GPU is present, it will automatically be used, if one is not, the CPU will be used. This is the purpose of the `.to(device)` function being called on several operations.

For all tasks in this part, if arguments are not specified assume PyTorch defaults.

Task 1: LSTM Network

Implement an LSTM Network according to the function docstring. When combined with an appropriate loss function this model should achieve ~81% when run using the provided code.

Task 2: CNN Network

Implement a CNN Network according to the function docstring. When combined with an appropriate loss function this model should achieve ~82% when run using the provided code.

Task 3: Loss function

Define a loss function according to the function docstring.

Task 4: Measures

Return (in the following order), the number of true positive classifications, true negatives, false positives and false negatives. True positives are positive reviews correctly identified as positive. True negatives are negative reviews correctly identified as negative. False positives are negative reviews incorrectly identified as positive. False negatives are postitive reviews incorrectly identified as negative.

Part 3 [12 marks]

The goal of this section is to simply achieve the highest accuracy you can on a holdout test set (i.e. a section of the dataset that we do not make available to you, but will test your model against).

You may use any form of model and preprocessing you like to achieve this, provided you adhere to the constraints listed below.

The provided code `part3.py` is essentially the same as `part2.py` except that it reports the overall accuracy, and at the end of training it saves the model in a file called `model.pth` (which you will need to submit). A good starting point would be to copy the relevant sections of code from your best model for `part2.py` into `part3.py`.

Your code must be capable of handling various batch sizes. You can check this is working ok with the submission tests. The code provided in `part3.py` already does this.

You can modify and change the code however you would like, however you MUST ensure that we can load your code to test it. This is done in the following way:

1. Import and create and instance of your network from the `part3.py` file you submit.
2. Restore this network to its trained state using the `state-dict` you provide.
3. Load a test dataset, preprocessing each sample using the `text_field` you specify in your `PreProcessing` class.
4. Feed this dataset into your model and record the accuracy.

You should check the docs on the `torchtext.data.Field` class to understand what you can and canâ€™t do to the input.

Specific to preprocessing, you may add a post-processing function to the field, as long as that function is also declared in the `Preprocessing` class. You may also add a custom tokenizer, stopwords, etc. Note that none of this is necessarily required, but it is possible.

You may wish to carry out some data augmentation. This is because in practice more data will outperform a better model. Data augmentation (transforming the data you have been provided and creating a new sample with the same label) is allowed. You are allowed to modify the `main()` function to create additional data in place. You may not call any remote API's when doing this. Assume the test environment has no internet connection.

You may NOT download or load data other than what we have provided. If we find your submitted model has been trained on external data you will receive a mark of 0 for the assignment.

Marks for part 3 will be based primarily on the accuracy your model achieves on the unseen test set.

When you submit part 3, in addition to the standard checks that we can run and evaluate your model, you will also see an accuracy value. This is the result of running your model on a very small number of held-out training examples (~600). These samples can be considered representative of the final test set, however the final accuracy will be calculated from significantly more samples (~18 000). The submission test should take no longer than 10s to run.

Example of a successful submission:

```
submission_test.py::test_rnnCell PASSED
submission_test.py::test_rnn PASSED
submission_test.py::test_rnnSimplified PASSED
submission_test.py::test_lstm PASSED
submission_test.py::test_conv PASSED
submission_test.py::test_part2_networks PASSED
submission_test.py::test_measures PASSED
submission_test.py::test_part3
Importing your code..
Using device: cuda:0
Loading model.pth ..
Loading Vocab objects..
Loading submission test samples..
Loaded 652 samples
Evaluating model..
Submission accuracy = 89.74%
PASSED
```

Constraints

1. Saved model state-dict must be under 5MB and you cannot load external assets in the network class
2. Model must be defined in a class named `network`.
3. The save file you submit must be generated by the `part3.py` file you submit.
4. Must use 50d GloVe Vectors for vectorization. This means no pretraining. We are solely interested in the problem as a classification task, so trying to use something like BERT or GPT-2 is not relevant.
5. While you may train on a GPU, you must ensure your model is able to be evaluated (i.e. perform inference) on a CPU.

Common Questions:

- **Can I train on the full dataset if I find it?** No. We are aware that it is possible to obtain the full IMDB dataset. For this reason we will be automatically searching code for the loading of external assets. If this is found you will receive 0. In addition we will retrain a random selection of submissions, and those achieving high accuracy. If we find the code used for training does not match the model output you will receive a mark of 0.

- **Can I train on the dev set?** Yes, you should use the dev set during development to guide your architectural choices, however more data will almost always help a model, so prior to submission it would be a good idea to train on all labeled data provided.

- **Can I use different word vectors?** No.

- **My model is only slightly larger than 5MB, can you still accept it?** No, the 5MB limit is part of the assignment spec and changes the way to approach the problem compared to if there was no limit.

- **Can we assume you will call `model.eval()` on our model prior to testing?** Yes.

- **Can we assume a max length on the reviews?** No. But nothing will be significantly longer than what is present in the test and dev sets.

General Advice:

- You have been provided only rudimentary skeleton code that saves your model and prints the loss and accuracy at various inputs. You will almost certainly need to expand on this code so as to have a clearer understanding of what your model is doing.

- If you find your local accuracy is high, but the submission accuracy is low, you are overfitting to your local data.

- When doing a project like this, the effect of tooling is generally underestimated. You will need a system to log your experimental results while developing. One way to do this is a simple document with loss curves matched to hyperparameters and a git commit tag. There are more sophisticated systems such as sacred or tensorboard that you might want to look into as well.

- Blindly modifying code, looking at the output, then modifying again will have you going in circles very quickly. Decide on a hypothesis you want to test, then do so and record the result. Then move onto the next idea.

- You should consider the test script to be the final arbiter with regards to whether a certain approach is valid. If you do it, and the submission test runs and you get a good accuracy then the approach is valid. If it causes errors then it is not valid.

Submission

You can test your code by typing

```
python3 part2.py
python3 part3.py
```

You should submit by typing

```
give cs9444 hw2 part1.py part2.py part3.py model.pth
```

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

```
9444 classrun -check
```

The submission deadline is Sunday 24 November, 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.

Final Notes

1. Similarly to Assignment 1, we will be using PyTest to automatically grade submissions.

2. For part 2, you can pass the submission test and still have a very incorrect model. These tests just check if we can run them. You should rigorously test your code based on the specifications listed here, as well as within the provided file.

3. Ensure that you are passing submission tests early, because if a submission cannot be run, it will receive 0 marks for that part. There will be no special consideration given in these cases. Automated testing marks are final. "I uploaded the wrong version at the last minute" is not a valid excuse for a remark. For this reason, ensure you are in the process of uploading your solution at least 2 hours before the deadline. Do not leave this assignment to the last minute, as it is likely that close to the deadline, the wait time on submission test results will increase.

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!
