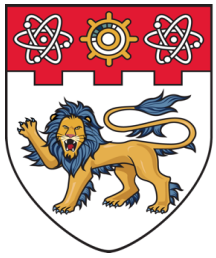


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 : Neural Networks and Deep Learning

Project: Sentiment Analysis - Comparative study on sentiment polarity detection using deep learning techniques

Agarwal Hitesh	(U1823360H)
Arya Shashwat	(U1822436J)
Hasan Mohammad Yusuf	(U1822478E)

Introduction

Sentiment analysis in Natural Language Processing (NLP) is a technique which is used to determine whether a piece of text is positive or negative. It is widely used by businesses to determine the social sentiment of consumers towards their products and services. There have been numerous studies on sentiment analysis using a wide range of algorithms to determine the outcome. In this paper, we focus on **(1) Examining the effect of model architecture and size on model performance, and (2) Reducing training time without compromising accuracy majorly**. We believe that our study would help corporations decide which models are best suited to their application depending upon their priorities.

We focus mainly on 3 types of base architectures: Convolutional Neural Networks (CNN), Long Short-Term Memory(LSTM) and Transformers. All three kinds of architectures have previously shown to give state of the art results in NLP. Each architecture has their way to learn the texts which leads to their respective performance.The CNN architecture aims to capture the temporal and spatial dependencies in the input data whereas the LSTM architecture is optimised to perform better on longer input sequences by referencing past context. The transformers are a unique architecture that does not use sequence-aligned RNNs or convolution to compute representations of its input and output, instead relying solely on self-attention and positional embeddings.

Review of existing techniques

There have been some studies in the past which aimed to compare multiple machine and deep learning techniques with respect to model performance. Dang et al. (2020) [8] compares various DNN, CNN and RNN models with respect to various benchmark datasets and word embedding techniques. Devika et al. (2016) [9] focuses on comparing Machine Learning techniques like SVM, NB, Maximum Entropy for sentiment analysis. Both these studies lack the inclusion of the transformers architecture which is a major addition to the deep-learning world.

Further studies have been done in the fields of RNNs and CNNs on techniques to reduce training time of these models. LSTM performs better and faster than traditional RNN models on longer sequences as its architecture allows for a memory unit that helps the LSTM model remember past context [4]. This however needs more training time as the whole input sequence has to be parsed in the model one token at a time.

Previous research shows that the RNN and LSTM models are not very efficient in handling long sequences. In the age of GPUs and TPUs, both models cannot be trained in parallel due to sequential training. CNNs are more successful with shorter sentences and train faster than RNNs [1]. The transformer model avoids recursion by using attention processes and positional embeddings to process phrases as a whole [2] and is independent of sequential word training, allowing for parallelism.

There are several techniques CNNs use to reduce training time. A very common approach is to use pre-trained word embedding like word2vec (Mikolov et al., 2013) [7] and GloVe as opposed to training them from scratch. Additionally, Zhang et al. (2015) [10] introduces character level embedding with large scale datasets which shows that such raw embedding forms help reduce training time for large datasets. However, there is a lack of research on whether this technique works on small datasets. A lot of corporate AI applications face the problem of having enough data and it would be worthwhile to see whether such techniques would work with less data or not.

The Transformer is a model architecture that seeks to solve sequence-to-sequence tasks while also resolving long-range dependencies. Although transformers models are being used in today's world due to their high performance compared to the other models, we need to also account for the training time required by the model to achieve the same performance.. There has been some work on decreasing the training times of the transformers. The approach was to use a larger transformer model compared to the base model but the training is stopped very early. This phenomenon occurs because larger models converge to lower test error in fewer gradient updates. This increase in convergence outpaces the extra computational cost of using larger models. Consequently, when considering wall-clock training time, larger models achieve higher accuracy faster [3].

Datasets

We used the IMDb and Yelp Reviews datasets for our experiments. This was done to validate our results with multiple datasets and to uncover any potential obstacles our models would face when varying the size of the data they have been trained on.

The IMDb Movie Reviews dataset is a binary sentiment analysis dataset made up of 50,000 positive and negative reviews from the Internet Movie Database (IMDb). There are an equal amount of positive and negative reviews in the sample. The second dataset used was the original Yelp reviews dataset which contains reviews about businesses submitted by uses on Yelp's website. It is too large (11.37 GB) and not feasible to be trained on our local machines. Therefore, we sampled a subset of this dataset containing 150,000 rows. It is similar to the IMDB dataset in its format, with the "review" column containing the raw text reviews and the "label" column containing the sentiment labels. Both datasets contain 2 class labels signifying a positive or negative review.

Before evaluating or fitting a model to the data, it must be cleaned. To emphasize the attributes that we will want our models to pick up on, we will need to clean up the text data. All reviews were processed to discard all the **punctuations, URLs, HTML patterns and stopwords**.

To ensure consistency and fair comparison across the three approaches outlined in this report (i.e LSTM, CNN and Transformer) we processed both datasets once and used the same data in all our experiments. Both datasets have been split for appropriate validation and testing in the following format: **Train (70%); Test (15%); Validation (15%).**

Methodology

When conducting comparative analysis, we started with fitting a base vanilla model for each, optimising its hyper-parameters and then building on top of this base model to implement a novel approach for reducing training time. This was implemented for all three models on both datasets. We have also implemented two different models based on size for our transformers approach. We wanted to investigate the effect of transformers model parameters size on its training and performance.

LSTM

Baseline (LSTM-Base): For the scope of this project, we have used bi-directional LSTM model for the prediction of sentiment polarity of two distinct datasets of varying sizes (IMDB movie reviews, Yelp reviews). In contrast to the traditional LSTM units provided by tensorflow, we have used GRU (Gated Recurrent Units) for training our model, as they give superior results to LSTM while following the same fundamental architecture [6].

Since the tensors in the LSTM model work much faster with integers, we first convert all the string data to integer tokens using the native tensorflow tokenizer. These tokens can then be fed as inputs to our model. The total parameters trained on the base LSTM model used are 23,719,297.

LSTM with attention (LSTM-Att): With our base LSTM model trained and tested, our next aim was to look for novel ways to improve the training time of the LSTM model.
Although LSTM addresses the vanishing gradient descent problem in vanilla RNNs by adding a memory cell for context, it still lacks in performance when the sequence length is considerably large.

To mitigate this problem we implemented an LSTM model with an attention mechanism [5]. This architecture can be broken down in 4 main components-

1. **Input word embeddings:** This step forms the input to the LSTM model. It takes in the word tokens and uses an embedding layer for lookup of the word embeddings needed for each data point. These word embeddings are then passed through a convolution layer to extract only the important features from these words. Then passing this data through a maxpool layer, we get a dense vector representation for each word (see fig 1).

2. **LSTM (GRU) layer:** Since we are using a bi-directional model, this layer makes two passes on the data in each direction (see fig 2) -
    a. Forward pass - Takes in a sequence of word vectors and generates a new vector based on what it has seen so far in the forward direction (starting from the start word up until current word) this vector can be thought of as a summary of all the words it has seen.
    b. Backward pass - It does the same function as the forward layer in the opposite direction, from the end of the sentence to the current word.

3. **Attention layer:** Not all words contribute equally to the expression of the sentiment in a message. We use an attention mechanism to find the relative contribution (importance) of each word. The attention mechanism assigns a weight to each word annotation. We compute the fixed representation of the whole message as the weighted sum of all the word annotations.

4. **Output dense layers** - After we get the attention vectors from the previous layer, we then use a linear layer to map these probabilities using a relu function. Since we only predict two tags (“positive” and “negative”), the last dense layer has only one sigmoid output.

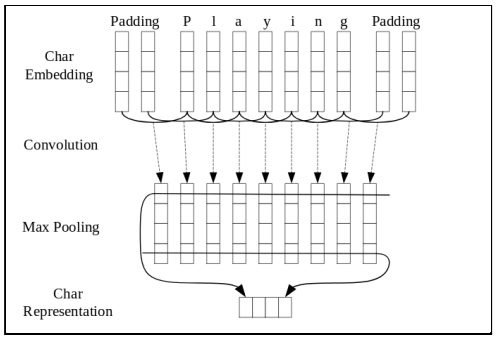


Fig. 1 - Input layer embedding

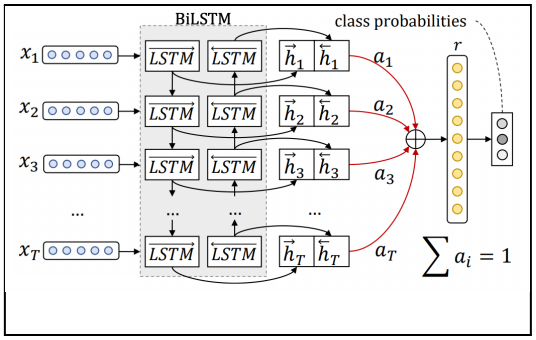


Fig. 2 - Bi-directional LSTM with attention

CNN

Baseline (CNN-Base): The base CNN architecture is a variation of the CNN model proposed in Collobert et al. (2011) [11] which gives state-of-the-art results for sentiment analysis.

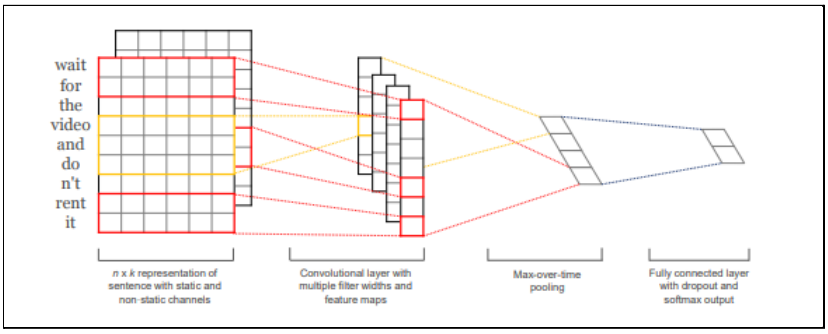


Fig. 3 - Architecture of CNN-Base

Character Level CNN (CNN-Char): The improvement CNN model follows the character level CNN architecture introduced in Zhang et al. Further explanation on the architecture can be found below:

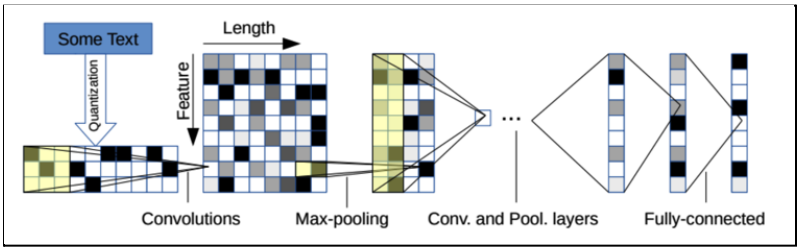


Fig. 4 - Architecture of CNN-Char

1. **Data quantization**: We quantize the input data from a series of texts into a series of characters (consists of alphabets, numbers and common punctuation). We have chosen 68 characters plus one character for unknown OOV characters as part of the vocabulary. After quantization, we make sure the length of all input sequences are equal by adding padding. We set the sequence length for each data point to be 1014. This number is decided based on the distribution of character length for all data points.

2. **Convolution**: Next we apply convolution operations to get the feature map. The aim of this operation is to extract high-level features from the input text. Its hyperparameters include the filter size (frames), kernel size and stride. In our model, we have kept stride to be consistently 1 throughout the model. The values of filter size (frame) and kernel size changes according to the layer and can be viewed in Table 1.

3. **Pooling**: Convolution layer introduces some spatial invariance, hence pooling is typically applied after convolution to fix this. In our model, we apply max pooling where each pooling operation selects the maximum value of the current view. The hyperparameter for pooling is typically the view/pool size which can be viewed in Table 1.

In total, 6 convolution layers are applied where three have max pooling layers.

4. **Fully Connected (FC):** At the end of CNNs, a FC layer is typically used to learn non-linear combinations of high-level features given by convolution and pooling operations. In our case, we have used 3 fully connected layers where the configurations of the layers can be viewed in Table 2. There's a dropout layer with a probability of 0.5 after the 7th and 8th layer..

Table 1 - Convolutional layers used in CNN-Char experiments

Layer	Frame	Kernel	Pool
1	256	7	3
2	256	7	3
3	256	3	N/A
4	256	3	N/A
5	256	3	N/A
6	256	3	3

Table 2 - Fully-connected layers used in our experiments

Layer	Output Units	Activation
7	1024	relu
8	1024	relu
9	2	sigmoid

Transformers

We have used two pre-trained models with different model size/number of parameters. We want to compare the performance and training time taken by the two different model sizes on the standard and novel approach and grab some insights. The models are as follows:

1. **DistilBERT**: DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters. (Number of parameters: 66,955,010)

2. **BERT**: It is a larger model. It's a bidirectional transformer pre-trained using a combination of masked language modeling objective and next sentence prediction on a large corpus comprising the Toronto Book Corpus and Wikipedia. (Number of parameters: 109,483,778)

Baseline(DistilBERT/BERT-FP): The inputs to all neural networks must be of the same shape and size. When we pre-process the texts and utilize them as inputs to our model, we realise that not all of the sentences are of the same length. We need the inputs to be the same size, which is where the padding comes in. We also provide the model with an “attention mask” for each input text, which identifies the [PAD] tokens and tells the transformer model to ignore them. Our baseline approach is ‘**Fixed Padding**’. In this approach, all the sequences are padded to a fixed length that is equal to the longest sequence present in the entire dataset.

Sentence	Token squence						Batch No.	Batch Length
Sequence 1	Movie	was	very	good	[PAD]	[PAD]	1	6
Sequence 2	It	was	a	very	good	movie		
Sequence 3	Great	movie	[PAD]	[PAD]	[PAD]	[PAD]		
Sequence 1	The	movie	was	horrible	[PAD]	[PAD]	2	6
Sequence 2	The	story	was	bad	[PAD]	[PAD]		
Sequence 3	Worst	movie	ever	[PAD]	[PAD]	[PAD]		
Sequence 1	It	was	decent	a	movie	[PAD]	3	6
Sequence 2	Good	movie	[PAD]	[PAD]	[PAD]	[PAD]		
Sequence 3	Bad	movie	[PAD]	[PAD]	[PAD]	[PAD]		

Fig. 5 - Fixed Padding

Smart Batching (DistilBERT/BERT-SB): We used two modifications in the data processing that considerably reduced transformer training time while having no detrimental impact on accuracy.

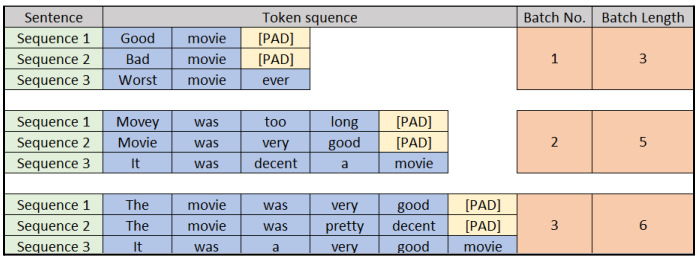
1. **Dynamic Padding**
   While the attention mask prevents the [PAD] tokens from influencing the transformer's interpretation of the text, the [PAD] tokens are fully included in all of the transformer's mathematical operations. This means they have an effect on our training and evaluation speed.

   Although all samples inside a batch must be the same length, transformers are not concerned about the batch length, and we can give it batches of text with varying maximum lengths. Instead of a fixed length value for the entire data, we limit the amount of new pad tokens to the length of the longest sequence of each mini batch. We call it "dynamic" padding since the quantity of tokens added varies across tiny batches.

Sentence	Token squence						Batch No.	Batch Length
Sequence 1	Movie	was	very	good	[PAD]	[PAD]	1	6
Sequence 2	It	was	a	very	good	movie		
Sequence 3	Great	movie	[PAD]	[PAD]	[PAD]	[PAD]		
Sequence 1	The	movie	was	horrible			2	4
Sequence 2	The	story	was	bad				
Sequence 3	Worst	movie	ever	[PAD]				
Sequence 1	It	was	decent	a	movie		3	5
Sequence 2	Good	movie	[PAD]	[PAD]	[PAD]			
Sequence 3	Bad	movie	[PAD]	[PAD]	[PAD]			

Fig. 6 - Dynamic Padding

2. **Uniform Length Batching**
   We push the logic even further by creating batches with sequences of comparable lengths, avoiding extreme scenarios where the majority of sequences in the mini batch are short and we have to add a lot of pad tokens to each of them because one sequence in the same mini batch is excessively long.



Sentence	Token squence						Batch No.	Batch Length
Sequence 1	Good	movie	[PAD]				1	3
Sequence 2	Bad	movie	[PAD]					
Sequence 3	Worst	movie	ever					
Sequence 1	Movey	was	too	long	[PAD]		2	5
Sequence 2	Movie	was	very	good	[PAD]			
Sequence 3	It	was	decent	a	movie			
Sequence 1	The	movie	was	very	good	[PAD]	3	6
Sequence 2	The	movie	was	pretty	decent	[PAD]		
Sequence 3	It	was	a	very	good	movie		

Fig. 7 - Uniform Length Batching

Experiments and results

Below we have summarised our experiment results for the comparative analysis for all 3 models and their improved versions for both datasets. The summary of our results focuses around the training time for each model w.r.t. the accuracy achieved and the total parameters trained on.

For fair comparison between model architectures, the following parameters have been kept constant throughout with some exceptions to minimize the variables affecting training time:

1. Batch size = 16
2. Word vector sequence length = 400
3. Maximum number of words in vocabulary = 30522
4. Embedding Dimensions = 768
5. Same train-test-validation data

Limitations:

1. CNN-Char model doesn't converge with batch size 16 for Yelp dataset, regardless of hyperparameter tuning, hence we used batch size 128 for that specific experiment.
2. Character level embedding affects the maximum number of words in vocabulary and word sequence length, hence the above parameter constants do not apply to CNN-Char.
3. For the transformer models, the Yelp dataset was reduced to 1/3rd due to GPU resource limitation. The Colab connection was getting timed out due to the restriction of the free-tier account and the long training time. Thus, we will assume the training time for the full data to be three times the current training time.

Table 3 - Model Performance (IMDB Dataset)

Model	Test Accuracy (%)	Val Accuracy (%)	Training time until convergence	Time reduction (%)	Parameters
LSTM-Base	90.34 %	90.12 %	45 min 24 sec	23.78 %	23.7 M
LSTM-Att	88.98 %	89.06 %	34 min 36 sec		23.7 M
CNN-Base	88.61 %	88.65 %	4 min 50s	66.2 %	23.7 M
CNN-Char	50 %	50 %	1 min 38 s		11.3 M
DistilBERT-FP	91.2 %	91.0 %	41 min 36 sec	61.97 %	66.9 M
DistilBERT-SB	91.1 %	90.9 %	15 min 49 sec		66.9 M
BERT-FP	91.8 %	91.7 %	1 hr 17 min 17 sec	59.75 %	109.4 M
BERT-SB	91.8 %	91.4 %	31 min 06 sec		109.4 M

Table 4 - Model Performance (Yelp Dataset)

Model	Test Accuracy (%)	Val Accuracy (%)	Training time until convergence	Time reduction (%)	Parameters
LSTM-Base	86.05 %	87.39 %	2 hr 1 min 39 sec	7.4 %	23.7 M
LSTM-Att	87.21 %	87.72 %	1 hr 54 min		23.6 M
CNN-Base	92.24 %	91.90 %	17 min 1 sec	31.5 %	23.7 M
CNN-Char	90.55 %	90.31 %	11 min 39 sec		11.3 M
DistilBERT-FP	93.1 %	93.4 %	1 hr 32 min 24 sec	77.06 %	66.9 M
DistilBERT-SB	92.9 %	93.2 %	27 min 9 sec		66.9 M
BERT-FP	93.3%	93.8%	3 hr 37 min 42 sec	74.64%	109.4 M
BERT-SB	93.4%	93.7%	1 hr 16 sec		109.4 M

Discussion

In terms of model size and training time, the accuracy of the three models varies across datasets. From the results table 3, we can see that the base LSTM and CNN models have a comparable number of parameters (23.7M) with similar accuracy. Their training times however differ by about 40 mins which shows that CNN outperforms RNN. It is noteworthy that DistilBERT-SB was able to achieve a higher accuracy than both LSTM and CNN even though it was trained on 1/3rd the dataset. This is possible because transformers process sentences as a whole with multi-head attention and positional embedding.

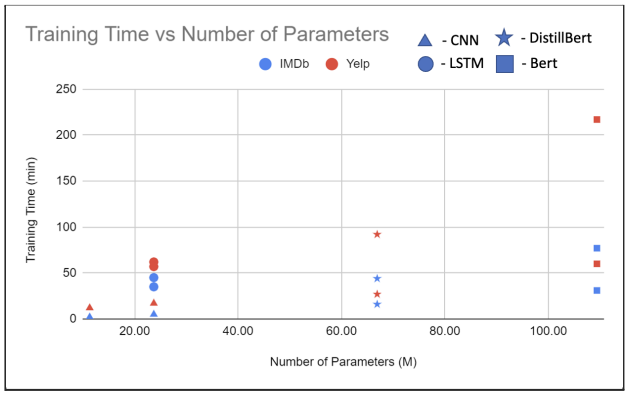


Fig. 8 - Training time vs parameter size

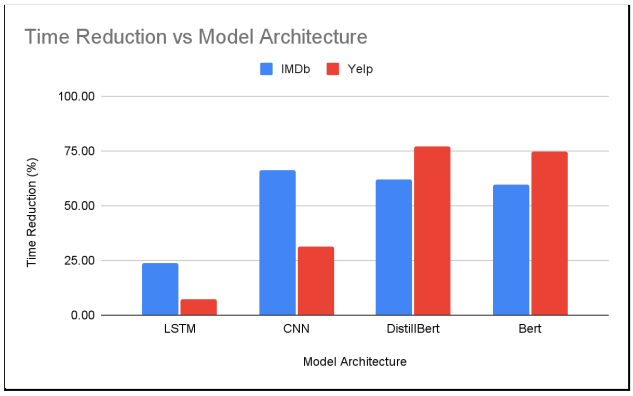


Fig. 9 - Train time reduction for each model

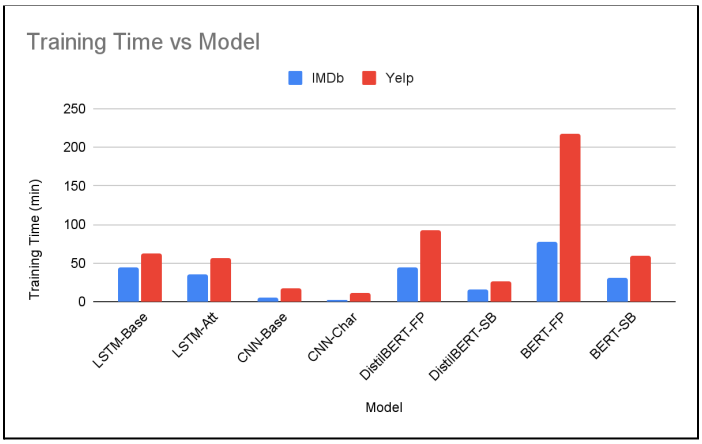


Fig. 10 - Training time per model for both datasets

From the above tables, we observe the effects of datasize on the different models. The Yelp dataset is three times bigger than the IMDB dataset. We can see in the case of LSTM models that the training time increased by a smaller percentage (30%-60%) compared to the CNN models with the increase in training time around (252%-613%). We also observe that the time reduction percentage for the standard and novel approach reduces drastically with the increase in the data size.

Final insights from each model

LSTM: For the IMDB dataset, the LSTM model with attention mechanism was able to improve the training time by 23.7% as compared to the vanilla LSTM. It used attention vectors between LSTM layers for better context retention which helped the model converge faster without much tradeoff in accuracy. The time reduction for Yelp dataset is 7.4%. This can be used to conclude that LSTM with attention vectors performs similar to vanilla LSTM when the dataset size is high.

CNN: Experiments with both the datasets show that character level CNN converges much faster than baseline CNN. Char level CNN converges 66.2% and 31.5% faster for IMDb and Yelp reviews dataset as compared to baseline CNN. The main reason behind this is that the word embedding dimensions get much smaller leading to a smaller model size, hence faster training. Word embedding gets smaller since there are only 70 characters in vocabulary.

The accuracy for character level CNN in the case of IMDb is 50% which is equal to the probability of randomly choosing a category in case of binary classification. This means that the dataset is too small to accurately predict with such raw form of embeddings because word-level embeddings works perfectly fine and the model works well in case of large datasets (Yelp reviews).

Transformers:

1. As the number of tokens given to the model plays an important role in the training time, we can see that the novel approach has a lower training time while having similar or higher accuracy. The 'Smart Batching' approach was successful to reduce the number of tokens by 50% or more.

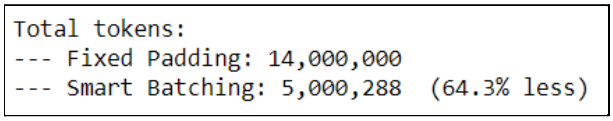


Fig. 11 - Comparison of number of tokens created by the two approaches for IMDB data

2. We can see that if we compare the performance of DistilBERT and BERT on the same data. We can see that even though DistilBERT is a smaller model, in our use case, the model was successful in providing similar accuracy with a lesser training time compared to the larger BERT model. The DistilBERT model was successful to reduce the training time by roughly 20% everytime when compared with the larger BERT model.

3. We can also see that if we compare the performance of the novel and standard approach on the transformer models, we get a drastic time reduction of more than 60% on both the datasets. Through this observation, we realise the importance of data handling in terms of batching, tokens and padding.

Conclusion

This study describes a series of experiments on deep learning networks in sentiment analysis. From the above discussion, we can conclude that we have successfully identified approaches in LSTM, CNN and Transformers to reduce training time on multiple-sized datasets. We have also seen how model architecture and size affects model performance. Future works in this area can include testing our approaches on multiple languages, varied pre-trained models, including latest state-of-the-art architectures and other sentiment analysis tasks.

Citations

[1] Yin, Wenpeng & Kann, Katharina & Yu, Mo & Schütze, Hinrich. (2017). Comparative Study of CNN and RNN for Natural Language Processing.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. (2017). Attention Is All You Need.

[3] Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, Joseph E. Gonzalez. (2020). Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers.

[4] Sepp Hochreiter, Jürgen Schmidhuber. (1997). Long Short-Term Memory Neural Computation.

[5] Wang, Yequan & Huang, Minlie & Zhu, Xiaoyan & Zhao, Li. (2016). Attention-based LSTM for Aspect-level Sentiment Classification. 606-615. 10.18653/v1/D16-1058.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.

[8] Nhan Cach Dang, María N. Moreno-García, Fernando De la Prieta. Sentiment Analysis Based on Deep Learning: A Comparative Study

[9] Devika M D, Sunitha C, Amal Ganesha. Sentiment Analysis:A Comparative Study On Different Approaches

[10] X. Zhang, J. Zhao, and Y. Lecun, “Character-level convolutional networks for text classification,” in Advances in Neural Information Processing Systems, 2015

[11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa. Natural Language Processing (Almost) from Scratch in Journal of Machine Learning Research 12, 2011.