Coursework 1: Question classification
COMP61332 Text Mining

1. Introduction

This coursework is a group project. Your task is to build two question classifiers using (i) bag-of-words and (ii) BiLSTM, which will have been covered in Weeks 3 and 4 of COMP61332, respectively.
Input: a question (e.g., "When was the first Wall Street Journal published ?")
Output: one of *N* predefined classes (e.g., NUM:date, which is the class label for questions that require an answer of date)

Intended Learning Outcomes

- to develop deep learning-based sentence classifiers using word embeddings and BiLSTM
- to evaluate and analyse your sentence classifiers according to different settings, such as different word embeddings, and different parameters to fine-tune the models
- to discuss your methods and results in the form of academic writing
- to act as a responsible member of a team, communicate with team mates, and contribute to the team's self-organisation, planning and conflict resolution for the duration of the group work

2. Instructions

Your implementation has to be in **python3**, using **PyTorch** (https://pytorch.org/). If you are not familiar with PyTorch, check out some tutorials first (e.g., this article and its Pytorch tutorials 1, 2, and 3 at the bottom of the page, or this pytorch tutorial).

2.1 Data

In this coursework, we will make use of the dataset from https://cogcomp.seas.upenn.edu/Data/QA/QC/ (Training set 5, which contains 5500 labelled questions). Because there is no development set, you have to randomly split the training set into 10 portions. 9 portions are for training, and the other is for development, which will be used for early stopping or hyperparameter tuning. The test set will be the TREC 10 questions available from the same site.

找第一个whitespace test set用网站上的

2.2 Word embeddings

You are required to implement two kinds of word embeddings.

```
TEXT.build_vocab(train_data)
vocab_size = len(TEXT.vocab)
embedding_vectors = torch.FloatTensor(np.random.rand(vocab_size,embedding_length))
word_embeddings = nn.Embedding(vocab_size, embedding_length)
word_embeddings.weight = nn.Parameter(embedding_vectors, requires_grad=True)
```

(1) *Randomly initialised word embeddings*. For example, to build a vocabulary, you can select those words appearing at least *k* times in the training set.

(2) *Pre-trained word embeddings* such as word2vec (see this relevant video) or GloVe.

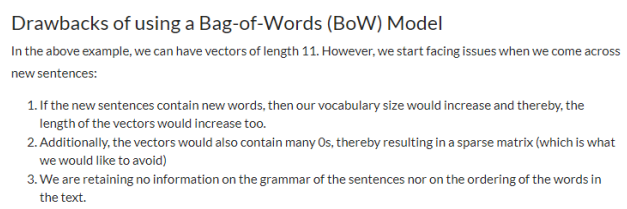
Adam Optimiser

Or BertAdam

Please note that your implementation should have an option to freeze or to fine-tune the pre-trained word embeddings during training. (*Although those options are applicable to randomly initialised word embeddings as well, you are not required to do so.*)

For preprocessing, you can ignore stop-words (e.g., "up", "a"), or lowercase all words (e.g., "When" becomes "when"). Do not forget to handle words that are not in the vocabulary!

自己定义stopwords的directory



2.3 Sentence representations

2.3.1 Bag-of-words (BOW)

A bag-of-words is a set of words (we can ignore word frequency here). For instance, the bag-of-words of the question above is

```
bow("When was the first Wall Street Journal published ?") =
{"When", "was", "the", "first", "Wall", "Street", "Journal", "published"}
```

We can define a vector representing a bag-of-words as:

$$vec_{bow}(s) = \frac{1}{|bow(s)|} \sum_{w \in bow(s)} vec(w)$$

where *s* is a sentence/question and $vec_{bow}(s)$ is the vector representation for *s*. *vec(w)* is the vector representation for word *w*.

For example:

```
vec("When was the first Wall Street Journal published ?") =
1/8 * (vec("When") + vec("was") + … + vec("published"))
```

2.3.2 BiLSTM

Please check this tutorial for using LSTM. You just need to do an extra step to replace LSTM by BiLSTM. Let's denote the vector representation for a sentence *s* produced by BiLSTM as:

$$vec_{bilstm}(s) = BiLSTM(s)$$

where *BiLSTM(s)* is the vector returned by your implementation of BiLSTM.

2.4 Classifier

Given $vec_{bow}(s)$ or $vec_{bilstm}(s)$ above, you will use a feed-forward neural network with a softmax output layer for classification. This feed-forward neural network for the classification task is presented in this video in Week 3.

2.5 Classifier (plus)

You can build more sophisticated classifiers, by

- combining $vec_{bow}(s)$ and $vec_{bilstm}(s)$ into one vector vec(s), and/or
- combining several classifiers (i.e., ensemble).

2.6 Interface

Your main source code should be in a file named `question_classifier.py`

We should be able to run your code to train a model by issuing the following command:

```
% python3 question_classifier.py --train --config [configuration_file_path]
```

We should also be able to run your code to test a model by issuing the following command:

```
% python3 question_classifier.py --test --config [configuration_file_path]
```

The program will load a configuration file storing all needed information in different sections, such as:

```
# Paths To Datasets And Evaluation
path_train : ../data/train.txt
path_dev : ../data/dev.txt
path_test : ../data/test.txt
```

```
# Options for model
model : bow  # bow, bilstm, bow_ensemble, bilstm_ensemble...
path_model : ../data/model.bow

# Early Stopping
early_stopping : 50

# Model Settings
epoch : 10
lowercase : false

# Using pre-trained Embeddings
path_pre_emb : ../data/glove.txt

# Network Structure
word_embedding_dim : 200
batch_size : 20

# Hyperparameters
lr_param : 0.0001

# Evaluation
path_eval_result : ../data/output.txt

# Ensemble model
model : bilstm_ensemble
ensemble_size : 5
path_model : ../data/model.bilstm_ensemble
```

Notes:
- If your code supports more than two required models (as mentioned in Section 2.5), such as an ensemble of 5 BiLSTM models, your configuration file may include the '`Ensemble model`' section as in the above example. In that case, the five BiLSTM models will be stored in

```
../data/model.bilstm_ensemble.0
../data/model.bilstm_ensemble.1
...
../data/model.bilstm_ensemble.4
```

- Output (e.g., `../data/output.txt`) is a file in which each line is a class for each testing question and the performance (i.e., accuracy).

- You may need to store some more information on the model (e.g., vocabulary). Do not hesitate to make use of the configuration file to specify paths to any information you might wish to store.

3. Deliverables

There are two deliverables for this coursework.

3.1 Your implementation (in a zip file)

- You can use any environment/operating system during development, but note that during marking, the code that you submit will be run on the virtual machine (VM) distributed by the Department of Computer Science.
- **Only** pytorch, numpy, and python3 standard libraries are allowed. You don't need any off-the-shelf NLP libraries like NLTK, Spacy or/and sklearn for preprocessing the data. If you

want to remove stopwords, you can find a stopword list here. Exceptionally, you can use sklearn library for evaluation metrics, and other libraries for your interface. (Please check section 5). In that case, you have to put required libraries into a file named `requirements.txt`

- The implementation should come with three folders:
    - `document`: a document containing a description for each function, a README file with instructions on how to use the code
    - `data`: training, dev, test, configuration files and some extra files needed for your models (e.g., vocabulary). For each model, you need one configuration file (e.g., bow.config, bilstm.config). Please note that you should **not** include your trained models or pre-trained word embeddings in the submission.
    - `src`: your source code.

3.2 A short paper reporting results

This should be in the form of a research paper (3-4 pages excluding references), such as https://www.overleaf.com/latex/templates/acl-rolling-review-template/jxbhdzhmcpdm. We highly recommend you to use latex with Overleaf, where you and your team can easily collaborate in writing. The report should contain at least the following points:

- Introduction
    - What is the problem?
    - Why is it important to be able to classify questions?
    - Why is this task difficult or interesting?
- Describe your approaches, e.g.
    - How did you turn sentences into vectors?
    - What are your models?
- Describe your experiments, e.g.
    - Experiment set-up,
        - What is the data split that you used?
        - Describe your preprocessing steps (e.g. removing stopwords, lowercasing words.)
        - Specify the performance metrics that you used. You don't need to explain them in detail since those details will be covered in Week 5.
    - Results: There are 8 possible combinations of the results. ***In the paper, you should report at least 6 sets***: 2 models x 3 word embedding settings including random initialisation (in this case, the default action is to fine-tune the word embeddings), pretrained embeddings with and without fine-tuning.
    - Ablation study, e.g.
        - What happens if you freeze/fine-tune the pre-trained word embeddings?
        - What happens if you use randomly initialized word embeddings instead of pre-trained word embeddings?
    - Some in-depth analyses, e.g.
        - What happens if you use only part of the training set?
        - Which classes are more difficult to classify?
        - Confusion matrix?
        - What is the effect of using other preprocessing steps?

4. Marking scheme

This coursework accounts for 25% of your final mark for COMP61332, and is worth 100 points. The following rubric will be used in marking your group project, where the first column specifies the various criteria and the second column indicates the maximum number marks your group can possibly be given.

Implementation of Question Classifiers		
Source code organisation	0	The code does not have three subfolders as required, no documentation, no readme file about how to run the code.
	2	The code only has one of the following things: (1) subfolders, (2) documentation, (3) readme file with `requirements.txt` (if any)
	5	The code is well structured and documented as required
Interface	0	The code cannot run by using the required command line
	2	The code can run by using the required command line
	5	The code allows us to input different options via configuration files
Model implementation	0	No model is implemented
	5	Only BOW model is implemented and run without errors
	10	Two models (BOW and BiLSTM) are implemented and run without errors
Training BOW model	0	After 10 epochs, the accuracy is less than 40%
	5	After 10 epochs, the accuracy is from 40%-50%
	10	After 10 epochs, the accuracy is more than 50%
Training BiLSTM model	0	After 10 epochs, the accuracy is less than 40%
	5	After 10 epochs, the accuracy is from 40%-50%
	10	After 10 epochs, the accuracy is more than 50%
Word embedding options	0	The yielded training losses and accuracies are not different with different options
	5	The yielded training losses and accuracies are different with different options
Freeze/fine-tune pre-trained embeddings	0	The yielded training losses and accuracies are not different with different options
	5	The yielded training losses and accuracies are different with different options
Bonus(*)	10	Extra model(s) is implemented and run properly
Short Paper		
Academic writing	0	The short paper looks like a technical report/user manual than a research paper
	5	The short paper was written for an academic audience. However there are some points that were not clearly presented, or it seemed like the discussion lacks originality/argumentation.
	10	The short paper was written for an academic audience and can potentially be published in a research workshop or symposium. Ideas were presented in a clear and well-argued manner.

Background and introduction	0	The paper does not provide an introduction to the task of question classification (e.g., why it is interesting) and does not clearly present the research questions that the project seeks to answer.
	5	The paper provides an introduction to the proposed topic and presents the analytical questions that the project seeks to answer.
Methodology	0	The paper does not provide sufficient details on the group's methodology.
	5	The paper provides details on the group's methodology, including how to turn sentences into vectors and description about the BOW and BiLSTM models. However some parts need further elaboration.
	10	The paper provides sufficient and clear details on the group's methodology, including how to turn sentences into vectors and description about the BOW and BiLSTM models.
Experiments	0	The paper does not provide any details about experiments, including experimental data, preprocessing steps, and the evaluation metrics.
	2	The paper provides details about experiments, including experimental data, preprocessing steps, and the evaluation metrics. However, some parts need further elaboration.
	5	The paper provides sufficient and clear details about experiments, including experimental data, preprocessing steps, and the evaluation metrics.
Analysis and interpretation	0	Quantitative results were obtained by the implemented classifiers but were not analysed and interpreted in order to answer the research questions set out in the background and introduction section.
	5	Quantitative results obtained by the implemented classifiers were analysed in order to answer the research questions set out in the background and introduction section, but in some parts the interpretation seems exaggerated (or are not aligned with the results).
	10	Quantitative results obtained by the implemented classifiers were adequately interpreted, allowing the group to answer the research questions they set out in the background and introduction section.
Exemplification/visualisation	0	The group did not provide any examples nor make use of visualisation to evidence their analysis and interpretation of quantitative results.
	5	The group provided examples and made use of visualisation, however some of these do not support or are not aligned with their findings/interpretation.
	10	The group explained their findings and interpretation by providing supporting examples or making use of suitable visualisation.

(*) Although we offer you a bonus of 10 points, the maximum mark is still 100.

Your deliverables will be assessed based on the marking scheme above, which will lead to one overall mark (out of 100). Everyone in your group will get the same mark: one of the Learning Outcomes of this coursework is focussed on acting as a responsible team member (see Intended Learning Outcomes in Section 1), hence it is everyone's duty to ensure that tasks are delegated fairly, that there are equal contributions, and that integration goes smoothly.

In the exceptional case where one or more group members have not put in an acceptable contribution, despite the team's effort to bring them back into the team and suitable discussions of

the issues arising, we implement a grievance procedure. A group can bring forward a "case of grievance" to the COMP61332 teaching staff by providing the following pieces of evidence:

- minutes of team meetings
- a written description of the events that led to the break-down of the team work, including a description of the actions that were taken to get the team back on track.

The case should be brought forward to the teaching staff no later than one working week after the coursework deadline. The COMP61332 teaching staff will then decide whether the situation is indeed exceptional and warrants a mark re-distribution.

5. Suggestions

5.1 Embeddings

For randomly initialised word embeddings, use the class
https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html#torch.nn.Embedding

For using pre-trained word embeddings, use the function `from_pretrained` of this class:
https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html#torch.nn.Embedding

5.1.1 Handling unknown words

To handle unknown words, you can use `#UNK#` tokens. The embedding of `#UNK#` can be initialised randomly by drawing from N(0, 1).

5.1.2 Pruning pre-trained embeddings

Instead of loading the whole pre-trained word embeddings, you can remove the embeddings of those words that do not appear in the dataset. You can download glove.small.zip from the course page on Blackboard. The word embeddings file also includes an embedding for `#UNK#`.

5.1.3 Fix random seeds

To make sure the three options of word embeddings are properly implemented, we should fix random seeds by adding the following lines to the beginning of `question_classifiation.py`

```
import torch, random
torch.manual_seed(1)
random.seed(1)
```

5.3 BiLSTM using Pytorch

As aforementioned, please check this tutorial for using LSTM in Pytorch first. For BiLSTM, you just need to set the `bidirectional` parameter of LSTM to true, as explained here. For example, a tagger using BiLSTM can be coded as follows:

```
import torch
import torch.nn as nn

class BiLSTMTagger(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
```

```
        # The BiLSTM takes word embeddings as inputs, and outputs hidden states
        # with dimensionality hidden_dim.
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, bidirectional=True)

        # The linear layer that maps from hidden state space to tag space
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)
```

5.4 Evaluation metrics

For this classification task, you can use both accuracy and F1 scores. Please check this python library for more information: https://scikit-learn.org/stable/modules/model_evaluation.html. As aforementioned, you don't need to explain them in detail in your report; those details will be covered in Week 5 anyway.

5.5 Reading configuration files

There are several ways to store configuration files. One way is to store configuration information in the same structure as Windows INI files and use configparser, a python library to read the file.
For example, given this config.ini file:

```
[PATH]
path_train = ../data/train.txt
path_dev = ../data/dev.txt
path_test = ../data/test.txt
```

We can parse it as follows:

```
import configparser
config = configparser.ConfigParser()
config.sections()

config.read("config.ini")

print(config.keys())
path_train = config["PATH"]["path_train"]
path_dev = config["PATH"]["path_dev"]
path_test = config["PATH"]["path_test"]
```

Another option is to use pyyaml: https://pyyaml.org/wiki/PyYAMLDocumentation

5.6 Parsing command line arguments

To parse command line arguments, we can use argparse, a standard Python library. Following is an example of parsing arguments so that you can run your code as described in Section 2.6:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--config', type=str, required=True, help='Configuration
file')
parser.add_argument('--train', action='store_true', help='Training mode - model is
saved')
```

```
parser.add_argument('--test', action='store_true', help='Testing mode - needs a
model to load')
args = parser.parse_args()
if args.train:
   #call train function
   train(args.config)
elif args.test:
   #call test function
   test(args.config)
```

Deadline: 12:00AM (Midnight, UK time), Friday, 11th March, via Blackboard.
Good luck!