Assignment 2
Word Embedding, FeedForward Neural Network and RNNs

STAT8021: Big Data Analytics (Spring 2025)
STAT8307: Natural Language Processing and Text Analysis (Spring 2025)

due: **March 23, 2025, Sunday, 11:59 PM**

Goal

The main goal of this assignment is for you to get a solid understanding of word embedding, feedforward neural networks and Recurrent Neural Network (RNN). In the first part, you will compute word embeddings in a toy example. In the second, you will play around with feedforward neural networks in PyTorch and use them to solve the sentiment classification problem from Assignment 1. In the third part, you will implement an RNN model for character-level classification. In the fourth part, there is a bonus question for the sentiment classification problem. **Note that you should use Python 3.8+ and PyTorch to finish this assignment. Since this assignment does not involve too much computation, you can use the CPU mode of PyTorch and run the experiments on your own computer. Otherwise, you can use the GPU from Google Colab at https://colab.research.google.com/.**

Submission

Please submit the following **two files** to Moodle for grading:

- A PDF report of your detailed answers to all the questions.
- A ZIP file of two folders:
  - Folder `code_Q2` containing: `models.py`.
  - Folder `code_Q3` containing: `dataset.py`, `models.py`, and `train_cls.py`.

(Optional) If you have finished the bonus question, please submit the Python file of your improved model and the generated `test_predictions.csv` file under folder `code_Q2`.

The total score of this assignment is **100 points**. The bonus score is **10 points**. Please write your procedures in the PDF report if you do not completely finish the code.

Part 1: Word Embedding

In this part, you will investigate word embeddings in a toy example. **Note that you don't need to write code in this part. Please write the calculation steps and answers in the report.**

Let us create a simple corpus containing 5 sentences which state some well-known facts about a royal family:

S1	The future king is the prince.
S2	Daughter is the princess.
S3	Son is the prince.
S4	Only a man can be a king.
S5	Only a woman can be a queen.

Q1 Many word vector implementations are driven by the idea that similar words will be used in similar contexts. By examining these contexts, we can try to develop embeddings for our words. Count-based word embeddings are constructed under this idea. In this question, we will elaborate on count-based word embeddings step-by-step.

[TOTAL: 20 points]

(a) List all distinct words sorted by ascending alphabet order in this corpus, and compute the number of distinct words. **[5 points]**
Hint: You need to do text preprocessing, including tokenization, converting words into lowercase, and removing punctuations by yourselves. This sorting method is the same as the Python sorted() function.

(b) Construct the word-word co-occurrence matrix M for window-size $m = 4$ (*i.e.*, left 4 words and right 4 words). Here, we only consider the following 10 words:

"daughter", "future", "king", "man", "only", "prince", "princess", "queen", "son", "woman"

Thus, the dimension of M is 10×10. The ordering of the words in the rows/columns should be the same as the ordering of the given 10 words. **[10 points]**

(c) We use SVD (Singular Value Decomposition) to reduce the dimension of co-occurrence matrix M and assume that the computed 2D embedding of the given 10 words as follows:

daughter	$(0.00, 0.00)$
future	$(1.09, 0.20)$
king	$(1.31, -0.91)$
man	$(0.73, 0.96)$
only	$(0.40, -0.87)$
prince	$(1.25, 0.54)$
princess	$(0.00, 0.00)$
queen	$(0.09, -0.35)$
son	$(0.53, -0.29)$
women	$(0.21, 0.66)$

Now we have two new word embeddings $X = (0.20, -0.50)$, $Y = (1.00, 0.00)$, but we don't know the semantic meaning of these two words. According to the 2D word embeddings given in the above table, you are required to give the most possible semantic meanings of X and Y. **[5 points]**

Part 2: Deep Averaging Network

Deep Average Network (DAN) is a typical deep unordered model obtaining nearly state-of-the-art accuracy on sentence or document-level tasks with very little training time. Specifically, if our input is $s = (w_1, ..., w_n)$, then we use a feedforward neural network for prediction with input $\frac{1}{n}\sum_{i=1}^{n} e(w_i)$, where e is a function that maps a word w_i to its real-valued vector embedding.

In this part, you will continue working on the sentiment classification problem in Assignment 1. Please download the start code `code_Q2.zip` from Moodle for this question and install the required Python packages in `requirements.txt`. You only need to complete the file `models.py`. **Please do not change other files.**

Q2 In this question, you are required to implement the deep averaging network (DAN) with PyTorch. Note that you can call the PyTorch built-in functions. **[TOTAL: 30 points]**

(a) Please implement SGD and Adam optimizers by Pytorch. Then, you need to implement Binary Cross Entropy (BCE) loss by Pytorch. Specifically, you need to complete the `__init__` method of `DANSentimentClassifier` class in `models.py`. **[5 points]**

(b) `DeepAveragingNetwork` class in `models.py` defines the network architecture of DAN, which consists of an embedding layer and a two-layer feedforward neural network. You are recommended to use BatchNorm in the network. Note that the output dimension of the feedforward neural network is 1 for binary classification. **[5 points]**

Hint: To handle input sentences with different lengths, we have padded all sentences into the same length with a `<pad>` token. Then, we have converted each sentence into a sequence of indices. You can use `torch.nn.Embedding` to convert indices into word embeddings. Note that you should ignore `<pad>` tokens when computing the average embeddings.

(c) Complete `fit` and `predict` methods of `DANSentimentClassifier` class in `models.py`. We have provided you with the backbone code of these two functions. You may follow this backbone code or rewrite these two functions. Now, you can run the following command to test if your implementation is correct. Please copy the output of this command into the report. If your implementation is correct, you should get at least **88% accuracy** on the validation set. **[10 points]**

```
python main.py --model dan --lr 0.001
```

(d) The learning rate has a significant impact on the performance of deep learning algorithms. In this question, you are required to run the following commands to test different learning rates: $0.1, 0.01, 0.001, 0.0001, 0.00001$, and report accuracy, precision, recall and F score of each setting on the validation set. According to the results and the training time, which learning rate is a good choice? Please briefly describe the reason. (In other words, you need to explain why too large or too small learning rate doesn't perform well.) **[5 points]**

```
python main.py --model dan --lr 0.1
python main.py --model dan --lr 0.01
python main.py --model dan --lr 0.001
python main.py --model dan --lr 0.0001
python main.py --model dan --lr 0.00001
```

(e) In this question, we train word embeddings in an end-to-end manner. The dimension of word embeddings also has important effect on the classification performance. You are required to run the following commands to test different embedding dimensions: $50, 100, 400$, and report accuracy, precision, recall and F score of each setting on the validation set. According to the results and the training time, which embedding dimension is a good choice? Please briefly describe the reason. (In other words, you need to explain why too large or too small embedding dimension doesn't perform well.) **[5 points]**

```
python main.py --model dan --lr 0.001 --embed_dim 50
python main.py --model dan --lr 0.001 --embed_dim 100
python main.py --model dan --lr 0.001 --embed_dim 400
```

Part 3: Character Language Modeling with RNNs

In this part, you will implement a character-level RNN language model. The dataset in this part is part of WikiText2. Only 27 character types (lower-case English characters and spaces) are present. You will do a simplified version of the language modelling task: binary classification of fixed-length (20 characters) sequences to predict whether the given sequence is followed by a consonant or a vowel (0 for consonant, 1 for vowel). Please download the start code `code_Q3.zip` for this question from Moodle.

Q3 Implement a character-level RNN classifier to predict whether the given sequence is followed by a consonant or a vowel. **[TOTAL: 40 points]**

(a) The inputs to the classifier should be sequences of character indices. To convert raw input string into character indices, you are required to complete the class `Vocab` in `dataset.py`. You should create a vocabulary which contains lower-case English characters (a-z) and the space, with corresponding indices from 0 to 26. Also, you need to implement three functions: `index_of` converts an input character into its index; `object_of` converts an input index into a corresponding character; `__len__` returns the size of your vocabulary. **[10 points]**

(b) Define a neural network to do this classification. Please complete the `RNNClsModel` class in `models.py`. You should embed the input character indices using a `nn.Embedding` layer and then feed resulting embedding tensors into an RNN. After that, you can use RNN's output feature of **the last timestamp** as the input of a two-layer feed-forward network, and obtain the probability of the next letter is a vowel. **Note that you are recommended**

to use nn.LSTM to implement the RNN while you are free to tune the architecture of the network. Please describe your model architecture in the report. **[10 points]**

(c) Complete the function `train_rnn_cls` in `train_cls.py`. We have provided you with the backbone code. You may follow this backbone code or rewrite the whole function. Please run the following command to test your implementation. **You are free to tune hyperparameters, including learning rate, batch size, epochs, etc.** Please copy your best output and corresponding figure `rnn_cls.jpg` (the figure will be generated after running the following command) into your report and describe the adopted hyperparameters. **[20 points]**

```
python train_cls.py
```

To receive full credit of Q3, you need to get at least 80% **accuracy on the validation set**[1]**.** In your report, you should: (1) Describe your model and implementation and (2) Report accuracy results and timing information. Even if you are not able to get this part fully working, write up and document as much as you can so we can give appropriate partial credit.

Part 4: Bonus

In this part, we encourage you to further improve this Deep Averaging Network (DAN) or use other neural networks for sentiment classification in **Q2**. Please don't use advanced language models like Transformers, BERT, GPT, etc. You should briefly introduce your improvements and write the accuracy of the validation set in your report. If you want to change Python files except for `models.py` under `code_Q2`, you should also submit your changed files into Moodle. Additionally, you should submit the `test_predictions.csv` file in `code_Q2/data` folder. **[10 points]**

Hint: Things you might try include but are not limited to: load pre-trained word embeddings from word2vec, try recurrent neural network (e.g., RNN, LSTM), tune hyperparameters (batch size, learning rate, architecture of MLP, optimizers, training epochs, etc.), add dropout, and add data augmentation. **Your final code here should be whatever works best, and the model should train and evaluate in a reasonable time (1-2 hours) on Colab or its machine-equivalent computer.** The bonus score you will get depends on your accuracy on the validation set and the relative performance on test data compared with other students' submissions.

PyTorch implementation and debugging tips

- You can use `torch.from_numpy` to convert `numpy.ndarray` into `torch.tensor`. For more tensor operations, please check the PyTorch document (`https://pytorch.org/docs/stable/index.html`).
- You can print training and validation loss over epochs (steps) to check if your learning process is correct. We recommend you use Tensorboard (`https://pytorch.org/docs/stable/tensorboard.html`) to visualize the training process. We have provided you with these codes for your reference.
- If you see NaNs in your loss, please check if your logarithm or division operations are valid. (There might be other reasons for this problem.)
- Google / Stack Overflow are your friends.

[1]Our reference implementation can achieve 82.0% accuracy with the default hyperparameters and an unoptimized implementation.