CSCI 544 – Homework 3

Data Generation

- For this homework, I have used the same methodology used in homework 1 to generate the dataset.
- I extracted the review_body and rating column from the dataset for this homework.
- I formed two classes and selected 50,000 reviews randomly from each class.
- Performed some data cleaning similar to that of Homework 1
- Finally, I split the dataset into 80-20% training and testing split.

Word Embedding

a) Try to check semantic similarities of the generated vectors using three examples of your own.

Using the genism model, I loaded the 'word2vec-google-news-300' model and checked for similarities using the 'most_similar' and 'cosine_similarity' functions.

- most_similar(positive=['king','woman'], negative=['man'])) gave us all the words that are similar to king + woman – man

  [('queen', 0.7118193507194519), ('monarch', 0.6189674139022827), ('princess', 0.5902431011199951), ('crown_prince', 0.5499460697174072), ('prince', 0.5377321839332581), ('kings', 0.5236844420433044), ('Queen_Consort', 0.5235945582389832), ('queens', 0.5181134343147278), ('sultan', 0.5098593831062317), ('monarchy', 0.5087411999702454)]
- most_similar('excellent')) gave us all the words similar to excellent

  [('terrific', 0.7409726977348328), ('superb', 0.7062715888023376), ('exceptional', 0.681470513343811), ('fantastic', 0.6802847981452942), ('good', 0.6442928910255432), ('great', 0.6124600172042847), ('Excellent', 0.6091997623443604), ('impeccable', 0.5980967283248901), ('exemplary', 0.5959650278091431), ('marvelous', 0.582928478717804)]
- most_similar('happy')) gave us all the words similar to happy

  [('glad', 0.7408890724182129), ('pleased', 0.6632170677185059), ('ecstatic', 0.6626912355422974), ('overjoyed', 0.6599286794662476), ('thrilled', 0.6514049172401428), ('satisfied', 0.6437949538230896), ('proud', 0.636042058467865), ('delighted', 0.627237856388092), ('disappointed', 0.6269949674606323), ('excited', 0.6247665286064148)]

b) **Check the semantic similarities for the same two examples in part (a). What do you conclude from comparing vectors generated by yourself and the pretrained model? Which of the Word2Vec models seems to encode semantic similarities between words better?**

In this part (b), I created a Word2Vec model and trained it on our dataset of reviews but splitting each review into words. For comparing, the two models I used the cosine similarity.

- 'he' and 'she'

  Custom model - 0.84833896

  Pretrained model – 0.61299497

  The custom trained model has better similarity compared to the pretrained model, mainly due to the large frequency of these words in or dataset.

- 'excellent' and 'outstanding'

  Custom model - 0.82512033

  Pretrained model – 0.5567486

  The custom trained model has better similarity compared to the pretrained model, mainly because of the dataset being reviews, words like 'excellent' or its synonym might appear often.

- 'him' and ('father'-'she')

  Custom model - -0.65903854

  Pretrained model – 0.0734347

  The custom trained model has better similarity compared to the pretrained model, mainly because of the dataset being reviews, reviews like "my father gifted me .." or "she liked it" can be seen often making it more eminent in the dataset.

Simple models

Perceptron –

1. Word2Vec - 0.8162753181573759
2. TF-IDF - 0.8092918311881443

SVM –

1. Word2Vec - 0.8264705882352941
2. TF-IDF - 0.8708965111143826

Conclusion –

Performance Comparison: The SVM model generally outperforms the Perceptron model for both Word2Vec and TF-IDF features. This suggests that the SVM model is better at finding decision boundaries that separate the classes in our dataset.

Feature Type Comparison:

- For Word2Vec features, both models perform relatively well, but SVM slightly outperforms Perceptron. Word2Vec features capture semantic information, which can be useful for sentiment analysis tasks.
- For TF-IDF features, the SVM model significantly outperforms the Perceptron. TF-IDF features represent the importance of words in documents, and in this case, it appears that they are more effective for our classification task.

Overall: The choice of feature type (Word2Vec vs. TF-IDF) has a significant impact on the model's performance. SVMs are more robust and benefit more from TF-IDF features, which capture the importance of individual words, while Word2Vec features are more effective for the Perceptron and still perform reasonably well.

In summary, the SVM model, when using TF-IDF features, provides the best performance on our classification task. However, the choice between the Perceptron and SVM and between Word2Vec and TF-IDF should consider the specific characteristics of our dataset and the requirements of our application.

Feedforward Neural Networks

a) **Report accuracy values on the testing split for your MLP**
   Accuracy of MLP: 80.33 %

b) **Report the accuracy value on the testing split for your MLP model. What do you conclude by comparing accuracy values you obtain with those obtained in the "Simple Models" section.**
   Accuracy of MLP (concatenate first 10 Word2Vec vectors): 74.285 %

   Now, let's draw some conclusions from these results:

   Impact of Input Features:

   - In Part A, you used average Word2Vec vectors as input features. These vectors represent a smoothed and aggregated representation of the words in each review.

- In Part B, you concatenated the first 10 Word2Vec vectors for each review as input features. This means you're using more granular and detailed information from the words at the beginning of each review.

Accuracy Comparison:

- Part A achieved a higher accuracy of 80.72%. This suggests that the average Word2Vec vectors, which capture the overall semantics of the reviews, were effective in distinguishing between the sentiment classes.
- In Part B, where you concatenated the first 10 Word2Vec vectors, you saw a lower accuracy of 74.285%. This lower accuracy might be due to using only the initial word vectors, which may not capture the overall sentiment as effectively as averaging all vectors in the review.

Model Learning:

- Part A's higher accuracy indicates that the model trained on average Word2Vec vectors learned to make predictions based on the semantic content of the entire review.
- In Part B, the model focused on the first 10-word vectors, which might not provide enough information to make accurate predictions for all reviews. It may work well for some cases but not for others.

Conclusion:

Based on the results, it's apparent that the choice of input features has a substantial impact on the model's performance. Using the average Word2Vec vectors (Part A) seems to be a better choice for our sentiment analysis task, as it captures a more comprehensive view of the review content.

Recurrent Neural Networks

a) **Report accuracy values on the testing split for your RNN model. What do you conclude by comparing accuracy values you obtain with those obtained with feedforward neural network models.**

Accuracy for RNN model - 77.315 %

Conclusion –

The FNN model (4.a) outperformed the RNN model (5.a) in terms of accuracy. The higher accuracy of FNN model might be attributed to the choice of input features and the overall model architecture.

FNNs, using average Word2Vec vectors, seem to capture the overall sentiment information more effectively for this specific sentiment analysis task. The hidden state size of 10 in the RNN might not be sufficient to capture the complexity and dependencies in the data, which could be a reason for the slightly lower accuracy.

In conclusion, based on the provided results, the feedforward neural network (FNN) model, particularly the one using average Word2Vec vectors as input features, outperformed the simple RNN for the sentiment analysis task. However, the specific choice of architecture and input features depends on the characteristics of the dataset and the task at hand. Experimentation with different models and hyperparameters is crucial to determine the best approach for your particular sentiment analysis task.

b) Repeat part (a) by considering a gated recurrent unit cell.

Accuracy for GRU - 77.625 %

c) Repeat part (a) by considering an LSTM unit cell.

Accuracy of LSTM: 77.68 %

What do you conclude by comparing accuracy values you obtain by GRU, LSTM, and simple RNN.

Conclusions:

- Similar Accuracy: It's interesting to note that all three RNN variations, including the Simple RNN, GRU, and LSTM, achieved similar accuracies on the testing data.
- RNN Memory: Despite differences in the internal architecture and handling of long-term dependencies, these three RNN cell types produced identical accuracy results for this particular sentiment analysis task.
- Dataset Characteristics: This suggests that, for this specific dataset and sentiment analysis task, the dataset might not have strong, long-range dependencies that would benefit from the advanced memory capabilities of GRU and LSTM.
- Model Complexity: While GRU and LSTM are more complex and have enhanced memory capabilities, this complexity might not be necessary for this specific sentiment analysis problem.
- Choice of Hyperparameters: The choice of hyperparameters, such as hidden state size and learning rate, can significantly impact model

performance. The identical accuracy values could be due to similar hyperparameters used for all RNN cell types.

- Experimentation: It's essential to keep in mind that the performance of these RNN variations may differ significantly for different datasets or tasks. Experimentation and hyperparameter tuning are key to finding the best-performing architecture for a specific problem.

In summary, the choice of RNN cell type, whether Simple RNN, GRU, or LSTM, did not significantly impact the accuracy for this particular sentiment analysis task. This suggests that, for this specific dataset, the relatively simpler Simple RNN was sufficient to capture the necessary patterns and dependencies. Further optimization and experimentation might reveal differences in performance for more complex datasets or tasks.

Accuracies –

Model Name	Accuracy (in %)
Perceptron (Word2Vec)	81.62
Perceptron (TF-IDF)	80.92
SVM (Word2Vec)	82.64
SVM (TF-IDF)	87.08
Simple FNN	80.33
FNN Concat	74.285
RNN	77.315
GRU	77.625
LSTM	77.68