CS4990/CS5990 Assessed Coursework Assignment 2: Sentence Semantic Similarity Measurement

This assignment must be submitted by **12 March 2021, 10am**. An extension can only be given by the Office (not the lecturer). Find more details about the extension policy at [here].

Overview

In this assignment, you are asked to develop a model to *measure the semantic similarity between sentences*. The main challenge of this task is to create good *vector-based sentence representations*: the vector representations need to encode the salient semantic information in the sentences; once the vector representations are available, you can measure the semantic similarity between sentences by simply computing their vectors' cosine similarity. With this understanding, your task in this assignment is to develop appropriate neural models to *encode sentences* (i.e., to create vector representations for sentences).

Data

A training set (**cw2_train.csv**) is provided to help you develop the sentence encoding model. It includes over 11k data entries, each entry containing two sentences and a numeric score between 0 and 1 indicating the semantic similarity between the two sentences.

A dev set (**cw2_dev.csv**) is also provided to help you evaluate the quality of the model you have developed. Its data entries are in the same format as in the training set.

Sample Code

A sample code (**cw2_sample.ipynb**) is provided to illustrate the whole pipeline for developing an encoder model. A simple encoder model is provided: the encoder represents a sentence by first averaging its words' embeddings, passing the averaged embedding to a fully connected layer and applying a non-linear activation function to obtain the final representation. At training time, the learning objective is to minimise the *mean squared error* (MSE) between the true similarity score and the predicted similarity score (i.e., the cosine similarity between the sentence representations). The model architecture is illustrated in Fig. 1.

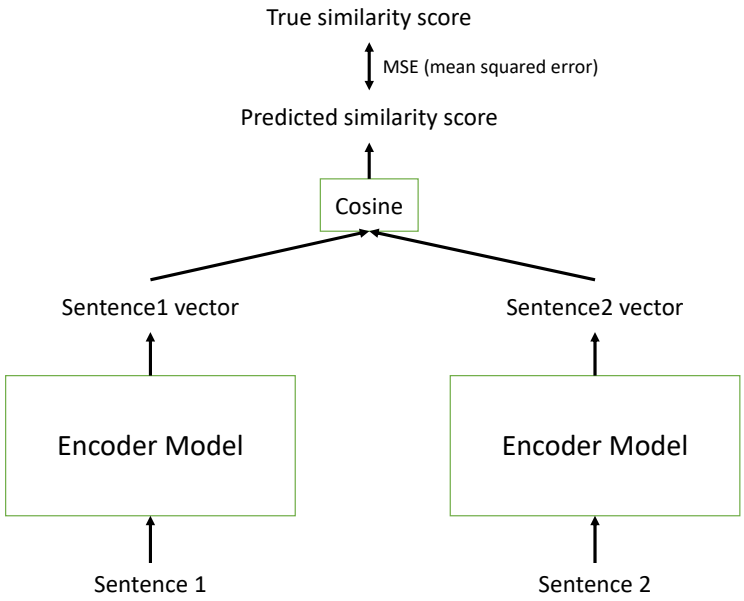

Fig.1 Model architecture. Note that the two 'Encoder Model' box in the Figure corresponds to the same encoder model. Or, in other words, they share the same parameters. Your task is to use different neural architectures to develop the encoder model and compare their performance.

Submission Requirements

Task 1: MLP-based Encoder (35 Points)

The sample code provides a simple MLP-based encoder model with one hidden layer. You are asked to implement a more complex MLP-based encoder, with at least two hidden layers. You should decide the hyper-parameters (e.g., number of layers, the dimension of each hidden layer, which activation function to use, etc.) with appropriate strategies. Deliverables:

- Report: explain how you have selected the hyper-parameters, compare the performance of different setups, and present the error analysis (i.e., inspect in which cases the model yields poor performance, and try to analyse the reason and propose solutions).
- Code: provide the implementation for the best MLP-based encoder you have developed. Name the file **test_mlp.ipynb**.
- Saved model: provide the saved model for the best MLP encoder you have developed. Name it **best_mlp.state_dict**.

Task 2: CNN-based or RNN-based Encoder (55 Points)

Develop a CNN- or RNN-based encoder model. You can use any variants of CNNs or RNNs, e.g., LSTM, or with the attention mechanism. Decide the hyper-parameters (e.g., number of filters, filter sizes, dropout or not, hidden layer size, etc.) with appropriate strategies. Deliverables:

- Report: describe which architecture you have used, explain why you choose it, discuss how you have selected the hyper-parameters, compare the performance of different setups you have tried, and present the error analysis.
- Code: provide the implementation of the best CNN/RNN encoder you have implemented. Name the file **test_cnn.ipynb** or **test_rnn.ipynb,** depending on the model you have developed.
- Saved model: provide the saved model for the best CNN/RNN encoder you have developed. Name it **best_cnn.state_dict** or **best_rnn.state_dict,** depending on the model you have developed.

Task 3 (Optional): Other Encoder Models (10 Points)

You can explore other neural architectures to develop the encoder model, e.g., BERT or GPT. Deliverables:

- Report: Describe which architecture you have used and why you choose it. Compare its performance with the MLP and CNN/RNN based models you have developed, and analyse its pros and cons.

- Code: provide the implementation of the additional model you have developed. Name the file **test_bonus.ipynb**.
- Saved model: provide the saved model for the best additional encoder you have developed. Name it **best_bonus.state_dict**

Requirements that Apply to All Three Tasks:

- Submit one report summarising your findings on all three tasks. Name it **cw2_report**
- You can use popular python libraries in your implementation, e.g., scikit-learn, nltk, pytorch. You are also allowed to adapt existing implementations published online (e.g., on GitHub) to develop your model, but in that case you must provide reference to the original work and explain what adaptations you have made.
- You can use any techniques you find appropriate to develop your model, including the ones not covered in this course. You need to explain why you choose to use them.
- In your submitted code (e.g., test_mlp.ipynb, test_rnn.ipynb), clearly specify which embedding you used (including the dimension size) and provide the link to download the embedding. Also, provide all necessary code to run your trained model. **The markers will rely on these files to evaluate your models' performance on some test data. It is your responsibility to make sure your submission can be executed; the markers will not troubleshoot your code, and any problems will result in a major loss of points.**

Submission Instructions

The coursework assignment must be completed strictly individually. The submission is entirely electronic. In order to submit, create a compressed file (zip or tar.gz) that includes all your submission files, name it *CW2.zip (or CW2.tar.gz)* and upload it on the Moodle page.

The file that you submit cannot be overwritten by anyone else, and it cannot be read by any other student. You can, however, overwrite your submission as often as you like, by resubmitting, though only the last version submitted will be kept. Submissions after the deadline will be accepted but they will be automatically recorded as late and subject to the College Regulations on late submissions. Please note that all your submissions will be graded anonymously; your name should not appear anywhere in your submission.