# Your Name: YUYUN FRANCIS BERINYUY

# Your Andrew ID: fyuyun

# Homework 1

# 1. Training Set Construction (5 pts)

Construct the training set for the amazon review dataset as instructed and report the following statistics.

|  |  |
| --- | --- |
| **Statistics** |  |
| the total number of unique words in T | 23206 |
| the total number of training examples in T | 2000 |
| the ratio of positive examples to negative examples in T | 1.00 |
| the average length of document in T | 187.68 |
| the max length of document in T | 3816 |

# 2. Performance of deep neural network for classification (20 pts)

Suggested hyperparameters:

1. Data processing:
   1. Word embedding dimension: 100
   2. Word Index: keep the most frequent 10k words
2. CNN
   1. Network: Word embedding lookup layer -> 1D CNN layer -> fully connected layer -> output prediction
   2. Number of filters: 100
   3. Filter length: 3
   4. CNN Activation: Relu
   5. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)
3. RNN:
   1. Network: Word embedding lookup layer -> LSTM layer -> fully connected layer(on the hidden state of the last LSTM cell) -> output prediction
   2. Hidden dimension for LSTM cell: 100
   3. Activation for LSTM cell: tanh
   4. Fully connected layer dimension 100, activation: None (i.e. this layer is linear)

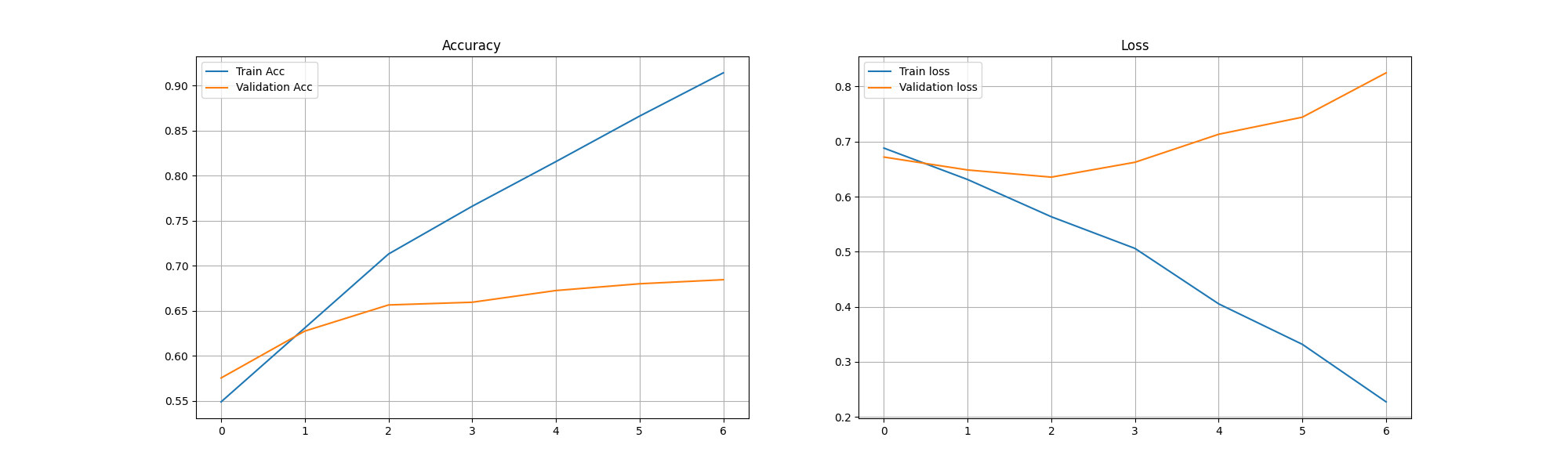
|  |  |  |
| --- | --- | --- |
|  | Accuracy | Training time(in seconds) |
| RNN w/o pretrained embedding | 67.60000000000001% | Approx 200s |
| RNN w/ pretrained embedding | 85.0% | Approx 300s |
| CNN w/o pretrained embedding | 77.40% | Approx 120s |
| CNN w/ pretrained embedding | 85.45% | Approx 180s |

# 3. Training behavior (10 pts)

Plot the training/testing objective, training/testing accuracy over time for the 4 model combinations (correspond to 4 rows in the above table). In other word, there should be 2\*4=8 graphs in total, each of which contains two curves (training and testing).

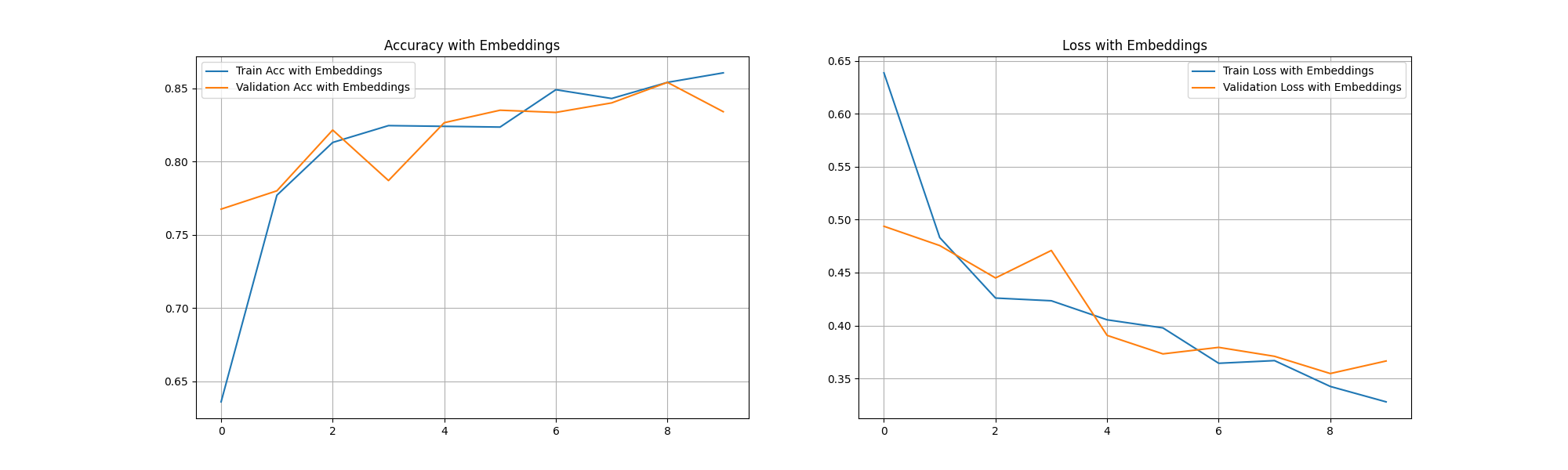
RNN w/o pretrained embedding

* training/testing objective over time
* training/testing accuracy over time



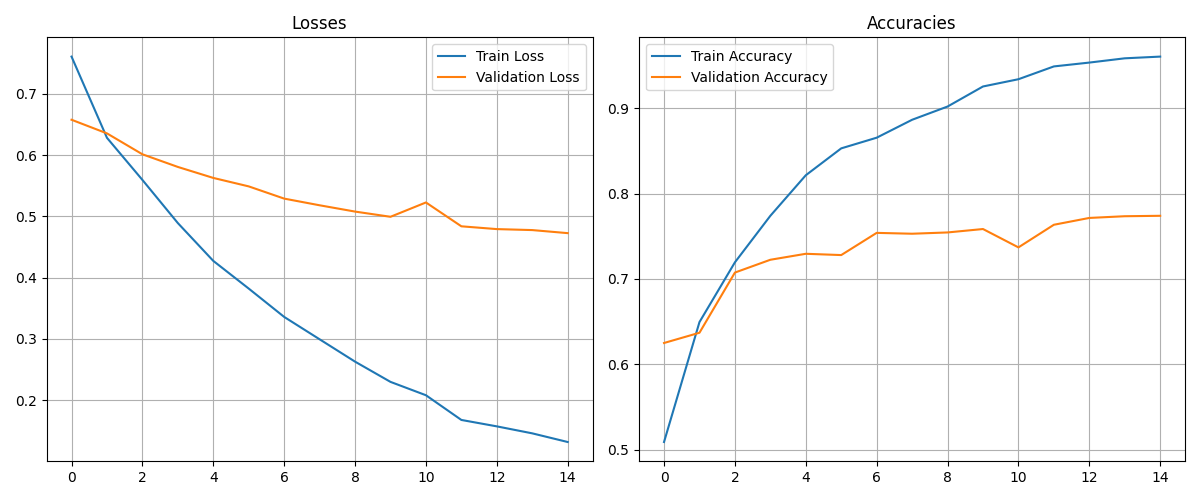
RNN w/ pretrained embedding

* training/testing objective over time
* training/testing accuracy over time



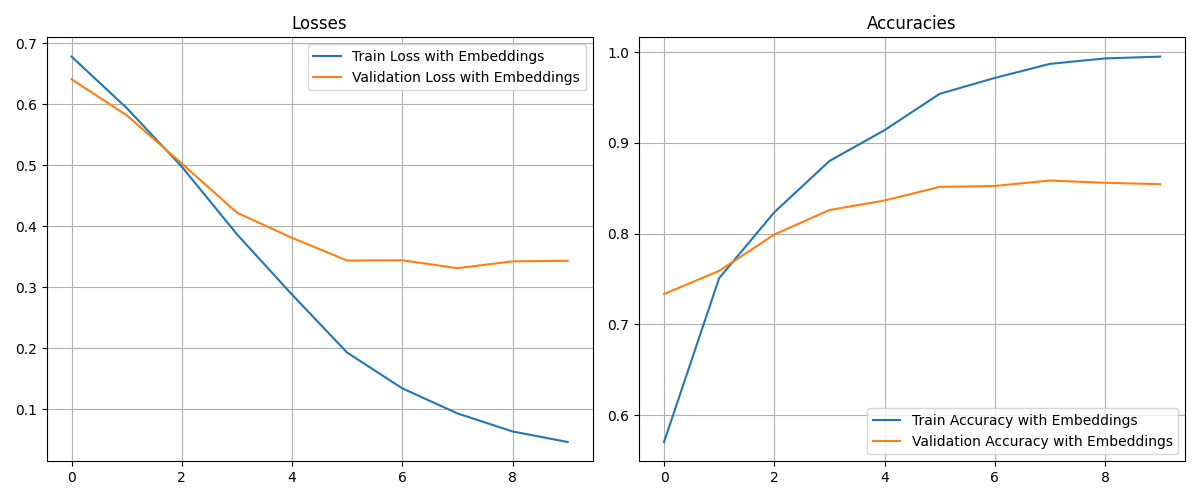
CNN w/o pretrained embedding

* training/testing objective over time
* training/testing accuracy over time



CNN w/ pretrained embedding

* training/testing objective over time
* training/testing accuracy over time



# 4. Analysis of results (10 pts)

Discuss the complete set of experimental results, comparing the algorithms to each other. Discuss your observations about the various algorithms, i.e., differences in how they performed, different parameters, what worked well and didn't, patterns/trends you observed across the set of experiments, etc. Try to explain why certain algorithms or approaches behaved the way they did.

**Overview**

The results show a comparison between a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) network, both with and without pretrained embeddings.

1. **CNN Performance**

* **Without Embeddings:**
  + Initial training accuracy was 53.05%, which gradually improved to 96.00% by epoch 15.
  + Validation accuracy started at 55.95% and improved to 78.10%.
  + The model showed a steady improvement in both training and validation accuracy, indicating effective learning.
  + The validation loss decreased consistently, suggesting that the model was not overfitting.
* **With Embeddings:**
  + Initial training accuracy was 58.70%, which improved to 99.65% by epoch 10.
  + Validation accuracy started at 67.70% and improved to 85.70%.
  + The model showed a significant improvement in both training and validation accuracy compared to the model without embeddings.
  + The validation loss decreased more rapidly, indicating that pretrained embeddings helped the model converge faster and more effectively.

**LSTM Performance**

* **Without Embeddings**:
  + Initial training accuracy was 55.55%, which improved to 86.9% by epoch 6.
  + Validation accuracy started at 59.75% and improved to 68.75%.
  + The model showed a steady improvement in training accuracy but had a fluctuating validation accuracy, indicating potential overfitting.
  + The validation loss decreased initially but then increased, suggesting overfitting after a few epochs.
* **With Embeddings:**
  + Initial training accuracy was 59.4%, which improved to 85.5% by epoch 10.
  + Validation accuracy started at 69.8% and improved to 83.35%.
  + The model showed a significant improvement in both training and validation accuracy compared to the model without embeddings.
  + The validation loss decreased consistently, indicating that pretrained embeddings helped the model generalize better.

**Observations and Patterns**

* **Impact of Pretrained Embeddings**:
  + Both models showed improved performance with pretrained embeddings. This is because pretrained embeddings capture how words relate to each other, which gives the models a better understanding of what the text means.
  + The models with embeddings converged faster and achieved higher validation accuracy, indicating better generalization.
* **Model Comparison:**
  + The CNN model generally performed better than the LSTM model in terms of validation accuracy.
  + CNNs are known for their ability to capture local patterns and features, which explains their better performance in our current amazon reviews task.
  + LSTMs, though at capturing sequential dependencies, struggled with relatively long sequences in our dataset.
* **Training and Validation Trends:**
  + Both models showed a steady improvement in training accuracy, but the validation accuracy trends differed.
  + The CNN model without embeddings showed a better improvement in validation accuracy, while the LSTM model without embeddings showed signs of overfitting.
  + With embeddings, both models showed more stable validation accuracy trends, indicating better generalization.
* **Early Stopping:**
  + Implemented early stopping to prevent overfitting, particularly for the LSTM model without embeddings, which showed fluctuating validation loss. This fluctuation indicated that there is overfitting.

**Conclusion**

* **Pre-trained Embeddings:** Significantly improved the performance of both models, leading to faster convergence and better generalization.
* **Model Choice:** The CNN model performed better than the LSTM model. This likely due to the ability of CNNs to capture long sequencial patterns better
* **Overfitting:** The LSTM model without embeddings showed signs of overfitting, highlighting the importance of regularization techniques and early stopping.

# 5. The software implementation (5 pts)

# Add detailed descriptions about software implementation & data preprocessing, including:

1. A description of what you did to preprocess the dataset to make your implementations easier or more efficient.

To preprocess the dataset and make the implementations easier and more efficient:

* **Loading Data:** Loaded the data from text files, with separate directories for positive and negative reviews.
* **Cleaning Text:** Cleaned the text data by removing punctuation, extra spaces, and digits using regular expressions.
* **Tokenization:** The text was tokenized into words, and stopwords were removed to reduce noise.
* **Vocabulary Creation:** Created a vocabulary of the most frequent words, limiting the size to 10,000 words.
* **One-Hot Encoding:** Converted the words to one-hot encoded vectors based on the vocabulary.
* **Padding:** Padded the sequences to a fixed length (500) to ensure uniform input size for the models.
* **Splitting Data:** The data was split into training and testing sets, and shuffled to ensure randomness.

2. A description of major data structures (if any); any programming tools or libraries that you used;

**Data Structures:**

* **Lists**: Used to store raw text data and labels.
* **Dictionaries**: Used for the vocabulary and one-hot encoding mappings.
* **Numpy Arrays:** Used for storing the final processed data and labels.
* **PyTorch Tensors:** Used for model inputs and outputs.

**Libraries:**

* **Pandas:** For data manipulation and creating dataframes.
* **Numpy:** For numerical operations and array manipulations.
* **Torch:** For building and training neural network models.
* **Matplotlib & Seaborn:** For data visualization and plotting.

3. Strengths and weaknesses of your design, and any problems that your system encountered;

**Strengths:**

* Efficiency: The preprocessing steps ensure that the data is in a format suitable for efficient model training.
* Flexibility: The use of dictionaries and lists allows for easy manipulation and extension of the vocabulary and data.
* Reusability: The code is modular, making it easy to reuse and adapt for different datasets or tasks.
* Visualization: The use of visualization libraries helps in understanding the data distribution and model performance.

**Weaknesses:**

* Memory Usage: The use of large lists and dictionaries can lead to high memory usage, especially with large datasets.
* Overfitting: The LSTM model without embeddings showed signs of overfitting, indicating a need for better regularization techniques.
* Complexity: The preprocessing steps and model training can be complex and require careful tuning of parameters.

**Problems Encountered:**

* Overfitting: The LSTM model without embeddings overfitted the training data, requiring early stopping and regularization.
* Imbalanced Data: The dataset had an equal number of positive and negative reviews, but real-world data might be imbalanced, requiring additional handling.
* Long Sequences: Handling very long sequences required careful padding and truncation to ensure uniform input sizes.