Final Project – (C) Sentiment Analysis

CZ4042 Neural Networks & Deep Learning

Name	Matric No	Contribution
Gupta Jay	U1822549K	33%
Kanodia Ritwik	U1822238H	33%
Mundhra Divyesh	U1822168E	33%

School of Computer Science & Engineering

Nanyang Technological University, Singapore

Introduction

Natural Language Processing remains one of the toughest tasks for computers, both in terms of accuracy and computing power. Since 2012 there has been a "neural" tsunami in NLP; traditional methods (e.g., Markov models) have been replaced with deep learning methods (e.g., large-scale neural nets. Instead of operating over discrete linguistic units (words, phrases), they are now represented with "learnable" vectors and operate over those vectors paired with discriminative training methods (e.g., stochastic gradient descent) as opposed to generative modelling.

In this project, we have tackled the third problem statement given on sentiment analysis in the assignment handout. More specifically, we will examine different types of neural network architectures and their effects on model accuracy, computation time, and efficiency. Furthermore, we have addressed the two additional sub-problems provided as follows:

- To deal with domain adaptation, that is, how can one adapt a network train on one domain to work in another domain.
- To deal with small datasets, that is, with insufficient number of training samples.

Existing Techniques

There are several types of existing neural network architectures specifically designed to handle sequential data for natural language processing tasks. We have experimented on the following networks for our analysis.

- **RNN –** Recurrent Neural Networks
- **LSTM –** Long Short-Term Memory Networks
- **Bi-Directional LSTM** – Bidirectional Long Short Term Memory Networks
- **GRU –** Gated Recurrent Units
- **CNN –** Convolutional Neural Networks
- **Transformers –** BERT Model with the Transformer Architecture with Attention

Methods

(1) Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are designed to work with sequential data. Simple Feed Forward Neural Network with Dense Layers are not able to consider the sequence of an input and that's where RNN helps us to use the previous information in the sequence to produce the latest output.

At each time step, the network is fed with the current input and the activation value from the previous step. This way the input in the previous step also plays a role in determining the output at the current step. The main problem with RNN is that it is difficult for the network to preserve information over a long range of timesteps. To deal with this issue, we have used GRUs and LSTM for better performance.

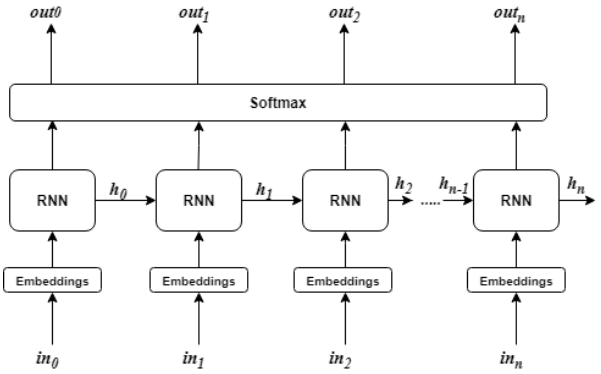

Figure 1: Illustration of a Recurrent Neural Network (RNN) architecture for sentence classification.

In this paper, we discuss the use of two such RNN with 100 and 500 embedding dimensions along with dropout. Along with the embedding layer, the model consists of a 32 unit RNN layer with dropout, and a fully connected layer of 1 unit, which acts as our softmax layer.

L #	Layer	Property
1	Embedding	100/500 Dim
2	Dropout	rate = 0.25
3	Simple RNN	32 units
4	Dropout	rate = 0.25
5	FC	1 Unit (Output)

Table 1: RNN Model Architecture

(2) Gated Recurrent Units (GRU)

Gated Recurrent Unit (GRU) is a similar network structure like RNN but the operation inside each GRU cell is a bit different. Each cell has two gates: The reset gate and the update gate. These gates decide what information should be allowed through the next stage and hence it can be trained to just retain information which are important in a long chain of sequence. The reset gate is used to decide if the previous cell state is important or not. The previous cell state is carried ahead if only it is termed as important by the reset gate.

The update gate decides if the cell state should be updated with the current activation value or not. This allows us to pass only relevant information through a series of steps and hence leads to better prediction results.

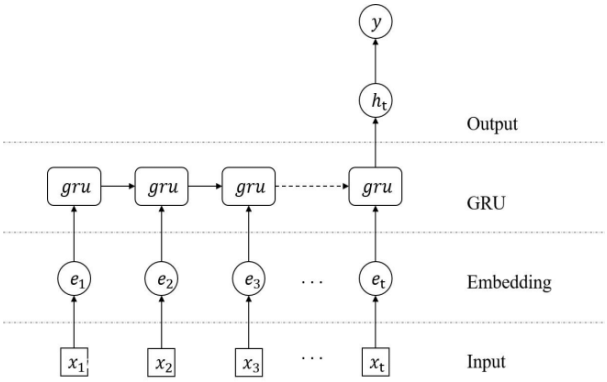

Figure 2: Illustration of a Gated Recurrent Neural Network (GRU) architecture for sentence classification.

In this paper, we discuss the use of two such RNN with 100 and 500 embedding dimensions along with dropout. Along with the embedding layer, the model consists of a 32 unit GRU layer with dropout, and a fully connected layer of 1 unit, which acts as our softmax layer.

L #	Layer	Property
1	Embedding	100/500 Dim
2	Dropout	rate = 0.25
3	GRU	32 units
4	Dropout	rate = 0.25
5	FC	1 Unit (Output)

Table 2: GRU Model Architecture

(3) Long Short Term Memory Units (LSTM)

Long Short Term Memory (LSTM) is similar to GRU but with two additional gates, which are forget gate and output gate. LSTM is a comparatively more complex model than GRU and hence performs better in cases with long sequence chains. GRU on the other hand is much more efficient and performs better than LSTMs on less training data. The forget gate in LSTM is used to decide how much information from the previous state should be kept and the rest is forgotten. The output gate controls which parts of the cell are output to the hidden state and hence determine what the next hidden state will be.

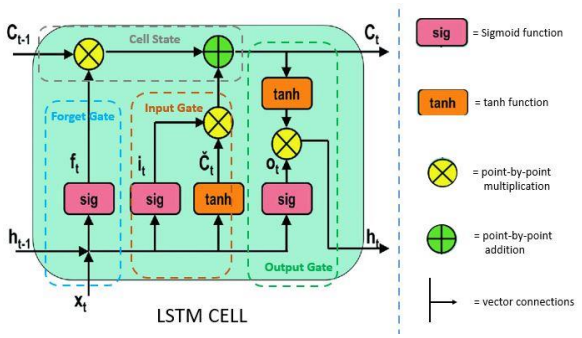

Figure 3: Illustration of an LSTM cell with its 4 gates.

In this paper, we discuss the use of two such LSTM with 100 and 500 embedding dimensions along with dropout. Along with the embedding layer, the model consists of a 32 unit LSTM layer with dropout, and a fully connected layer of 1 unit, which acts as our softmax layer.

L #	Layer	Property
1	Embedding	100/500 Dim
2	Dropout	rate = 0.25
3	LSTM	32 units
4	Dropout	rate = 0.25
5	FC	1 Unit (Output)

Table 3: LSTM Model Architecture

(5) Bi-Directional LSTM

Bidirectional LSTMs are an extension of traditional LSTMs which improve model performance in some cases. This architecture trains two instead of one LSTMs on the input sequence. The first training is done on the input sequence and the second training is done on a reversed copy of the input sequence. This helps the model to provide additional context to the network and hence a better learning capability for the model.

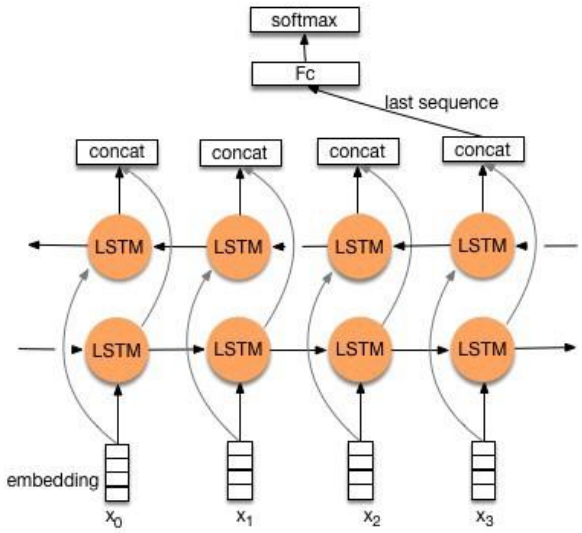

Figure 4: Illustration of a Bi-directional LSTM architecture for sentence classification.

In this paper, we discuss the use of two such Bi-directional LSTM with 100 and 500 embedding dimensions along with dropout. Along with the embedding layer, the model consists of two 64 unit Bi-directional LSTM layer with dropout, and a fully connected layer of 1 unit, which acts as our softmax layer.

L #	Layer	Property
1	Embedding	100/500 Dim
2	Dropout	rate = 0.25
3	Bi-directional LSTM	64 units
4	Bi-directional LSTM	64 units
4	Dropout	rate = 0.25
5	FC	1 Unit (Output)

Table 4: Bidirectional LSTM Model Architecture

(6) Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are typically used for computer vision tasks such as image classification and object detection. However, findings by many authors including Britz D [1], Kim Y [2], Zhang et al. [3] proved that CNNs can be used for tasks in natural language processing and showed promising results. In computer vision tasks, the filters slide over the local patches of the images to learn its features, whereas in NLP tasks, the filters slide over the words or word embeddings.

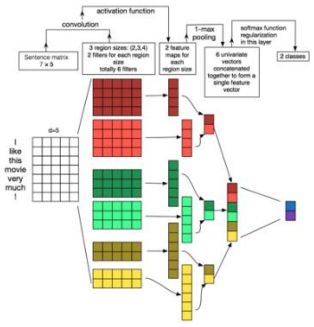

Figure 5: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification

In this paper, we discuss the use of one such CNN, consisting of a Word2Vec embedding layer, three 1D convolutional layers, and two fully connected layers, as described below.

L #	Layer	Property
1	Embedding	Word2Vec, 300 Dim
2	1D Conv. w. Max Pool	200 Filters, Size 2
3	1D Conv. w. Max Pool	400 Filters, Size 3
4	1D Conv. w. Max Pool	800 Filters, Size 4
5	FC w. Dropout	512 Units
6	FC w. Dropout	128 Units
7	FC	2 Units (Output)

Table 5: CNN Architecture

(7) Transformers

Transformers are Neural networks which follow a sequence to sequence architecture allowing them to handle long range dependencies in sentences easily. They are made of a sequence of encoder and decoder layers and rely completely on self-attention to compute input sentence embeddings.

Bidirectional Encoder Representation from Transformers is a transformer-based machine learning technique for NLP developed by Google which only uses the encoder layers of the transformer architecture along with the self-attention mechanism. The original English-language BERT has two models , BERT base and BERT large. The model are pretrained from unlabelled data extracted from the BookCorpus with 800M words and English Wikipedia with 2,500M words [4]. Since its inception, a variety of modifications of the model have come up including smaller versions like BERT small with lower embedding size offering lower training and inference time with minimal drop in performance. BERT has some specialised input requirements which requires pre-processing such as tokenisation and mask generation. For ease of implementation we are using the pretrained model from TensorFlow Hub which can be fine-tuned . It also offers a Pre-processing layer which pre-process the input text automatically before feeding it to the BERT Encoder.

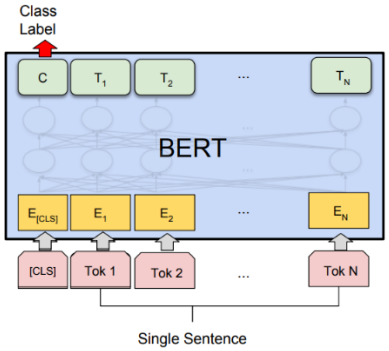

Figure 6: Illustration of BERT architecture for sentence classification

In this paper, we discuss the use of two BERT models, BERT base and BERT small with 516 and 768 embedding dimensions. We further add a fully connected layer with Dropout and with a single neuron to make binary classification.

L #	Layer	Property
1	Input	Sentences
2	Pre-processing	Tokenization,mask etc.
3	BERT Encoder	BERT embeddings
4	Dropout	rate=0.1
5	FC	1 Units (Output)

Experiments & Results

All the neural network architectures described in the 'Methods' section were ran on two datasets: IMDB Movie Reviews and Twitter. Both the datasets are chosen from different domains and vary in size significantly.

Dataset	Classes	Train Samples	Test Samples
IMDB	2 (Pos., Neg.)	25,000	25,000
Twitter	2 (Pos., Neg.)	1,120,000	40,000

Table 6: Datasets used for model evaluation

The results are listed in ***Table 7*** and ***Table 8***, with further discussions in the next section.

Discussion

Based on the tables below and in the appendix, we observe that we have experimented with the number of epochs for IMDB dataset as the number of data samples are less as compared to the Twitter dataset. IMDB dataset showed comparable results with 3 and 10 epochs, but the Twitter dataset gave worse results with more than 3 epochs. The results obtained here correlate with the theory we have studied for the RNN based modes:

RNN, GRU, & LSTMs

- We can see that GRU, and LSTM give significantly better results than Simple RNN model as they are able to learn long term dependencies.
- We can observe that GRU model performs better than LSTM model for IMDB dataset but worse for Twitter. This is because IMDB dataset has comparatively very less data than the Twitter dataset. Hence, LSTM can perform better for Twitter dataset based on their ability to retain more information in larger datasets.
- The training time of GRU is lesser than LSTM which signifies its efficient computation methodology.
- Bi-directional LSTM model performs better for the Twitter dataset but worse in IMDB dataset. This is because the model overfits on the smaller number of training examples for IMDB dataset because of the huge number of parameters to be trained. On the other hand, the large number of parameters trained help us in the case of twitter dataset as can be seen through the results.

CNNs

- The CNN model gives worse performance than all models since it is unable to capture the long-term sequential dependencies in the text sequences. However, it is noteworthy that the model gives a reasonable performance on both the datasets, given it takes about fifteen minutes to train on the Twitter dataset and less than a minute on the IMDB dataset, with even lower inference times.
- When a CNN is used for NLP tasks, the filters slide over the word embeddings rather than the patches of images in Computer Vision tasks. Location Invariance and local Compositionality made intuitive sense for images, but not so much for NLP [1]. Although in some cases, words appear closer to each other may be semantically related, it is not always true, especially for long sentences. As the CNN model cannot capture long-term dependencies, it is unable to outperform models based on recurrent units, attention models, or transformers.
- A big advantage of CNN based NLP models is their training and inference times. Due to the shared-weight architecture of convolutions and their filters, the number of parameters required to learn significantly reduces. This property makes CNN networks useful in industrial cases where the inference time is critical to operations and a tradeoff balance can be achieved between accuracy and train/inference times.

Transformers

- The BERT Small and BERT base model outperforms all the other models in terms of accuracy in both the IMDB as well as the twitter dataset offering almost state of the art results. The BERT base model in general offers better accuracy compared to BERT small as it learns more embeddings (768 vs 512) but takes longer to train and infer.
- In the Twitter dataset, the Bi-directional LSTM offers comparable results than BERT small with much lower training and inference times. The BERT Base model outperforms by ~1% it but at the cost of an unrealistic train time of 750 minutes. This is because the BERT model was trained on a huge dataset when it could have given similar results on being trained on a smaller portion of the dataset in lesser time.
- In the IMDB dataset, BERT small offers pleasant results, outperforming all models in terms of accuracy with a smaller train and inference time.
- The BERT base models significantly perform better by more than ~5% taking reasonable train and inference time.

Conclusion

Based on all the discussion points mentioned in the section above, we can gauge the performance of different NLP models and their respective training and inference times.

In general, it is observed that RNNs, and the different variants of RNNs such as GRUs, LSTMs, and Bi-Directional LSTMs take longer to train as compared to other models such as CNN and BERT Transformer model. This is reasonable because RNNs have signals traveling both forward and back and may contain various "loops" in the network where numbers or values are fed back into the network. Due to this property, the training is not parallelizable, and thus, they are not computationally efficient. On the other hand, CNNs took extremely less time to train and infer new data as compared to all models tested. This is due to the concept of shared weights which reduces the training time significantly.

Specifically, on the Twitter Dataset, the BERT model outperforms all other models in terms of accuracy. However, due to its heavy architecture, it takes longer to train and infer as well. The performance of GRU, LSTM, and Bi-directional LSTM is similar in the range of ~80%, whereas the accuracy of the CNN model is the lowest at 74%.

On the IMDB Dataset, the performance of the models is mostly similar across all model architectures except RNNs, which perform the worst despite having a high training time.

In conclusion, we observe that RNNs and its variants take longer to train than other architectures. The training time of CNNs are the lowest due to their shared-weight matrix architecture.

Twitter Dataset

Model	Epochs	Learning Rate	Test Loss	Test Accuracy %	Train Time (m)	Infer Time (m)
RNN	3	0.001	0.512	77.13	103.7	0.75
GRU	3	0.001	0.473	78.93	92	0.78
LSTM	3	0.001	0.470	79.11	94	1.35
Bi-directional LSTM	3	0.001	0.491	79.60	110.45	1.97
CNN	5	0.0002	0.933	71.14	17.27	1.4
BERT Small	3	3e-5	0.42	79.68	200	8
BERT Base	3	3e-5	0.73	82.47	750	75

Table 7: Model Evaluation on the Twitter Dataset

IMDB Dataset

Model	Epochs	Learning Rate	Test Loss	Test Accuracy %	Train Time (m)	Infer Time (m)
RNN	10	0.001	0.771	79.9	59.38	0.67
GRU	10	0.001	0.617	86.0	9.89	0.68
LSTM	10	0.001	0.550	85.5	10.38	0.38
Bi-directional LSTM	10	0.001	0.552	85.1	50.45	2.37
CNN	10	0.00025	1.23	83.3	0.45	0.03
BERT Small	3	3e-5	0.69	85.4	10	2
BERT Base	3	3e-5	0.73	88.4	58	8

Table 8: Model Evaluation on the IMDB Dataset

Domain Adaptation

The second aspect we have studied is to deal with domain adaptation, that is, how can one adapt a network trained on one domain to work in another domain.

The fundamentals of domain adaptation arise from the concept of Transfer Learning. It is a process where the knowledge obtained to solve one problem is fine-tuned and reused to solve another problem. In deep neural networks, it is generally observed that the initial few layers learn the more specific features of the training samples, whereas the final layers tend to learn more generic patterns. These generic patterns are used to solve other similar problems.

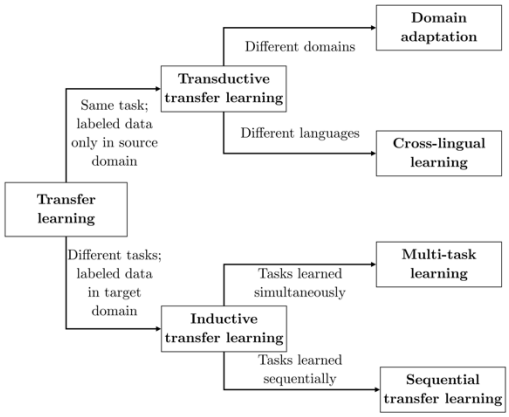

Figure 7: Taxonomy in NLP

In domain adaptation, the task of the model remains fixed, i.e., in our case, binary sentiment analysis and the language remains the same as well. Domain Adaptation in NLP tasks is largely dependent on the word embeddings used in model architecture.

Generally, well-known pre-trained word embeddings like BERT perform well on nearly all types of domains. This is due to a variety of factors. Firstly, some models like BERT-Large is trained on a huge volume of data, close to 2,500 million words and the embeddings are generated in a context dependant manner, i.e. a word having multiple meanings is well captured by the embeddings generated due to BERT's attention and transformer architecture.

Although, the word embeddings work relatively well on different domains, the performance of these models drop if they are used on niche domains with specific taxonomies like medicine and law, and can be better if the embeddings are trained specific to the domain of the problem.

Embeddings	Training Corpus
Word2Vec	Google News
GloVe	Wikipedia 2014, Twitter, Gigaword, Common Crawl
ELMo	1 Billion Word Benchmark, Wikipedia, Monolingual World Crawl
BERT	BooksCorpus, English Wikipedia

Table 9: Common Pretrained Word Embeddings and their Training Corpuses

Most word embeddings are trained on a large corpus of general corpora which contains commonly occurring words. Initially, models like Word2Vec and GloVe contained 'context-free' embeddings, i.e., the embedding does not take into account the word order during training. To solve the problem of representing words that have more than one meaning when used in different context, models like ELMo and BERT were introduced which try to capture the context of the sentence when producing its embedding.

Even with improvements, domain adaptation remains a challenge since the word embeddings will be of poorer quality when domain specific low-frequency words are encountered, e.g., Anosognosia, Bradykinesia, Trichotillomania from the medical field.

Approach #1: Hyperparameter Tuning

Tuning the hyperparameters of a pre-trained word embedding model has become a standard practice in the NLP research space. Fine-tuning is done in a few ways. Firstly, a few layers of the model is freezed and trained on the target text corpus. Secondly, tuning the number of word-embedding dimensions to fit the target domain is also beneficial.

Approach #2: Use Domain Specific Data

To tackle the issue of domain adaption, another solution is to re-train the embeddings on domain specific data. This has been done in Word2Vec embeddings. Word vectors for bioinformatics known as BioVectors to be used in biological sequences (e.g. DNA, RNA, and Proteins) for bioinformatics applications has been proposed by Asgari and Mofrad. The limitation of this method is that it is expensive to obtain domain specific corpora since they are not commonly available on the internet.

Dealing with Small Datasets

The third aspect we look into is to deal with the problem of smaller datasets in the context of Natural Language Processing tasks. Deep learning models have shown exponential increase in performance with addition of more data from time to time. Hence here we will look at how to obtain augmented data for our use case.

Approach #1: Back Translation

In this technique we first convert a sentence into a different language using a model and then convert it back to the target language. Doing this, the new sentence meaning, and structure almost remains the same as the original one, but some of the words are replaced with different ones.

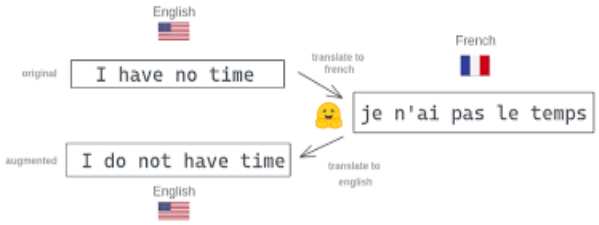

Figure 8: Illustration of Back Translation

For our experiments we will be converting some of the original movie reviews into German and then convert it back to English. We will use the transformers library [5] provided by python, to get the translator models. We use the T5-base model [6] for English to German translation and then use Bert2Bert model [7] for German to English translation.

Results

For this experiment, we use a Gated Recurrent Unit (GRU) Architecture. We train the model for 3 epochs as it seemed to give the best result in our earlier experiments as well.

L #	Layer	Property
1	Embedding	100/500 Dim
2	Dropout	rate = 0.25
3	GRU	32 units
4	Dropout	rate = 0.25
5	FC	1 Unit (Output)

Table 10: Model Architecture for use in Back Translation

# Augmented Reviews	50	100	50	100
Size of Training data before Augmenting	1000	1000	25000	25000
Accuracy before Augmenting	72.89	72.89	87.14	87.14
Size of Training Data after Augmenting	1050	1100	25050	25100
Accuracy after Augmenting	75.31	76.43	87.21	87.24

Table 11: Model Performance before and after Augmentation

Approach #2: Random Replacement

We can use the same two steps of Random insertion but instead of inserting new words we can replace old words with their synonyms. To do so using Transformers, we can use the pipeline functions provided by the transformer's library.

Let us have a look at the steps [8]:

- *Make a copy of the original movie review*
- *Generate a random integer between 0 and length of the review*
- *Replace the word at the random integer with '[MASK]' in the copied review string*
- *We will then feed this masked string into the pipeline function using the following definition: pipeline('fill-mask', model='bert-base-cased')*
- *This will give us a new string where the '[MASK]' word will be replaced by a new word which is a synonym of the same word in the original review.*
- *We then add this new string into our database with the same label as the original review.*

Below are some of the augmented text that we were able to derive through our experiments:

- 'The **heroin** addicted mafia → 'The **drug** addicted mafia'.
- 'The movie tells the **story** of the four hamilton siblings'. → 'The movie tells the **tale** of the four Hamilton siblings'.

We didn't include the results for this experiment as there was no significant improvement using this method. This can be attributed to the fact that the review lengths in our database are very long and replacing just one word is not enough to give us augmented data. Future work in this area can be to try and replace around 10 words within a particular review and consider that string as our new addition in the database.

Approach #3: Random Insertion

In this technique we do sentence augmentation by randomly inserting words in a sentence. It can be broadly divided into 2 steps.

Given a sentence, in step 1, we can randomly insert new words and generate 'n' new sentences. A better approach would be to language models like BERT which are often used for sentence generation to insert words randomly based on some context.

In step 2, we can filter 'm' out of 'n' sentences (m<=n) by calculating the similarity between the augmented sentences and the original sentence. The 'm' augmented sentence which satisfy a similarity threshold is then chosen. The similarity can be calculated using cosine similarity of embeddings from models like Universal Sentence Encoder.

This gives us 'm' new sentences in our database for each sentence.

Approach #4: Using Pre-trained models

Using pre-trained models often proves to be a great approach to deal with small datasets as these models have already been trained on huge corpuses and generalize well over general classification tasks. For the sake of our project, since we were already dealing with pre-trained models of BERT, we decided to see how much its performance changes on varying the size of the train dataset. We experimented the Pre-trained BERT base model on the Twitter dataset. In the initial experiments, the BERT Base model was trained on the 60% of the dataset (960,000 samples) giving us a performance of 82.47% on the test dataset. To prove its efficiency with small dataset, in this experiment we limited the size of the train dataset to only to only 0.06%. Here are the results:

Model	Train size	Test Size	Test Acc %
BERT	60%	30%	82.47
BERT	0.06%	0.94%	78.54

Table 12: BERT model on different training sizes

We can see that even though we reduced the train dataset size by a factor of 1000, the performance of the BERT model is still quite good, and it is able to able to beat the RNN and GRU model trained on the entire train dataset.

References

[1] D. Britz, "Understanding Convolutional Neural Networks for NLP," WildML, 7 November 2015. [Online]. Available: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[2] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Empirical Methods in Natural Language Processing*, 2014.

[3] J. Z. Y. L. Xiang Zhang, "Character-level Convolutional Networks for Text Classification," in *Conference and Workshop on Neural Information Processing Systems*, 2015.

[4] Wikipedia, "BERT (language model)," 1 November 2021. [Online]. Available: https://en.wikipedia.org/wiki/BERT_(language_model).

[5] HuggingFace, "Transformers," 2020. [Online]. Available: https://huggingface.co/transformers/.

[6] HuggingFace, "T5," 2020. [Online]. Available: https://huggingface.co/transformers/model_doc/t5.html.

[7] HuggingFace, "Encoder Decoder Models," 2020. [Online]. Available: https://huggingface.co/transformers/model_doc/encoderdecoder.html.

[8] HuggingFace, "Source code for transformers.pipelines," 2020. [Online]. Available: https://huggingface.co/transformers/_modules/transformers/pipelines.html.

END OF ASSIGNMENT

Appendix

Twitter Dataset

Model	Embedding Size	Epochs	Learning Rate	Test Loss	Test Accuracy %	Train Time (m)	Infer Time (m)
RNN	100	10	0.001	0.53	74.2	26.37	0.68
GRU	100	10	0.001	0.46	77.3	34.38	0.64
LSTM	100	10	0.001	0.45	78.7	38.3	1.37
LSTM	100	10	0.001	0.45	78.7	38.3	1.37
Bi-directional LSTM	100	10	0.001	0.41	81.3	169.4	2.8
BERT Small	512	3	3e-5	0.42	79.7	200	21
		5	3e-5	0.42	79.7	330	21
BERT Base	768	3	3e-5	0.73	82.5	750	70
		5	3e-5	0.62	82.5	1100	70
CNN	300	10	6.25e-05	2.297	69.42	34.65	1.43
		5	0.0002	0.933	71.14	17.27	1.4
		2	0.001	0.542	73.36	7.05	0.99

Table 13: Results over varying epochs for all models on the Twitter Dataset

IMDB Dataset

Model	Embedding Size	Epochs	Learning Rate	Test Loss	Test Accuracy %	Train Time (m)	Infer Time (m)
RNN	500	10	0.001	0.771	79.9	59.38	0.67
		3	0.001	0.455	80.2	3.38	0.23
GRU	500	10	0.001	0.617	86.0	9.89	0.68
		3	0.001	0.334	86.7	3.6	0.67
LSTM	500	10	0.001	0.550	85.5	10.38	0.38
		3	0.001	0.345	86.5	4.01	0.38
BERT Small	512	3	3e-5	0.69	85.4	10	2
		5	3e-5	0.46	85.6	15	2
BERT Base	768	3	3e-5	0.73	88.4	58	8
		5	3e-5	0.62	88.5	92	8
Bi-directional LSTM	500	10	0.001	0.552	85.1	50.45	2.37
		3	0.001	0.331	86.1	23.43	3.35
CNN	300	10	6.25e-5	1.49	82.9	0.87	0.03
		5	0.00025	1.23	83.3	0.45	0.03
		2	0.001	0.366	84.1	0.30	0.03

Table 14: Results over varying epochs for all models on the IMDB Dataset