Homework 4 (100 points)

Sequence Models

The homework will be due in 2 weeks from the day of release.

A Recurrent Neural Network (RNN) is a neural network that can be used when your data is treated as a sequence, where the order of the data-points matter. You will use an existing RNN in part 1, then implement an RNN in part 2. In part 3, you will demonstrate the usage of any of the word-embeddings we discussed in class. Upload a .txt file with a link to your file as your submission on Submitty (You may have different links for each task).

You need to perform the following tasks for this homework (Task 1 is for Graduate (6000) level ONLY):

6000 level ONLY - Task 1 (50 points): This task involves training existing models. Download the character level RNN at https://github.com/karpathy/char-rnn

You are required to read the documentation provided in this repository and experiment with the RNN model. This is a legacy repository; therefore, one task would be to research and use a recent version. Train the model on 'tiny Shakespeare' dataset available at the same location. Create outputs of the model after training for i) 5 epochs ii) 50 epochs and iii) 500 epochs. What significant difference do you observe between the 3 outputs? Explain. Repeat the experiment with the LSTM model provided in the repository. Explain the differences and/or similarities between the results of both models.

Task 2 (50 points): In this task, you will pick a dataset (time-series or any other form of sequential data) and an associated problem that can be solved via sequence models. You must describe why you need sequence models to solve this problem. Include a link to the dataset source. Next, you should pick an RNN framework that you would use to solve this problem (This framework can be in TensorFlow, PyTorch or any other Python Package).

> **Part 1 (10 points):** Implement your RNN either using an existing framework OR you can implement your own RNN cell structure. In either case, describe the structure of your RNN and the activation functions you are using for each time step and in the output layer. Define a metric you will use to measure the performance of your model (NOTE: Performance should be measured both for the validation set and the test set).
>
> **Part 2 (30 points):** Update your network from part 1 with first an LSTM and then a GRU based cell structure (You can treat both as 2 separate implementations). Re-do the training and performance evaluation. What are the major differences you notice? Why do you think those differences exist between the 3 implementations (basic RNN, LSTM and GRU)?

Note: *In part 1 and 2, you must perform sufficient data-visualization, pre-processing and/or feature-engineering if needed. The overall performance visualization of the loss function should also be provided.*

Part 3 (10 points): Can you use the traditional feed-forward network to solve the same problem. Why or why not? (*Hint: Can time series data be converted to usual features that can be used as input to a feed-forward network?)*

Task 3 (50 points):

Part 1: Implementing Word Embeddings (10 points)

- Use a pre-trained word embedding model (Word2Vec, GloVe, FastText, or BERT embeddings).
- Provide a comparative discussion on why you chose this embedding over others.
- Load embeddings efficiently (either from pre-trained vectors or using an NLP library like Gensim, SpaCy, or Hugging Face).
- Allow dynamic user input of two words and output their respective embeddings.
- Handle cases where a word is out of vocabulary (OOV) and suggest ways to approximate its embedding.

Part 2: Cosine Similarity Computation (20 points)

- Implement a function that computes the cosine similarity between two-word embeddings.
- Explain why cosine similarity is useful in word embedding space.
- Allow batch processing, where users can input multiple word pairs for simultaneous similarity computation.
- Visualization Requirement: Create a 2D or 3D scatter plot (e.g., using PCA or t-SNE) to visually show how similar and dissimilar words cluster together in the embedding space.

Part 3: Designing a Novel Dissimilarity Metric (20 points)

- Define a custom dissimilarity score that goes beyond cosine similarity. Possible approaches include:
    - Euclidean distance (How far apart words are in vector space).
    - Word entropy-based dissimilarity (How uncommon two words are relative to each other in corpora).
    - Semantic contrast measure (Using external knowledge bases like WordNet).

- Either design your own metric or cite an existing one from literature (provide a proper reference). Explain why your metric captures novelty/diversity better than cosine similarity alone.
- Allow users to toggle between different similarity/dissimilarity measures via function parameters.
- Visualization Requirement:
    - Plot the ranking of words based on their similarity/dissimilarity to a given word (e.g., how words like "cat" rank against "dog," "lion," and "table" using different metrics).
    - Use a heatmap to demonstrate and compare similarity and dissimilarity across multiple (any number of your choice) word pairs.