Natural Language Processing:
Sentiment Analysis on Rotten Tomatoes Dataset

This report describes our method in training a sentiment analysis model. Sentiment analysis is an example of text categorization, where we predict a given review (words or sentences) to be either positive or negative. Our dataset originated from Rotten Tomatoes, a review aggregation website for films and television shows. The dataset is introduced in the proceedings of ACL 2005 conference and has been pre processed and modified for research purposes [BPL05]. The dataset is available in the *datasets* library via *pip install datasets*. We imported the library, downloaded the dataset, and split it into the default train-val-test split given by the line below.

```
from datasets import load_dataset
dataset = load_dataset ("rotten_tomatoes")
train_dataset = dataset [‘train’]
validation_dataset = dataset [’validation’]
test_dataset = dataset [’test’]
```

Our model is initialised by pre-trained dense word embeddings (an array of continuous values) to represent words. This generic representation is then used off-the-shelf to train a recurrent neural network (RNN) for sequence modelling to solve our particular task.

Part 1. Preparing Word Embeddings

We chose Word2Vec [Mik+13] as our pre-trained word embedding initialization. Word2Vec is trained on a binary classification task with the following formulation: given a centre word C, predict whether a word I is a context or surrounding word. This task is solved by modelling each word as two vectors, one for the case when the word is a context, and one for centre. The model uses logistic regression with negative sampling to build synthetic negative examples. We use a pre-trained model available on *gensim* library. This model represents each word as a 300-dimensional vector.

```
import numpy as np
import gensim.downloader as api
word_vectors = api.load("word2vec-google-news-300")
```

We train our model on the training split of the dataset. We obtained our training data to consist of 18951 distinct words that make up the vocabulary. Out of these words, 4585 words do not exist in the pre-trained Word2Vec model. To handle these out-of-vocabulary (OOV) words, we map each of such words to be represented by a single zero vector.

```
for word in oov_words:
```

```
    idx = word_to_idx[word]
    embedding_matrix[idx] = np.zeros(embedding_dimension)
```

The snippet above shows zero vector embedding, which simply assigns a zero vector for OOV words, treating it as having no semantic meaning.

Assuming OOV words are non-critical, using zero vectors is justified as they prevent these words from affecting model outputs, avoiding noise and maintaining accuracy. This neutral impact simplifies analysis and debugging via consistent OOV handling.

Part 2. Model Training & Evaluation

Our first approach involves passing each word, represented as a Word2Vec vector to a single Elman RNN layer [MSC+13]. RNN is a special type of neural network that incorporates the context of previous output to current input. This architecture is known to improve performance for sequential data, such as text and audio. The architecture computes the hidden state (output) in a sequential manner. We take the hidden state of the last token in the input and pass it to a fully-connected layer with sigmoid activation function to output a single value, the probability that the sequence is a positive sequence. Here, we take the threshold to be 0.5, i.e., the review is taken to be positive when the probability is greater than 0.5, and negative otherwise. The implementation for the RNN model is available on *PyTorch*, a pythonic and flexible deep learning framework. The code snippet explains how a custom model is declared in a class inherited from *nn.Module* class. For this section, we freeze the pre-trained embedding matrix to disable it from training. This is easily done by setting the freeze parameter of the *nn.embedding.from_pretrained* method to be True.

```
import torch
import torch.nn as nn

class SentimentRNN(nn.Module):
  def __init__(embed_matrix, . . .):
    self.embedding = nn.embedding.from_pretrained(embed_matrix, freeze=True)
    self.rnn = nn.LSTM(input_size, hidden_size, num_layers, dropout)
```

(a) Report the final configuration of your best model, namely the number of training epochs, learning rate, optimizer, batch size.

The final hyperparameters are shown in this table:

Number of epochs	10
Learning rate	0.001
Optimizer	Adam Optimizer
Batch size	64

(b) Report the accuracy score on the test set, as well as the accuracy score on the validation set for each epoch during training.

Epoch	Validation Accuracy	Test Accuracy
1	0.7383	0.7495
2	0.7214	0.7345
3	0.7758	0.7767
4	0.7852	0.7758
5	0.7814	0.7814
6	0.7711	0.7739
7	0.8002	0.7795
8	0.7833	0.7777
9	0.7842	0.772
10	0.7927	0.7664

(c) RNNs produce a hidden vector for each word, instead of the entire sentence. Which methods have you tried in deriving the final sentence representation to perform sentiment classification? Describe all the strategies you have implemented, together with their accuracy scores on the test set.

The score reported in part (b) is when we used the hidden state representation for the last token to be passed the fully-connected network. We also tried implementing mean pooling and max pooling and obtained the test accuracies to be the same: 0.7777.

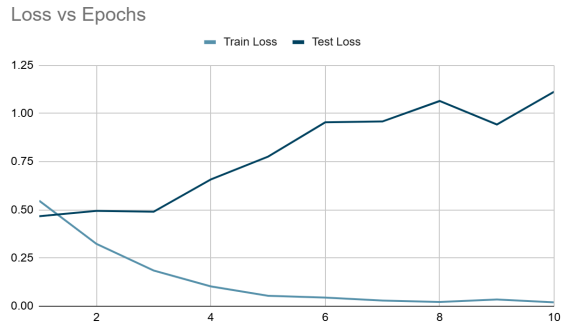
Part 3. Enhancement

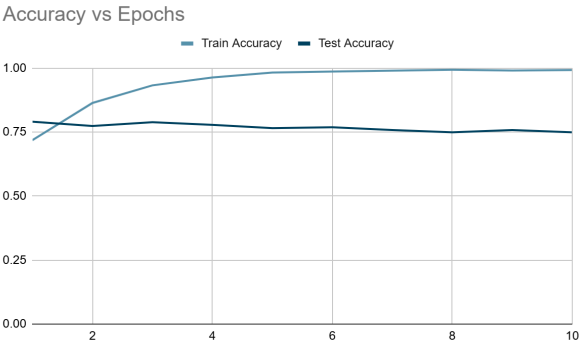
Part 3.1 Update word embeddings during the training process

(a) Report the accuracy score on the test set when the word embeddings are updated (Part 3.1)

To update the word embeddings during training, we ensured that the embedding layer is not frozen. This allows the embeddings to be trained alongside the other model parameters. The word embeddings will gradually adjust based on the sentiment task, potentially improving performance.

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.5476	0.7186	0.4672	0.7908
2	0.3234	0.8640	0.4945	0.7739
3	0.1852	0.9328	0.4907	0.7889
4	0.1028	0.9634	0.6580	0.7786
5	0.0542	0.9830	0.7761	0.7655
6	0.0446	0.9866	0.9547	0.7692
7	0.0294	0.9903	0.9587	0.7580
8	0.0219	0.9936	1.0650	0.7495
9	0.0351	0.9909	0.9430	0.7580
10	0.0202	0.9927	1.1129	0.7495





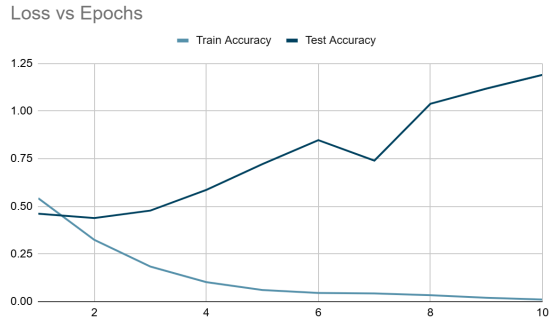
Part 3.2 Apply solution of mitigating influence of OOV words

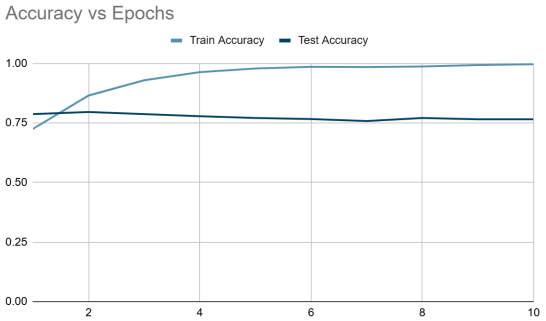
(b) Report the accuracy score on the test set when applying your method to deal with OOV words in Part 3.2.

Since we assigned OOV words to be zero vectors in the embedding matrix, the OOV words have reduced influence during training and testing.

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.5425	0.7254	0.4616	0.7871
2	0.3242	0.8654	0.4384	0.7964
3	0.1843	0.9294	0.4778	0.7871
4	0.1023	0.9638	0.5862	0.7786

5	**0.0608**	**0.9790**	**0.7215**	**0.7711**
6	**0.0457**	**0.9857**	**0.8474**	**0.7664**
7	**0.0432**	**0.9849**	**0.7402**	**0.7580**
8	**0.0335**	**0.9871**	**1.0382**	**0.7711**
9	**0.0200**	**0.9934**	**1.1182**	**0.7655**
10	**0.0108**	**0.9968**	**1.1902**	**0.7655**





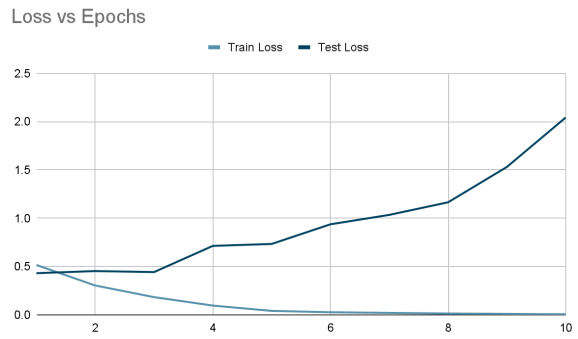
Part 3.3 biLSTM and biGRU

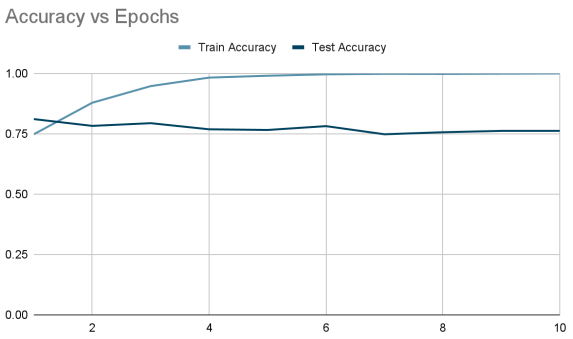
In this analysis, we will utilise biLSTM and biGRU models to evaluate their accuracy over 10 epochs.

biLSTM

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	**0.5164**	**0.7354**	**0.4315**	**0.8049**
2	**0.3040**	**0.8706**	**0.4533**	**0.7946**
3	**0.1837**	**0.9327**	**0.4428**	**0.7917**
4	**0.0962**	**0.9661**	**0.7141**	**0.7645**
5	**0.0412**	**0.9860**	**0.7340**	**0.7777**
6	**0.0271**	**0.9897**	**0.9375**	**0.7674**
7	**0.0204**	**0.9936**	**1.0349**	**0.7711**

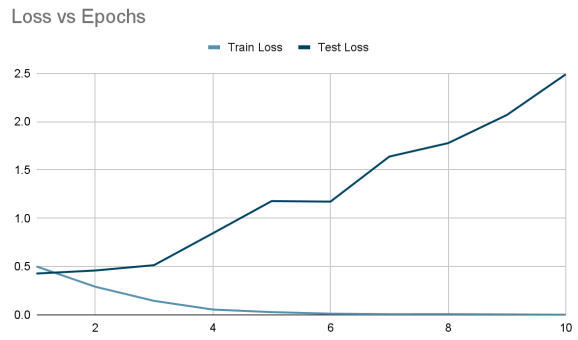
8	0.0140	0.9951	1.1653	0.7448
9	0.0104	0.9964	1.5313	0.7636
10	0.0058	0.9984	2.0428	0.7636

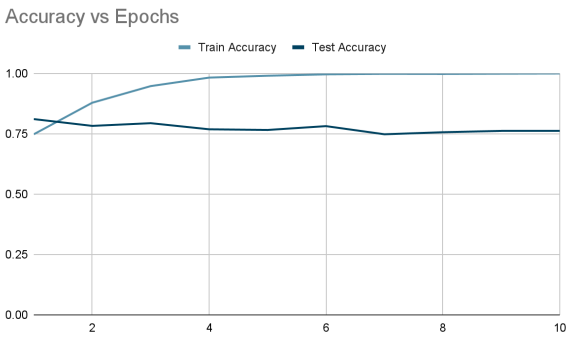




biGRU

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.5013	0.7478	0.4278	0.8105
2	0.2914	0.8783	0.4592	0.7824
3	0.1449	0.9471	0.5144	0.7936
4	0.0550	0.9823	0.8443	0.7683
5	0.0284	0.9900	1.1777	0.7655
6	0.0126	0.9958	1.1720	0.7814
7	0.0073	0.9979	1.6382	0.7477
8	0.0078	0.9974	1.7774	0.7561
9	0.0047	0.9985	2.0692	0.7617
10	0.0016	0.9993	2.4898	0.7617





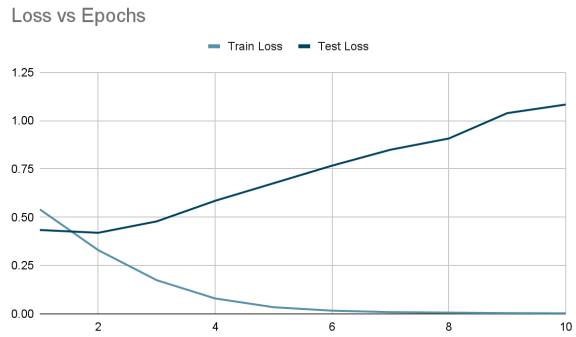
Replacing a simple RNN model with a bidirectional GRU (biGRU) or bidirectional LSTM (biLSTM) model significantly enhances the model's ability to understand context by incorporating recurrent computations in both directions. This advancement is primarily due to the bidirectional computations, which allow the model to process input sequences from both forward and backward directions. As a result, for each time step, the biGRU or biLSTM has access to both past and future context. This dual access leads to a more comprehensive understanding of the sequence, enabling the model to capture dependencies and relationships within the data more effectively. Such an approach is particularly beneficial for tasks like sentiment analysis, where the meaning of a word can be influenced by both preceding and following words. By considering information from both directions, these models can discern subtleties in language that a unidirectional model might overlook.

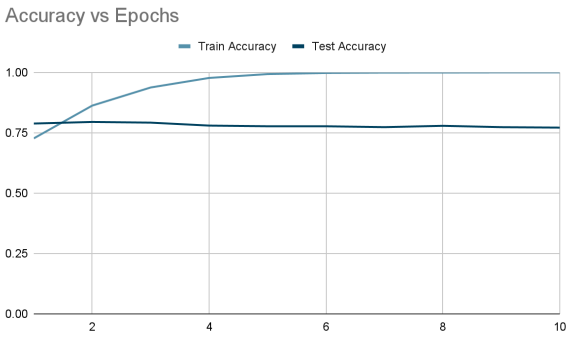
In addition to bidirectional computations, stacking multiple layers of GRUs or LSTMs further enhances the model's capacity. Each layer in a stacked architecture can learn increasingly complex representations of the data. The lower layers tend to focus on local patterns, capturing immediate relationships within the data, while higher layers are capable of understanding more global patterns and abstractions. This hierarchical learning structure allows for improved feature extraction, as multiple layers enable the model to distil more nuanced features from the input data. Consequently, this leads to better performance across various tasks, as the model learns to represent the data in ways that are more informative and relevant for specific applications.

Part 3.4 CNN

CNN

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	**0.5408**	**0.7267**	**0.434**	**0.788**
2	**0.3298**	**0.8623**	**0.4196**	**0.7946**
3	**0.1743**	**0.9373**	**0.4786**	**0.7917**
4	**0.0797**	**0.9768**	**0.5851**	**0.7795**
5	**0.0344**	**0.9927**	**0.676**	**0.7767**
6	**0.0165**	**0.9973**	**0.7669**	**0.7767**
7	**0.009**	**0.9989**	**0.8494**	**0.773**
8	**0.0066**	**0.9991**	**0.9076**	**0.7786**
9	**0.0039**	**0.9996**	**1.0389**	**0.773**
10	**0.0028**	**0.9995**	**1.0832**	**0.7711**





Replacing the simple RNN model with CNN improves the model's capability to capture local feature patterns within the text as CNNs apply multiple convolutional filters of varying sizes to the input word embeddings, enabling the detection of n-gram features which aid in sentiment classification. For example, a filter size of three can identify trigrams such as "not good" or "very bad," which are important in determining the sentiment polarity of a review. These convolutional operations allow the model to recognise and learn important patterns irrespective of their position in the sentence.

In addition to convolutional layers, pooling layers like max pooling are incorporated to reduce the dimensionality of the feature maps generated by the convolutional filters. Max pooling retains the most important features by selecting the maximum value within each window, thereby highlighting the most significant n-gram features for sentiment prediction. This not only minimises computational complexity but also enhances the model's focus on the most impactful parts of the text. **However, while CNNs are capable of extracting local features, they are limited in capturing long-range dependencies and broader contextual relationships within the text. This constraint prevents CNNs from fully understanding the comprehensive context and nuanced sentiments present in longer and more complex reviews, which bidirectional LSTM/GRU models are inherently designed to handle more effectively.**

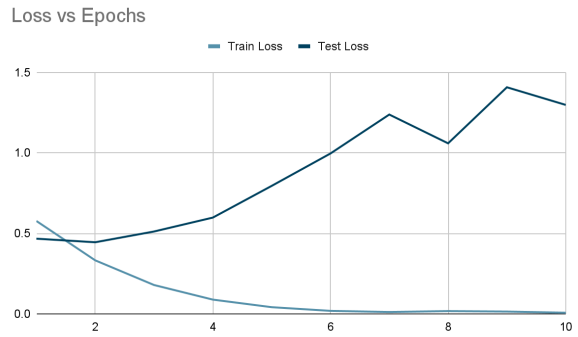
Comparative Performance: Our findings indicate that the **bidirectional models (biLSTM/biGRU)** outperform **CNNs**, which in turn outperform **simple RNNs**. This performance hierarchy is closely tied to the characteristics of the Rotten Tomatoes dataset. **Bidirectional LSTM/GRU models** achieve the highest accuracy because they process the text in both forward and backward directions, hence are more effective at capturing contextual information and long-range dependencies essential for understanding nuanced sentiments in lengthy and complex reviews. **CNNs** perform better than simple RNNs by efficiently extracting local sentiment-rich features and patterns through convolutional and pooling layers, which are beneficial for identifying key phrases within diverse vocabulary. However, they do not fully capture the extensive contextual dependencies that bidirectional models manage, leading to slightly lower performance. **Simple RNNs** are limited by their inability to handle long-range dependencies and contextual nuances as effectively as the more advanced architectures, resulting in the lowest accuracy among the three.

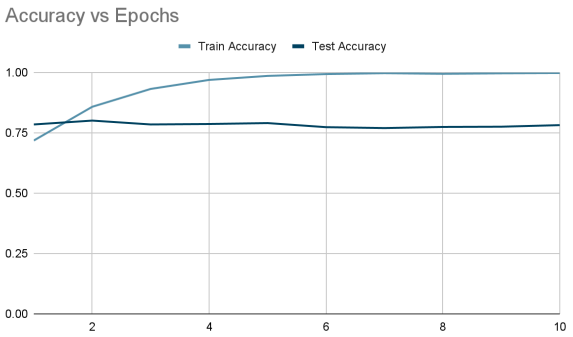
Part 3.5 Further Improvements

(e) Describe your final improvement strategy in Part 3.5. Report the accuracy on the test set using your improved model.

Added attention to the biLSTM mode. Attention allows the model to focus more on words that are sentimentally important and assign higher weights to them [BCB14]. When higher weights are assigned to certain words, the word will contribute more and play a bigger role in the model's prediction. This selective focus often results in better generalisation to unseen data, which can lead to increased accuracy.

Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
1	0.5768	0.7178	0.4669	0.7842
2	0.3324	0.8574	0.4447	0.8002
3	0.1792	0.9313	0.5115	0.7842
4	0.0885	0.9686	0.5979	0.7861
5	0.0415	0.9852	0.7947	0.7899
6	0.0186	0.9927	0.9960	0.7730
7	0.0114	0.9965	1.2378	0.7692
8	0.0173	0.9938	1.0591	0.7739
9	0.0142	0.9959	1.4072	0.7749
10	0.0070	0.9972	1.2977	0.7814





References

[BPL05] Bo Pang and Lillian Lee. “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales.” In *Proceedings of the Association for Computational Linguistics Conference*, 2005.

[Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." In *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 2013.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." In: *Proceedings of the International Conference on Learning Representations (ICLR 2015)*, 2014.