Review on Sentiment Analysis

Abstract

Sentiment analysis is well-known task in NLP that serves as a basis for more fine-grained tasks and provide unexplored practical applications from the analysis of online reviews and comments. The focus of this work is to evaluate the effect of different model architectures and pre-trained word embeddings on sentiment analysis. The models selected were T5, BERT, LSTM, RNN and CNN. The general results were satisfactory, with the pre-trained T5 and BERT models outperforming the remaining. Whereas the embeddings selected were GloVe and word2vec, and tested on LSTM and RNN models. While the results were similar, the 100-dimension version of GloVe 6B performed the best. Overall, the project was successfully performed, and laid foundations for possible future work including transfer learning and fine-grained sentiment analysis.

Introduction

Sentiment analysis is a natural language processing task that classifies text into various sentiment categories, such as positive, negative, or neutral. The input would usually be a text sequence, and the output would be a class label (in some datasets, there can be 5 classes including negative, somewhat negative, neutral, somewhat positive, and positive). For example, the text "This is such a great show!" should likely be classified as "positive".

With the explosion of social media platforms in recent decades, we express our thoughts very frequently online, such as through Instagram posts, comments, tweets and different reviews. This massive amount of natural language data can give valuable insights into what people think, and whether they think positively or negatively towards a certain topic, product or item. For example, during elections, performing sentiment analysis on forums and news articles can suggest who has more supporters. Or product vendors can use sentiment analysis to quickly grasp how well-received the product is by sentiment analysis of its reviews and social media posts mentioning it. All these can facilitate decision-making in numerous aspects,

Another interesting thing is the complexity of the problem. This task seems pretty simple to humans, who are trained to interpret others' emotions through verbal and non-verbal signs since we were born. However, for machines, it can be hard to identify some underlying irony and sarcasm through simple rule-based methods like having a list of "positive" words and a list of "negative"

words and use that to score sentences' positiveness. For example, the review "This movie was actually neither that funny, nor super witty." is a negative one, but the words "funny" and "witty" can confuse the models to treat it as a positive one [4]. For a machine to accurately classify the sentiment expressed in text, it must have a deep understanding of nuances in natural language and be able to interpret the context in which the text is written.

In fact, when doing research on this topic, I am surprised to find out that despite the great capabilities of language models like GPT-3.5 which seems very human-like to many, the state-of-the-art(SOTA) performance on multi-class sentiment analysis datasets are far from perfect (SOTA got below low 60% accuracy at SST-5 dataset [12]). This may hint that while AIs are good at interpreting and reconstructing texts, there may still be lacking in understanding the underlying meaning, emotions and other signal hidden behind words. This makes sentiment analysis an interesting and curious topic to work on, since it is a relatively simple one in these kind of analysis, while provides a good stepping stone to understanding the complexity in natural language.

Similar to NLP, diverse and ever-changing methods were applied to approach this problem. In general, they can be divided into lexicon, machine learning and deep learning based. Considering the nature of the course, the focus of this project will be on some machine learning and mostly deep learning methods used to handle the problem. To allow a better picture of the field's development, significant deep learning model architectures from fully-connected networks to the recent pre-trained transformers will be trained and compared. Another comparison dimension would be word embeddings, which includes Word2Vec, GloVe and other pre-trained embeddings. Hopefully this work can highlighting the field's progress and provide a better picture of what may be expected in the future, and in what areas can new solutions be worked on (e.g. more fine-grained classification, multi-lingual texts, transfer learning).

Related Work

To me, model architecture and word embeddings are two different parts of building a solutions. Different embeddings can be mixed-and-matched with different model architectures. Hence, for the sake of clarity and a better comparison, they will be separately discussed in the following section.

Model Architecture

There are three major sentiment analysis solution types: lexicon-based, machine learning-based and deep learning-based, some common models or approahces are shown below [3].

Lexicon-based work can include some rule-based approaches like the sementic orientation [1]. While some common machine learning approahces include Naive Bayes classification in NLTK[2], one of the most commonly used in practical applications. It is a machine learning approach based on Bayes' theorem, which states that the probability of a hypothesis (in this case, the class of a text document) given the observed evidence (the words in the document) is proportional to the probability of the evidence given the hypothesis multiplied by the prior probability of the hypothesis. Other machine learning models like logistic regression and SVM are common also.

While in this work, the focus will be on deep learning.

A model I would take note of is the Multi-View Recurrent Neural Network neural network architecture proposed by Stanford researchers, together with the sentiment analysis treebank dataset[4].

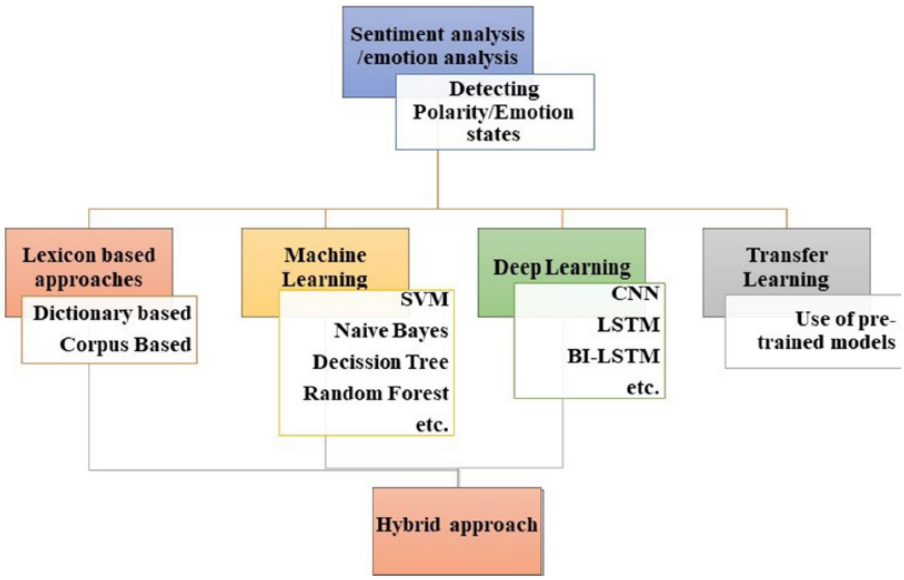

Figure 1: Common approaches for sentiment analysis

The architecture is based on a combination of multiple views of the input text data, which are integrated using a recurrent neural network (RNN) to make the final sentiment prediction. The use of multiple views allows the model to capture more complex patterns in the input data, which can improve the accuracy of sentiment prediction.

Other common architectures including LSTM, CNN [5] and BERT[7] are also important milestones to be investigated.

Word Embeddings

Word embedding is an important tool for NLP which quantify text sequences. Its importance is highlighted in the paper proposing the significant word2vec [8]. It enables the conversion from text to high-dimensional number vectors that to some extent capture the words meaning and similarity. This work takes into account the importance of context when labelling the meaning of words, with the important principle of "words appearing in similar context likely have similar meanings". Hence, this work uses neural network to learn the word embeddings from the context in which words appear.

While GloVe is another commonly used embedding using co-occurence matrix to compute embeddings [13]. It represents vectors as the log of the ratio of co-occurrence probabilities. It is usually compared with Word2Vec, in which the two have rivaling performances and often require testing out which one is better for the specific task.

Moreover, there are more pre-trained text embeddings based on the two algorithms mentioned above. For example, universal sentence encoder [11] is a powerful encoder trained on a large corpus of text data and can generate high-quality embeddings for full sentences or short phrases.

Apart from that, I have found two more pre-trained embeddings that are trained on the Google

News corpus. They are the gnews-swivel embeddings, a method that captures the statistical relationships between words and generates a high-dimensional vector representation of each word[9]; and gnews-nnlm embeddings based on a feedforward neural network that is trained to predict the next word in a sequence of words[10].

Data

The selected datasets for the evaluation are widely used benchmarking datasets in the field of natural language processing. These datasets include the IMDB movie review dataset and Stanford's SST dataset, both of which have been utilized by numerous researchers. By analyzing the performance of the trained models on these datasets, a more robust benchmarking and evaluation of their performance can be achieved.

In addition to the benchmarking datasets, a financial sentiment analysis dataset was selected from Kaggle to introduce an additional dimension to the evaluation process. The benchmarking datasets mainly consist of reviews which are subjective commentaries on specific aspects. This aspect poses a possible bias and gap in evaluating the model's performance. The financial sentiment analysis dataset includes sentences that provide financial information such as company profits and stock prices.

Notably, many of these sentences are factual and objective, and the sentiment lies in the nature of the news, rather than in the emotions and opinions expressed. For instance, a text discussing rising profits is likely classified as "positive." This dataset presents an interesting supplement to the "conventionally used" datasets and raises the question of potential discrepancies in performance between these two types of datasets. Further analysis and evaluation are necessary to explore these discrepancies.

IMDB movie review

Description The IMDB movie review dataset is a large dataset of movie reviews collected from the IMDB website. The dataset includes 50,000 reviews, each labeled as either positive or negative. The goal of the dataset is to provide a benchmark for natural language processing (NLP) algorithms to classify the sentiment of movie reviews.

An example input for this dataset would be a movie review such as "I really enjoyed this movie. The acting was great and the story was engaging." The corresponding output would be a label of "positive".

Source of Data The IMDB movie review dataset is available on the website of Stanford University's Stanford AI Lab (SAIL). The dataset was created by Andrew Maas, along with other researchers from SAIL. For ease of implementation, I used the dataset of Kaggle for training the models: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

Relevant Statistics The state-of-the-art models on this dataset typically achieve accuracy scores in the range of 94-96%. Current SOTA XLNet achieved 96%, BERT achieved 95% and a CNN+LSTM approach achieved 89% [15].

Stanford SST-2

Description The Stanford Sentiment Treebank (SST-2) is a dataset of movie reviews and their corresponding sentiment labels. The reviews are taken from the Rotten Tomatoes movie review website and are labelled as either positive or negative. A special feature of the SST-2 dataset is that

it provides a fine-grained sentiment label for each sentence in a review rather than a single label for the entire review.

An example input for this dataset would be a movie review sentence such as "The acting was excellent, but the plot was slow." The corresponding output would be a label of "positive" for the first part of the sentence and "negative" for the second part.

Source of Data The SST-2 dataset was created by Stanford University's Stanford NLP Group. The dataset is available for download at: https://nlp.stanford.edu/sentiment/index.html#download

Statistics The state-of-the-art models on the SST-2 dataset perform well and achieve accuracy scores near 100%. BERT-large got 94.9%, XLNet got 97% and the current STOA T5-11B got 97.5% [16].

Stanford SST-5 (fine-grained)

Description The Stanford Sentiment Treebank (SST-5) is a dataset of movie reviews and their corresponding sentiment labels. Similar to the SST-2 dataset, the reviews are taken from the Rotten Tomatoes movie review website. However, the SST-5 dataset provides a more fine-grained sentiment label for each review, with labels ranging from very negative to very positive.

Note that in this task, even the STOA models fail to reach 60% accuracy. It is likely that unlike previous binary classification tasks, where it is relatively easy to make the decision of whether it is "good" or "bad", the model can be confused if a comment is "very" or "somewhat" negative or positive. This may be due to individual differences among annotators, where they may have varying standards of the degree of sentiment.

An example input for this dataset would be a movie review such as "The acting was mediocre, but the story was interesting, and the special effects were fantastic." The corresponding output would be a label of "2" on a scale of 0 to 4, with 0 being very negative and 4 being very positive.

Source of Data The SST-5 dataset was created by Stanford University's Stanford NLP Group. The dataset is available for download at: https://nlp.stanford.edu/sentiment/index.html#download

Relevant Statistics The state-of-the-art models on the SST-5 dataset typically achieve accuracy scores in the range of 50-59%. BERT achieved 55.5%, GPT-2 achieved 58.5% and RoBERTa achieved 59.8% [17].

Financial Sentiment Analysis

Description The financial sentiment analysis dataset is a combination of the FiQA and financial phrasebank datasets. It contains total 5322 financial sentences in which each are either classified as "positive", "negative" or "neutral".

An example input for this dataset would be "Tesco share price jumps as Q3 sales top estimates". The corresponding output would be a label of "positive".

Source of Data https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

Relevant Statistics Since the dataset is on Kaggle unlike previous datasets, there are not much information of current model's performance on it.

Approach

The aim of this project is to compare the performance of different models and word embeddings in sentiment analysis. Specifically, the results of Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Tiny-BERT (Tiny-BERT), BERT, and T5 models in classifying sentiments in the four different datasets introduced above were compared in the first part. Whereas GloVe and word2vec pre-trained embeddings were compared using LSTM and RNN for the second part.

Overall streamline

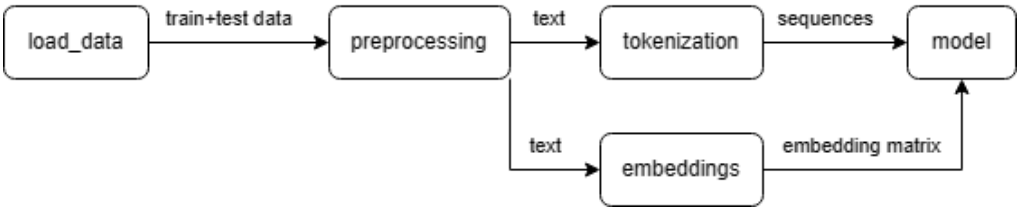

Figure 2: Training and testing pipeline

The figure above shows the training and testing pipeline for this work. All trainings and inference were performed on Kaggle using different notebooks upon different models/settings. LSTM and RNN models were trained on CPU, while all other models were were trained on GPU. Details of each model are included in the subsequent sessions.

Load Data

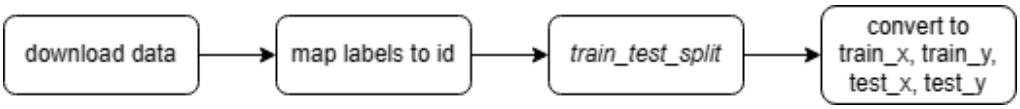

Figure 3: Load data pipeline

In the notebook, a load_data function is prepared for each dataset, namely load_imdb, load_sst2, load_sst5, and load_fin. The outputs of all functions are formatted consistently to ensure a streamlined subsequent analysis. Specifically, the output of each function consists of four dataframes with identical column names.

The train/test_x dataframes contain input string data, which are the sentences to be classified. Meanwhile, the train/test_y dataframes contain the corresponding sentiment labels. The mapping from text sentiment labels to numerical indices is shown below. Smaller indices are used for negative sentiment, while larger indices are used for positive sentiments. Such mappings were applied to all models except for T5, which is a sequence-to-sequence model that require text output, so the original labels were retained (except for SST-2, where the "very" was removed from "very negative" and "very positive").

Dataset/Label	0	1	2	3	4
sst2	(very) negative	(very) positive			
sst5	very negative	negative	neutral	positive	very positive
imdb	neg	pos			
fin	negative	neutral	positive		

Table 1: Text label to index mapping

For the data source, the imdb dataset is loaded from ai.stanford.edu and parsed using keras's get_file function. As for the SST dataset, since it is given as parse trees, it would be complicated and time-consuming to download and parse them each time. Therefore, they were first converted into csv files locally, then uploaded online for the training notebooks to retrieve, which significantly simplified the code.

For the train and test data distribution, to allow a fair comparison with benchmark results, the default train-test-split were used for IMDB and SST datasets. For the financial dataset, the train_test_split function of scikit-learn was applied with default configuration, which splits 25% of the data as testing data.

Preprocessing

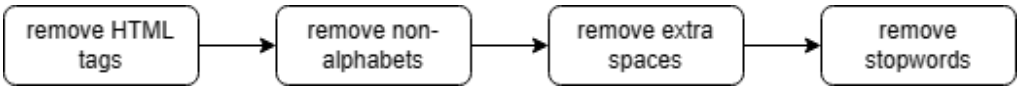

Figure 4: Preprocessing pipeline

The above figure shows the text preprocessing steps. The first three steps were performed using regex, while the nltk's stopwords corpus was used for stopwords removal.

Tokenization and Embeddings

Model Comparison

The comparison of different model architectures was conducted without the use of pre-trained word embeddings, except for CNN models since they failed to learn under without the embeddings. The tokenization and embeddings of CNN are the same as in the "embeddings comparison" section below. To prepare the input text for the RNN, and LSTM models, the TextVectorization function in Keras was utilized to convert the text data into sequences. The sequences were then processed by the Kera's Embedding layer added as the first layer of each model. The parameters of the TextVectorization function are shown below:

Parameter	Value
max_tokens	10000
output_mode	int
output_sequence_len	128

Table 2: TextVectorization Parameters

For BERT and Tiny-BERT models, the pre-processing layer provided with the pre-trained layer in the model was applied to the text data. Consequently, no tokenization was performed, and the text was passed directly to the models after basic pre-processing.

In the case of T5, the default tokenizer that comes with the pre-trained model was employed to tokenize the input sentences. The maximum sequence length was capped at 128 to ensure consistency with the other models.

Embeddings Comparison

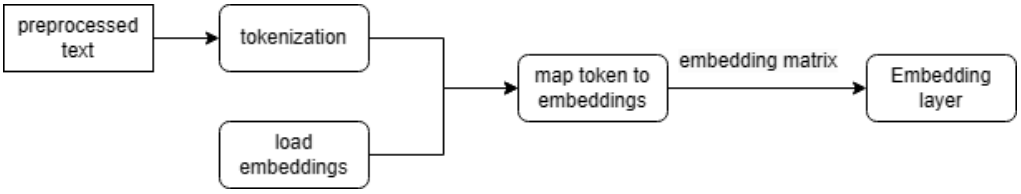

Figure 5: Embeddings generation pipeline

To utilize pre-trained word embeddings, they are first downloaded, with the dimensionality varying depending on the specific embeddings used. The text data is then preprocessed and tokenized using Keras' tokenizer. Each token in the tokenizer is then mapped to its corresponding embedding and appended to the embedding matrix. If a token exists in the pre-trained embeddings, its embedding is used; otherwise, a random vector of the same dimensionality as the embeddings is generated.

The resulting embedding matrix is passed to the Embedding layer of the model. This enables the layer to convert the tokenized sentences into the corresponding vectors, which can then be passed to subsequent layers in the model for further processing.

The first two embeddings used are the 100-dimension and 200-dimension versions of GloVe 6B. It contains English word vectors pre-trained on the combined Wikipedia 2014 + Gigaword 5th Edition corpora (6B tokens, 400K vocab). It could be accessed at https://www.kaggle.com/datasets/rtatman/glove-global-vectors-for-word-representation.

The third embedding used were the google news word2vec embeddings with 300 dimensions. It contains pre-trained vectors trained on a portion of the Google News dataset (consisting of about 100 billion words) for about 3 million words and phrases. It could be accessed at https://www.kaggle.com/datasets/sugataghosh/google-word2vec.

Model Architecture

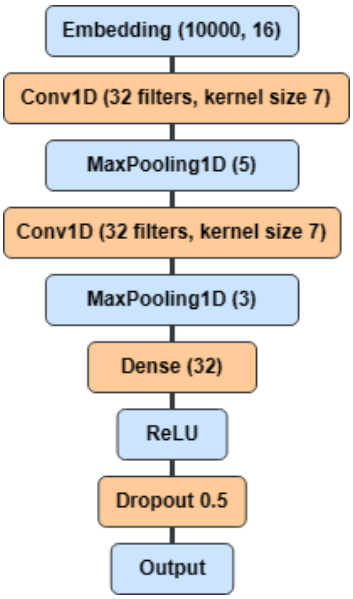

Figure 6: CNN

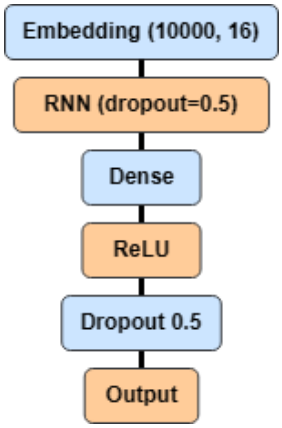

Figure 7: RNN

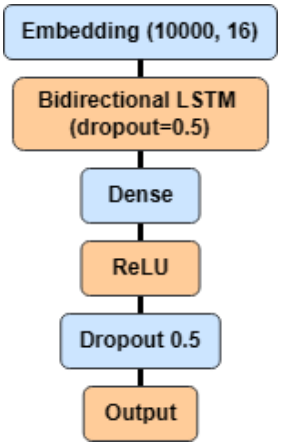

Figure 8: LSTM

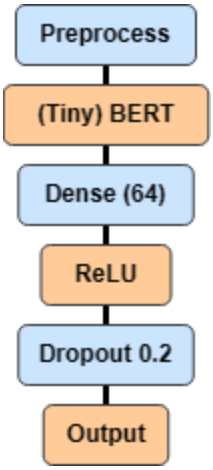

Figure 9: BERT

The figures above show the model architecture of CNN, RNN, LSTM and BERT models. All models above were built using Keras's layers components. Note that the whole T5 model was loaded for fine-tuning, so no changes were made to its structure.

For RNN and LSTM, several combinations of the number of nodes in the RNN/LSTM and Dense layers were tested. The best result for RNN was to use 32 RNN nodes and 16 dense nodes. While that for LSTM is 16 LSTM nodes and 16 dense nodes.

As for BERT, the BERT and preprocess were loaded using Tensorflow Hub's Keras layer. Both were preprocessed using bert_en_uncased_preprocess. The BERT layer applied was bert_en_uncased_L-12_H-768_A-12 and tiny BERT layer was small_bert/bert_en_uncased_L-2_H-128_A-2.

For T5, the pre-trained model t5-small is loaded from HuggingFace [19]. It is used instead of regular T5 due to computational resource constraints on Kaggle.

Hyperparameters

Optimizer

For the RNN, LSTM, and CNN models, Keras' Adam optimizer with a default learning rate of 0.001 was used. The Adam optimizer is an adaptive learning rate optimization algorithm that is well-suited for large datasets and noisy problems.

For the BERT model, the AdamW optimizer was used, which is a variant of Adam that uses weight decay regularization to prevent overfitting. The initial learning rate was set to 3e-5, with the number of warmup steps was set to 10% of the number of training steps.

For the T5 model, AdamW optimizer was also used. The initial learning rate was set as 3e-4 with 0 warmup steps. The weight decay rate was set to 0.01 with a linear learning rate schedule.

It is notable that the AdamW optimizer was tested for the RNN, LSTM and CNN too, but there was no observable improvement, so it was not adopted in the final version.

Metrics

Accuracy was the main metric used to interpret and evaluate the model performance. As for loss calculation, the sparse categorical cross-entropy loss was used for the SST-5 and fin dataset, since they have 5 and 3 classes respectively. While the binary cross-entropy loss was used for SST-2 and IMDB dataset since they have 2 classes only with labels 0 and 1.

Experiments

Model Comparison

In this section, the models, with their configuration same as that mentioned in the previous section, were tested on all datasets. For CNN, RNN and LSTM, since they were stopped after the validation accuracy dropped for two epochs, there was no fixed number of training epochs. In general, they lasted for three to eight epochs.

However, it is observed that the models overfit quite early, and the validation accuracy remains generally unchanged with an increasing number of training epochs. Meanwhile for the pre-trained models, larger models like BERT and T5 are less prone to overfitting, but the smaller Tiny-BERT tends to overfit soon too.

Another thing to note is that due to the large size of IMDB data (in terms of both number of data and average data length), T5 failed to run due to memory error, so no results were available.

The table below shows the accuracy of the trained models on the testing set:

Model/Dataset	**IMDB**	**SST-2**	**SST-5**	**Fin**
CNN	0.8333	0.7738	0.3502	0.6646
RNN	0.5001	0.4992	0.2308	0.5339
LSTM	0.8336	0.7545	0.3801	0.6591
Tiny-BERT	0.8538	0.7716	0.4059	0.6352
BERT	0.8856	0.8878	0.4534	0.7303
T5	N/A	0.9108	0.4393	0.8669

Table 3: Accuracy on various datasets for different models

Taking an overview, we can see that the best-performing model on average is T5, with a dominating accuracy of 0.9108 on the SST-2 dataset and an accuracy of 0.8669 on the Fin dataset. It was followed by BERT and Tiny-BERT, which were unsurprising given that they were pre-trained. While CNN, using the 100d GloVe embeddings, achieved results similar to LSTM. Finally, RNN performed the worst among all. There are few insights that could be made from the results:

First, pre-training can significantly improve model performance. Unsurprisingly, large pre-trained

models trained on many corpus and datasets significantly outperformed other models trained from scratch. CNN, after being equipped with pre-trained embeddings, also improved significantly from failing to learn at all to scoring comparably to LSTM. It is reasonable since they had prior knowledge to the meaning and semantics of words from their pre-training, which gave them a significant edge on down-stream text classification tasks, especially when the size of the datasets used for fine-tuning are very limited compared to their pre-training corpus.

Second, T5 model performed significantly better at SST-2 and Fin, but it performed worse than BERT for SST-5. An important reason is that T5 is pre-trained on the SST-2 dataset, so it is good at SST-2 and Fin, which is a three-class classification task that is similar to SST-2. Another possible explanation for the worse SST-5 performance is that since T5 is a sequence-to-sequence model (as opposed to other models that output an integer instead), it may not capture the increasing level of "positiveness" of the labels compared to the integer outputs that use 0 to 4 for increasing for increasingly positive sentences. Also, its judgement could be interfered by the SST-2 dataset it pre-trained on, since for the same sentence, its label is "very positive" in SST-5, but "positive" in SST-2.

When compared to the benchmarks, most information available were the pre-trained model performances of the SST-2 dataset. For T5, its performance on SST-2 in the original paper was 0.918, which is sufficiently similar to the results obtained. Also, the performances of BERT and Tiny-BERT on SST-2 were 0.9 and 0.832 respectively. The results obtained here were generally slightly worse, which could be due to insufficient hyperparameter tuning, text preprocessing and selection of dictionary size or embedding dimensions. Moreover, the performance using word2vec on SST-2 is 0.8, which is slightly better than CNN that used the best embeddings out of the three options.

Word Embeddings Comparison

The LSTM and RNN models were used to test the word embeddings. Note that NN was also attempted, but it failed to converge properly, and so was excluded in the experiment.

Before selecting the embeddings, a model architecture tuning was performed, which tested the results of number of nodes in RNN/LSTM and dense layer.

Model/Dataset	**IMDB**	**SST-2**	**SST-5**	**Fin**
LSTM + GloVe100	0.8722	0.8171	0.4158	0.6783
LSTM + GloVe200	0.8493	0.7792	0.3878	0.6578
LSTM + word2vec	0.8689	0.7748	0.4049	0.6522
RNN + GloVe100	0.7281	0.8034	0.3900	0.5813
RNN + GloVe200	0.7014	0.7471	0.3352	0.5313
RNN + word2vec	0.7740	0.7243	0.3610	0.6071

Table 4: Accuracy on various word embeddings

Overall, the performance of the three embeddings was similar, with Glove100 achieving the best results. One possible reason is that the dataset employed is pretty small, so using higher dimension embeddings may over-complicate the data, making it in turn harder for the model to learn.

Conclusion

Overall, the project was successfully performed. Some possible future works including exploration of transfer learning and fine-grained sentiment analysis.

References

[1] Turney, P. (2002) Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews, arXiv.org. Available at: https://arxiv.org/abs/cs/0212032 (Accessed: 18 March 2023).

[2] NLTK: nltk.classify.naivebayes (2023). Available at: https://www.nltk.org/_modules/nltk/classify/naivebayes.html (Accessed: 18 March 2023).

[3] Nandwani, P. and Verma, R. (2021) "A review on sentiment analysis and emotion detection from text", Social Network Analysis and Mining, 11(1). doi: 10.1007/s13278-021-00776-6.

[4] "Recursive deep models for semantic compositionality over a sentiment treebank," Deeply Moving: Deep Learning for Sentiment Analysis. [Online]. Available: https://nlp.stanford.edu/ socher-r/EMNLP2013_RNTN.pdf. [Accessed: 18-Mar-2023].

[5] Krizhevsky, A., Sutskever, I. and Hinton, G. (2012) "ImageNet Classification with Deep Convolutional Neural Networks", Advances in Neural Information Processing Systems, 25, p. Available at: https://papers.nips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (Accessed: 22 April 2023).

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997,doi: 10.1162/neco.1997.9.8.1735.

[7] Devlin, J. et al. (2018) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv.org. Available at: https://arxiv.org/abs/1810.04805 (Accessed: 22 April 2023).

[8] Mikolov, T. et al. (2013) Efficient Estimation of Word Representations in Vector Space, arXiv.org. Available at: https://arxiv.org/abs/1301.3781 (Accessed: 18 March 2023).

[9] TensorFlow Hub (2023). Available at: https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1 (Accessed: 18 March 2023).

[10] TensorFlow Hub (2023). Available at: https://tfhub.dev/google/nnlm-en-dim50/2 (Accessed: 18 March 2023).

[11] TensorFlow Hub (2023). Available at: https://tfhub.dev/google/universal-sentence-encoder/4 (Accessed: 18 March 2023).

[12] "Papers with code - SST-5 fine-grained classification benchmark (sentiment analysis)," The latest in Machine Learning. [Online]. Available: https://paperswithcode.com/sota/sentiment-analysis-on-sst-5-fine-grained. [Accessed: 18-Mar-2023].

[13] Pennington et al. (2014) GloVe: Global Vectors for Word Representation]. Available at: https://aclanthology.org/D14-1162 (Accessed: 22 April 2023).

[14] Shazeer, N. et al. (2016) Swivel: Improving Embeddings by Noticing What's Missing, arXiv.org. Available at: https://arxiv.org/abs/1602.02215 (Accessed: 18 March 2023).

[15] Papers with Code - IMDb Benchmark (Sentiment Analysis) (2023). Available at: https://paperswithcode.com/sota/sentiment-analysis-on-imdb (Accessed: 18 March 2023).

[16] Papers with Code - SST-2 Binary classification Benchmark (Sentiment Analysis) (2023). Available at: https://paperswithcode.com/sota/sentiment-analysis-on-sst-2-binary (Accessed: 18 March 2023).

[17] Papers with Code - SST-5 Fine-grained classification Benchmark (Sentiment Analysis) (2023). Available at: https://paperswithcode.com/sota/sentiment-analysis-on-sst-5-fine-grained (Accessed: 18 March 2023).

[18] TensorFlow Hub (2023). Available at: https://tfhub.dev/google/collections/bert/1 (Accessed: 18 March 2023).

[19] HuggingFace (2023). Available at https://huggingface.co/t5-small (Accessed: 22 April 2023).