Coursework 1: Question classification

This coursework is a group (3-4 students) project. Your task is to build two question classifiers using (i) bag-of-words and (ii) BiLSTM.

- Input: a question (e.g. *"How many points make up a perfect fivepin bowling score ?"*)
- Output: one of N predefined classes (e.g. *NUM:count*)

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. https://medium.com/biaslyai/learn-pytorch-basics-6d433f186b7a, chapters 1, 2, and 3, https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html).

Data

You use the data from https://cogcomp.seas.upenn.edu/Data/QA/QC/ (Training set 5). Because there is no dev set, you will randomly split the training set into 10 portions. 9 portions are for training, and the other is for development (e.g. early stopping, hyperparameter tuning).

Word embeddings

Your implementation accepts two kinds of word embeddings.

1. You randomly initialize word embeddings. (To build a vocabulary, you can select those words appearing at least *k* times in the training set.)

2. You use pre-trained word embeddings such as word2vec (https://code.google.com/archive/p/word2vec/) or GloVe (https://nlp.stanford.edu/projects/glove/). Note: your implementation has an option to ***freeze*** or to ***fine-tune*** the pre-trained word embeddings during training.

For preprocessing, you can ignore stop-words (e.g. "up", "a"), or lowercase all words (e.g. "How" becomes "how"). *Don't forget to handle words that are **not** in the vocabulary!*

Sentence representations

Bag-of-words

1. 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("How many points…") =
```

```
{"How", "many", "points", "make", "up", "a", "perfect", "fivepint",
"bowling", "score"}
```

2. Turning a bag-of-words to a vector:

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

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

For example:

```
vec("How many points…") =
1/10 * (vec("How") + vec("many") + … + vec("score"))
```

BiLSTM

https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html is a good tutorial for using LSTM. You just need to do an extra step to replace LSTM by BiLSTM. Let's denote

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

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.

Classifier (plus)

You can build more sophisticated classifiers, by

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

Interface

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

For training, run

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

For testing, run

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

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

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

# 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
```

Note:
- If your code supports more than two required models (`bow`, `bilstm`), such as an ensemble of 5 bilstm models, your configuration file may include:

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

5 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 of the model (e.g. vocabulary). Don't hesitate to make use of the configuration file.

Deliverables

There are two deliverables for this coursework:

1. *(50 marks)* Your implementation (in a zip file). The implementation should come with three folders:
    - `document`: a document containing a description for each function, a README file instructing how to use the code, *(5 marks)*
    - `data`: training, dev, test, configuration files (excluding word embeddings), and some extra files needed for your models (e.g. vocabulary). Note: for each model, you need one configuration file (e.g. `bow.config`, `bilstm.config`)
    - `src`: your source code. *(45 marks)*

2. *(50 marks)* Short paper reporting results. This should be in the form of a research paper (2-3 pages excluding references) http://acl2020.org/downloads/acl2020-templates.zip (latex is highly recommended). The report should contain at least below points:
    - Introduction *(2 marks)*
        - What is the problem?
    - Describe your approaches, e.g. *(10 marks)*
        - How to turn sentences into vectors?
        - What are your models?
    - Describe your experiments, e.g. *(38 marks)*
        - Experiment set-up, *(2 marks)*
            - What is the used data?
            - Describe your preprocessing steps (e.g. removing stopwords, lowering words.)
            - What is the performance metric?
        - Results, *(6 marks)*
        - Ablation study, e.g. *(15 marks)*
            - What if you freeze/fine-tune the pre-trained word embeddings?
            - What if you use randomly initialized word embeddings instead of pre-trained word embeddings?
        - Some in-depth analyses, e.g. *(15 marks)*
            - What if you use only part of the training set?
            - Which classes are more difficult than the other?
            - Confusion matrix?

- What if you use other preprocessing steps?

Note

1. Source code

- The code has to be runnable on Linux/MacOS. (You can develop the code on Windows, but make sure that it can run on Linux or MacOS.)
- ***Only*** pytorch, numpy, and python3 standard libs are allowed. (You don't need any fancy libs like NLTK, Spacy or/and sklearn for preprocessing the data. If you want to remove stopwords, you can find a stopword list here https://gist.github.com/sebleier/554280)
- Don't include your trained models in the submission.
- Don't include pre-trained word embeddings in the submission.
- If none of your models work (i.e. the code is not runnable or the performance is less than 50%) , you will get **0** marks for this coursework.
- If only one model works, you will get 35 marks maximum for the implementation/report (so 70 marks maximum in total).

2. Report

- The report has to include the information (name, student number…) of every member.
- Don't need to state clearly the duty of each member. Every member will get the same marks.

3. Bonus

- If you build an extra classifier (check out "classifier plus"), you will get a bonus of 10 marks. (But your total marks can't exceed 100.)

Suggestions

1. Embeddings

For randomly initialized word embeddings, use the class
https://pytorch.org/docs/stable/nn.html?highlight=embedding#torch.nn.Embedding

For using pre-trained word embeddings, use the function
https://pytorch.org/docs/stable/nn.html?highlight=embedding#torch.nn.Embedding.from_pretrained

2. Handling unknown words

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

3. Pruning pre-trained embeddings

Instead of loading whole pre-trained word embeddings, you can remove the embeddings of those words that do not appear in the dataset. You can find here (glove.small.zip) such word embeddings from glove.

The word embeddings file also includes an embedding for #UNK#.

Deadline: 17:00, Tuesday, 3rd March, via Blackboard.