COMP7606C: ASSIGNMENT #1 - QUESTIONS 2 & 3

Due date: 3/30/2020 23:59 HKT for both Question 2 and Question 3. These questions DO NOT require written answers. Students may discuss with other students, but each student must finish their programming tasks individually and must submit her/his code via moodle.
The following instructions still apply:

- Please use `Python 3 (3.5-3.7)` as the default programming language, and `Tensorflow 2.1` as the deep learning platform. Besides, the following packages may be required:
  - (required) `Numpy >= 1.17`
  - (optional) `sklearn` and `matplotlib`
- All you need to submit are the Python scripts, `q2_word2vec.py, q2_sentiment.py, and q3_ner.py`, and the `results/prediction.txt`. Please submit each file separately to the relevant submission page in Moodle and rename the files to `your_student_uid.py` (for example, `0123456789.py`).

IMPORTANT: Strictly follow the instructions, otherwise your assignment will not be appropriately graded.

1. Basic Neural Network (30 points)

Done!

2. Word2vec (30 points)

In this question we will learn how to load a pre-trained `word2vec` model, explore "math" with words, and apply the `word2vec` model to a downstream NLP task. We will use `Glove`[1] **glove.6B.50d** in this question. Please go to `Glove` website to download the pre-trained word vectors.

2.1. **Loading word vectors (5 points).** Implement `PretrainedWord2Vec._load_word2vec` in `q2_word2vec.py`.

2.2. **Getting word vector (5 points).** Implement `PretrainedWord2Vec.get_vector` in `q2_word2vec.py`. Note that **glove.6B.50d** has only 400,000 words. Words that are not in the vocabulary are called oov (out-of vocabulary) words. For oov word vectors, someone suggests to use the average of all the known word vectors to represent them. We have calculated the oov word vector, see `oov_vector` in `q2_word2vec.py`.

2.3. **Word similarity (5 points).** Implement `PretrainedWord2Vec.word_similarity` in `q2_word2vec.py`. Remember that cosine similarity is:

$$\text{sim}(\text{word}_1, \text{word}_2) = \frac{v_1^T v_2}{\|v_1\| \cdot \|v_2\|} \tag{2.1}$$

where v_1, v_2 are 50-d vectors for word_1 and word_2, respectively. For example, $\text{sim}(\text{king}, \text{queen}) = 0.7839044$.

2.4. **Finding most similar words (5 points).** Implement `PretrainedWord2Vec.most_similar` in `q2_word2vec.py`. For example, top-3 most similar words for "queen" are "princess", "lady", and "elizabeth".

2.5. **Visualizing the vectors.** See `PretrainedWord2Vec.visualization` for more information on visualizing the word vectors. There is no coding required.

2.6. **Sentiment analysis (10 points).** Now, with the pre-trained word vectors loaded, we are going to perform a simple sentiment analysis. This task is to classify movie reviews[2] into two classes, `positive` and `negative`, which are represented by 1 and 0 in the code. To predict the sentiment level of a given movie review, we are going to use the average of all the word vectors in the sentences as the feature v_s. Note that v_s has the same shape as the word vectors. Then the feature input layer is projected onto the output layer through two fully-connected layers:

$$h = \text{ReLU}(v_s \cdot W_1 + b_1) \tag{2.2}$$

$$\hat{y} = h \cdot W_2 + b_2 \tag{2.3}$$

[1] `https://nlp.stanford.edu/projects/glove/`

[2] Raw IMDB dataset:`https://ai.stanford.edu/~amaas/data/sentiment/`

where ReLU is the ReLU activation function. The loss function is `binary_crossentropy` which will be optimized by **Adam** algorithm.

2.6.1. *Constructing Tensorflow model (10 points).* Implement `create_model_and_compile` in `q2_sentiment.py`. Note that we will use `tensorflow.keras.layers.Dense` and `tensorflow.keras.Sequential` to create the fully connected layers, and `tf.keras.optimizers` to design the optimizer.

2.6.2. *Model training.* We provide processed IMDB dataset under `datasets/imdb/imdb.npz`. Dataset can be loaded with `datasets.imdb_utils.ImdbDataset` class. The `DataProcessor` in `q2_sentiment.py` transforms sentence into sentence vector. Run `python3 q2_sentiment.py` to monitor the training & validation losses and accuracies. Fine tune `learning_rate` to see its impact on performance.

3. Named entity recognition (40 points)

In this question, we will build a sequence-to-sequence (seq2seq) model for named entity recognition (NER). NER is a sub-task of information extraction that seeks to locate and classify each named entity into categories such as person names, organizations, locations, quantities, percentage, monetary values, etc. In this example, we only consider four categories:

- Person (PER): e.g., "Mark Gleeson", "Andre Caboche", etc
- Organization (ORG): e.g., "Reuters", "American Airlines", etc
- Location (LOC): e.g., "Netherlands", "Czech Republic", etc.
- Miscellaneous (MISC) "Japanese", "USD", etc

If a word is not considered a named entity, a null-class (O) will be used to label it. Therefore, the problem will be formulated as a five-class classification problem.

The seq2seq model for NER accepts a sequence of word indexes and produces a sequence of labels which represent the named entities of the words in the context. We are going to build a shallow seq2seq model, which consists of an embedding layer, an LSTM, and a fully-connected feedforward layer.

3.1. **Encoder layer (10 points).** The encoder is comprised of one embedding layer and one LSTM layer. Implement `Encoder` in `q3_ner.py`. Hint: use `tf.keras.layers.Embedding` for embedding layer and `tf.keras.layers.LSTM` for LSTM layer.

3.2. **Fully-connected feedforward layer (10 points).** Implement `FFC` in `q3_ner.py`.

3.3. **Masking and padding (10 points).** In order to batch process the data during training and prediction, same length sequences will be fed into the seq2seq model. Unfortunately, each sentence can be arbitrarily long in the dataset. The most common approach to address this problem is to *pad* the sequences with zeros. Suppose the longest sentence in our dataset has M words, then for each input of length T,

(1) Add "0-vectors" to $\boldsymbol{x}, \boldsymbol{y}$ to make them M words long;
(2) Create a *masking vector* $[m^{(t)}]_1^M$ which is 1 for all $t <= T$ and 0 for all $t > T$;[3]
(3) Modify our loss function such that the *padding* does not affect the loss and gradient computations.

$$J = \sum_t^M m^{(t)} \text{CE}(\hat{y}^{(t)}, y^{(t)}) \tag{3.1}$$

where CE represents cross-entropy implemented with tensorflow: `SparseCategoricalCrossentropy`

Implement masked loss function Equation 3.1 `MaskedLoss` in `q3_ner.py`.

3.4. **Model training.** We provide dataset from CONLL2003 shared task under `datasets/conll2003`. See here[4] for more information about the task. Class `datasets.conll_utils.ConllDataset` converts the words to word indexes, and also pads all the sequences into same length. Run `python3 q3_ner.py` and monitor the training and validation loss. The model checkpoints will be saved in `ner_checkpoints` folder. Of course you can fine-tune `n_epochs` to control how long the training takes, `batch_size` to fit to your computer memory, and `leaning_rate` for better results.

[3] In our code, we will actually use Boolean values True and False instead of 1 and 0 for computational efficiency.
[4] `https://www.aclweb.org/anthology/W03-0419.pdf`

3.5. **Making prediction (10 points).** The test data `datasets/conll2003/test.masked.txt` contains labels all masked to null-class (O). After the training completes, prediction will be made based on the latest checkpoint you save. Predicted file will be saved at `results/prediction.txt`, which you shall submit along with your code. Check that file to see how your trained model make predictions. Do not expect too high on the results, because the model you just create is not "deep" enough. At the moment we will not distribute the true test data for verification. Your result will be graded using $F1$ score in the end. We also encourage you to improve the result with your model. If you intend to try your own method, please do create another scripts and do not pollute the original scripts. Extra scripts are not necessary to be submitted, but `results/prediction.txt` is required.