DEEP LEARNING FOR NATURAL LANGUAGE PROCESSING (ELEC0141) 24 REPORT

SN: 20121776

ABSTRACT

In the field of natural language processing, text classification is one of the fundamental tasks explored by many researchers. With the goal of testing and comparing different models, this project developed two Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) layers, as well as transfer learned a Distilled Bidirectional Encoder Representations from Transformers (DistilBERT) model. They were trained on the Yahoo! Answers topic classification dataset for a 10-class classification task to obtain numerical and graphical data for evaluation.[1]

Index Terms— RNN, LSTM, GRU, BERT, DistilBERT

1. INTRODUCTION

Ever since the release of ChatGPT in November 2022, artificial intelligence, or more specifically, natural language processing (NLP) has captured global attention and become the mainstream in research. Common applications of this state-of-the-art technology include machine translation, question answering, text classification and more. To learn more about the advanced NLP methodologies, this project developed multiple models for a 10-class text classification task, with the aim of providing detailed explanations and performance comparisons.

The text data in this project was leveraged from the Yahoo! Answers topic classification dataset (Yahoo Dataset) [1]. It contains 10 unique question topics, with each having 140k training / 6k testing samples. Due to the limited computational resources, the original testing set was discarded, and 10k data points were sampled from the 1.4M training samples.

To provide clean and meaningful data to the models, samples were preprocessed to remove stop words, special characters, punctuation and etc. Sentences with abnormal lengths were also filtered to maintain consistency. At the end, a total of 46k samples were fed into the models with an 8:1:1 split for training, validation and testing.

This project comprises 3 models separated into two tasks. Task A consists of a Recurrent Neural Network (RNN) that uses either Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers, while task B features a Distilled Bidirectional Encoder Representations from Transformers (DistilBERT) model. Although task A's models were designed from scratch with PyTorch, the transformer for task B utilised transfer learning from the HuggingFace library as training a large language model is beyond the scope of this project.

This report explains the methodologies of all aforementioned models, as well as provides training, evaluation and testing details through numerical and graphical results.

[1]The code for this project can be accessed from the GitHub repository: DLNLP_assignment_24_SN20121776

2. LITERATURE SURVEY

A wide range of model architectures exists in the field of NLP, with deep learning approaches such as neural networks and transformers being the preferred option for text classification tasks. For instance, [2] carried out an in-depth comparison between convolution neural networks (CNN), deep belief networks (DBN) and recurrent neural network (RNN) across 7 unique classification datasets. Similarly, [3] compared the performances of a transformer (BERT) model against several classical machine learning algorithms, including logistic regression and support vector machine.

As RNNs and transformers have demonstrated outstanding results in literature, further research was conducted to develop deeper understanding of their theories as well as their implementation processes.

2.1. RNN

RNNs are characterised by a series of hidden memory cells that are recurrently updated. Past inputs are remembered and taken into consideration when determining the current outputs for sequential text classification tasks [4]. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are the two most common variants of RNN, with GRU being the less computationally intensive alternative [5].

[6] trained multiple LSTM and GRU models for a series of tasks including sentiment classification, relation classification, textual entailment and more. Since both variants have demonstrated competent performances, this project aims to implement both architectures to determine the best option for the Yahoo Dataset.

2.2. BERT and DistilBERT

BERT is a pre-trained language model developed in 2018 by [7] that uses the attention mechanism introduced by the renowned *Attention is All You Need* article [8]. The novelty of BERT comes with its capability of processing word tokens bi-directionally for better understanding of sentence contexts and semantics. However, being trained with 340 million parameters, transfer learning BERT model is computationally intensive [7]. A more lightweight alternative - DistilBERT was therefore proposed by Hugging Face developers to substantially speed up the model by 60% while reducing the model size by 40% [9].

Fine-tuning is crucial for transfer learning pre-trained transformers. [10] provided detailed strategies for adjusting the models for specific tasks, including optimal hyperparameter range and etc. This project will therefore employ a similar approach to fine-tune the DistilBERT model for optimised performances.

3. DESCRIPTION OF MODELS

3.1. Part A: RNN

Given that RNN is one of the most popular options for NLP text classification tasks, Part A of this project consists of a RNN model that uses either LSTM or GRU layers. While both variants are expected to perform similarly, their differences in architectures and computational efficiencies are worth exploring.

Note that diverse designs exist for RNNs, hence the following descriptions are based on the models used in this project.

Word ID	Embed 1	Embed 2	...	Embed n
0	x_{01}	x_{02}	...	x_{0n}
1	x_{11}	x_{12}	...	x_{1n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
m	x_{m1}	x_{m2}	...	x_{mn}

Table 1: An example word embedding look-up table.

The RNN starts with an embedding layer that converts each word in a tokenised sentence into word embedding vector to represent the contexts and semantics of the inputs texts. To maintain consistent input lengths, sentences are padded per batch, and a length vector is provided to indicate their length. This layer can be considered as a look-up table as shown in Table 1. Each indexed word (m) refers to n embedded values (x_{mn}), where m is defined by the total number of unique vocabularies in the training dataset and n is a tuneable hyperparameter. The output of this layer is a 2-dimensional dense vector that carries the embeddings of each word in the input sentence.

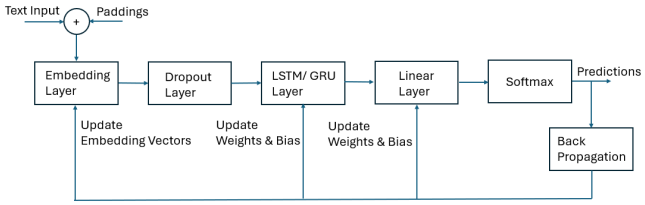


Fig. 1: The flow chart of the RNN model.

Following the embedding layer, a dropout layer is incorporated as shown in Figure 1. It randomly resets the embedding values x_{mn} back to 0 based on a tuneable probability parameter. Learned word patterns can therefore be removed arbitrarily to prevent the model from overfitting.

The new word embeddings then proceed to either the LSTM or GRU layers, which are explained individually in the following subsections for better clarity. Regardless of the selected variant, its outputs are further processed by a linear layer to match the dimensionality of the dense embedding vector to the total number of classes with the following equation.

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W}^T + \boldsymbol{b}$$

Where $\boldsymbol{y}$, $\boldsymbol{x}$, $\boldsymbol{W}$ and $\boldsymbol{b}$ are the output, input, weight and bias vectors respectively. These vectors, together with the embedded values x_{mn}, are randomly initiated at the beginning of training process and updated by backpropagation.

A softmax function is then used to convert the scores from the linear layer to a probability distribution over the available classes. The most probable class is considered as the model output for calculating the cross-entropy loss. Finally, the Adaptive Moment Estimation (Adam) optimization algorithm introduced by [11] is deployed to iteratively update the randomly initiated vectors during gradient descent and lead the model to convergence.

3.1.1. LSTM

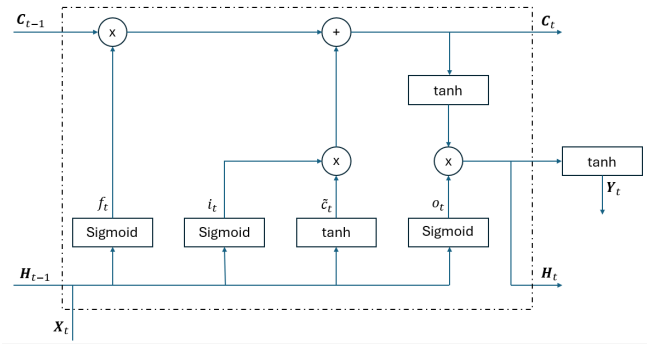


Fig. 2: Internal structure of the LSTM cell.

The LSTM layer consists of multiple memory cells interconnected in series as shown in Figure 2. $\boldsymbol{C}_t$ is the cell state

that memorises the information of the entire layer, $\boldsymbol{H}_t$ is the hidden state that captures word relationships and $\boldsymbol{X}_t$ is the input at timestep t [12].

The outputs of the LSTM layer can be derived either by the average of hidden states $\boldsymbol{Y}_t$ across all timesteps, or by extracting the hidden state from the final timestep. The former captures more contextual information, while the latter is more computational efficient.

Three gates are employed to regulate the flow of data to the next LSTM cell. The forget gate determines the information retained from the previous cell state, while the input and output gates update the cell and hidden states respectively.

Function	Equation
Forget Gate	$f_t = \sigma(\boldsymbol{W}_f[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_f)$
Input Gate	$i_t = \sigma(\boldsymbol{W}_i[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_i)$
Output Gate	$o_t = \sigma(\boldsymbol{W}_o[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_o)$
Potential Cell State	$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_c)$
Cell State	$\boldsymbol{C}_t = (i_t \times \tilde{\boldsymbol{c}}_t) + (f_t \times \tilde{\boldsymbol{c}}_{t-1})$
Hidden State	$\boldsymbol{H}_t = o_t \times \tanh(\tilde{\boldsymbol{c}}_t)$
Output	$\boldsymbol{Y}_t = \tanh(\boldsymbol{W}_y \boldsymbol{H}_t + \boldsymbol{b}_y)$

Table 2: Equations of the LSTM cell [12].

Table 2 lists the equations associated to the LSTM cell. $\boldsymbol{W}$ and $\boldsymbol{b}$ represent the weights and bias trained through back-propagation, and $[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}]$ is the concatenation of the input vector and hidden state. The symbol σ denotes the sigmoid function for introducing non-linearity, which can be expressed as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

3.1.2. GRU

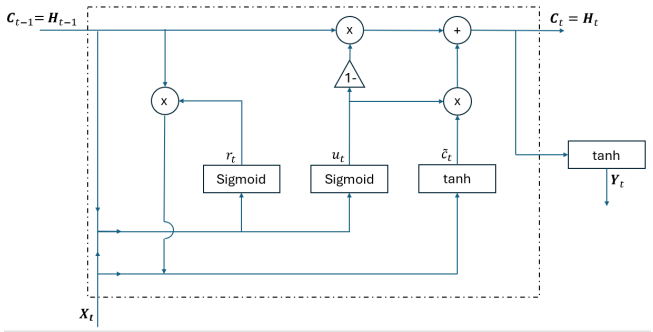


Fig. 3: Internal structure of the GRU cell.

As displayed in Figure 3, the GRU variant has simpler cell structure. The cell and hidden states from LSTM are combined to form a new hidden state ($\boldsymbol{H}_t$) to contain the layer memories and word relationships at once. The regulation gates are also simplified by merging the input and forget gates into a reset gate for the hidden state. Furthermore, the output gate is replaced by an update gate to revise the hidden state based on the current input sequence. As GRU shares the same equation formats as LSTM in Table 3, explanations are omitted. It is also worth noticing that much like LSTM, GRU supports output extraction from either the averaged or the last timestep's hidden state.

Function	Equation
Reset Gate	$r_t = \sigma(\boldsymbol{W}_r[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_r)$
Update Gate	$u_t = \sigma(\boldsymbol{W}_u[\boldsymbol{X}_t; \boldsymbol{H}_{t-1}] + \boldsymbol{b}_u)$
Potential State	$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_c[\boldsymbol{X}_t; r_t \boldsymbol{H}_{t-1}] + \boldsymbol{b}_c)$
Hidden State	$\boldsymbol{H}_t = (u_t \times \tilde{\boldsymbol{h}}_t) + (1 - f_t) \times \boldsymbol{H}_{t-1}$
Output	$\boldsymbol{Y}_t = \tanh(\boldsymbol{W}_y \boldsymbol{H}_t + \boldsymbol{b}_y)$

Table 3: Equations of the GRU cell [12].

3.2. Task B: BERT

Although RNN is one of the most prominent architectures for text classification, the fact that training is conducted on developer-provided datasets significantly limits its performance. Transformer models, however, have demonstrated exceptional classification results as large language models were pre-trained on billions of parameters. To explore the prowess of transformers while maintaining reasonable processing time, the light-weight variant of BERT - DistilBERT was chosen for this project.

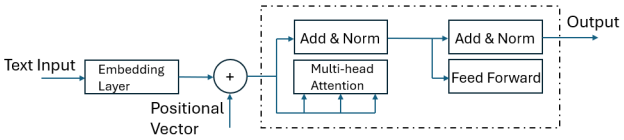


Fig. 4: Internal structure of a transformer encoder.

Given that text classification tasks do not involve generative outputs, an encoder-only model is used in this project and its flow chart is illustrated in Figure 4 [13]. Tokenised texts are first converted into word embeddings with the same concept described in Section 3.1. To preserve sentence sequence information, positional encodings are generated using the following equations and summed with the embedding vectors.

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{n}}}\right)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{n}}}\right)$$

Where pos is the position of the word in the sentence, i and n are the index and the dimensions of the embedding vector respectively.

The resultant vectors are then split into k heads, each containing a subset with n/k dimensions. This allows the paral-

lel computation of multi-head attention values to speed up the process [8]. The equation of the mechanism is defined as:

$$Attn(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{(d_k)}}V\right)$$

The query (Q), key (K) and value (V) vectors are calculated by multiplying the word embeddings of each head with randomly initiated weight matrices W_q, W_k and W_v respectively. By training them with backpropagation, the attention values can be iteratively updated.

The next step involves summing and normalising the attentions values from each head to stabilise the model. They are concatenated and added to the embedding vector for a new vector v, which is then used for computing the mean μ and standard deviation σ. The normalisation operation is expressed as:

$$Norm(v) = \gamma\frac{v - \mu}{\sigma} + \beta$$

Where γ and β are the scaling parameter and bias vector defined by the developer.

In the final stage of the encoder, the normalised values are directed to a feedforward neural network. This layer first carries out linear transformation with trained weights and bias, then introduces non-linearity using the Rectified Linear Unit (ReLU) function. Another normalisation step is included to further enhance model stability, and the final outputs can be passed through linear layers and softmax functions for class predictions.

4. IMPLEMENTATION

4.1. Preprocessing

The preprocessing step commenced with analysing the Yahoo Dataset. It contains a total of 1.4M training/ 60k testing data points, each with entries of class label, question title, question content and answer. Given the substantial size of the dataset, using all available data was deemed impractical for the scope of the project. The original dataset was therefore down-sampled to 10k data points, with the question content and answer columns removed to enhance efficiency.

Further inspection of sampled dataset revealed plenty of impure texts. For instance, some question titles were comprised of XML/ HTML tags, hyperlinks, punctuation, numbers and etc. To avoid model misinterpretation, a *preprocessor.py* module was developed to replace undesirable texts with empty spaces.

For further cleaning, the texts were converted into ASCII expressions and lowercased. Stop word removal and lemmatisation were then performed to retrieve meaningful data and reduce the words to their canonical forms. The Python libraries used for preprocessing are listed in Table 4.

Library	Function
bs4	Filter HTML/ XML tags.
re	Filter hyperlinks, punctuations, special characters, leading/ trailing/ repeated spaces.
unicodedata	ASCII conversion.
spaCy	Stop word removal, lemmatisation.

Table 4: Python libraries used for preprocessing.

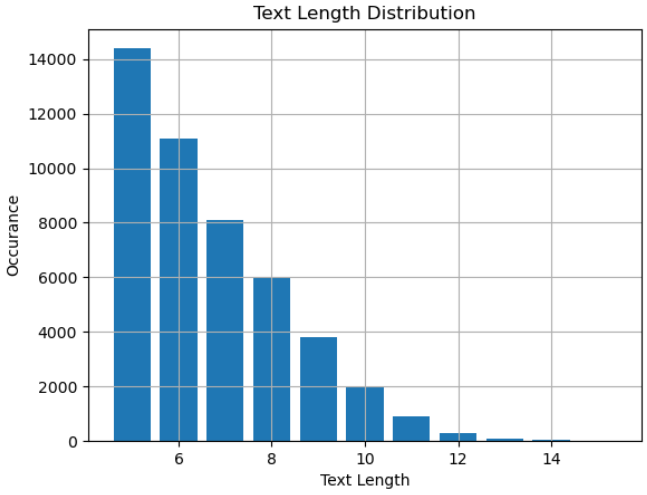


Fig. 5: Text length distribution of the sampled dataset.

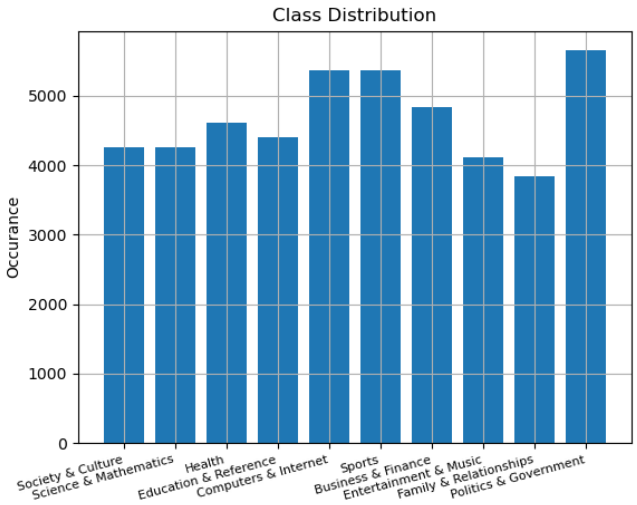


Fig. 6: Class distribution of the sampled dataset.

After filtering the unnecessary texts, it was noticed that some data had abnormal text lengths. The short texts (<5 words) carried no meaning, while the long texts (>15 words) hindered the subsequent padding process. Thus, the outliers were removed and a total of 46k data points remained. The distribution of text lengths can be viewed in Figure 5.

As Figure 6 showed that the randomly sampled dataset was balanced with around 4 to 5k entries per class, data augmentation was unnecessary. The preprocessed dataset was then exported as a CSV file for further usage, and divided into training/ validation/ testing splits with the ratio of 8:1:1 using the *custom_dataloader.py* module.

As a result, a total of 29634 unique vocabularies were retrieved from the training split and the sentences were tokenised using PyTorch's *get_tokenizer* function.

4.2. Task A: RNN

4.2.1. Hyperparameters tuning

Parameter	Symbol	Range
Model selection	*model*	LSTM or GRU
No. LSTM/ GRU Layers	*num_layers*	1 - 3
Embedding Size	*embed_dim*	16 - 256
Hidden Layer Size	*hidden_size*	16 - 64
Dropout Rate	*dropout*	0.1 - 0.5
Use Last Output	*use_last*	True or False

Table 5: Hyperparameters of the RNN models.

The RNN consists of 6 tuneable hyperparameters as shown in Table 5. The *model* parameter defines the inner architecture (LSTM/ GRU), while the *num_layers* value controls model complexity by adjusting the number of recurrent layers.

embed_dim specifies the length of embedding vector for each word, which governs the abundance of patterns capturable from the input sentences. Similarly, *hidden_size* determines the length of the hidden state vectors $\boldsymbol{H}_t$ within the recurrent layers. Increasing the length of embeddings or hidden state vectors enhances the availability of contextual information at the risk of overfitting.

The dropout rate mentioned in Section 3.1 can be controlled by the *dropout* parameter. Note that if multiple recurrent layers exist, additional dropouts are applied in between them due to the nature of stacked recurrent layers in PyTorch.

The output of the model can then be selected from either the averaged or the last timestep's hidden state using the *use_last* parameter.

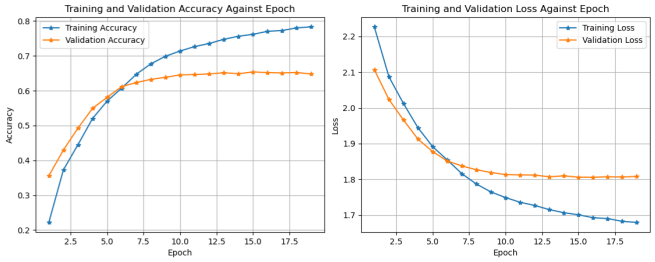
After defining the ranges of hyperparameters, several models were trained with different combinations for trial and error. Meanwhile, the learning rate and weight decay of the Adam optimisation algorithm were also tested to optimise model convergence. The cross entropy losses of the models were then compared to find the best selections as shown in Table 6.

Parameter	Symbol	Optimal
RNN		
Embedding Size	*embed_dim*	128
Hidden Layer Size	*hidden_size*	32
No. LSTM/ GRU Layers	*num_layers*	2
Dropout Rate	*dropout*	0.3
Use Last Output	*use_last*	False
Optimiser		
Learning Rate	*lr*	0.001
Weight Decay	*weight_decay*	1e-5

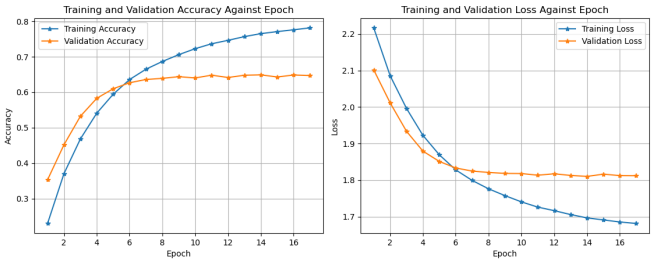
Table 6: Optimal hyperparameters of the RNN models.

4.2.2. Training and Validation

The LSTM and GRU models were trained on the optimal parameters using the *processor.py* module. Their plots of accuracy/ loss against epoch are displayed in Figure 7. As overfitting tends to occur after certain epochs, an early stopping function was implemented with the *early_stopper.py* module. The training loop is exited if no improvement in validation loss is observed over three consecutive epochs. The current best model is then used for evaluation.



(a) LSTM Accuracy Against Epoch (b) LSTM Loss Against Epoch



(c) GRU Accuracy Against Epoch (d) GRU Loss Against Epoch

Fig. 7: Traing and validation plots of the RNN models.

During training, the elapsed time, training and validation accuracies/ losses were logged at the end of each epoch. The evaluation results of the finalised models are summarised in Table 7.

Model	Epoch	T. Accu/ Loss	V. Accu/ Loss	Time
LSTM	19	0.783/ 1.679	0.648/ 1.808	190
GRU	17	0.782/ 1.681	0.647/ 1.812	175

Table 7: Training and validation results of the RNN models.

4.2.3. Prediction

Using the fine-tuned models, the testing set was processed to provide several performance metrics, including accuracy, precision, recall and f1score. In addition, confusion matrices and metrics comparison plots were also generated for visualisation. More details on the test performance will be presented in Section 5.

4.3. Task B: DistilBERT

4.3.1. Data preparation

To implement the DistilBERT model, the dataset was first reformatted. This is because the HuggingFace library only accepts *Dataset* objects, while the Pytorch RNN models in Section 4.2 requires input in *DataLoader* format. A new tokeniser was also defined using HuggingFace's built-in *AutoTokenizer* class to utilize the pretrained corpus.

4.3.2. Hyperparameters tuning

Parameter	Symbol	Range
No. Epoch	*num_train_epochs*	1 - 5
Learning Rate	*lr*	1e-5 - 1e-3
Weight Decay	*weight_decay*	1e-3 - 1e-2
Batch Size	*train_batch_size*	16 - 128
	eval_batch_size	

Table 8: Hyperparameters of the DistilBERT model.

As presented in Table 8, several hyperparameters were available for tuning when transfer learning the DistilBERT model. The *num_train_epochs* parameter controls the total number of training epochs. Since DistilBERT is a large language model, far fewer epochs are needed to obtain satisfactory results compared to the custom-trained RNN model. Similar to the RNN model, the learning rate and weight decay of the transformer's optimiser can be adjusted using the *lr* and *weight_decay* parameter.

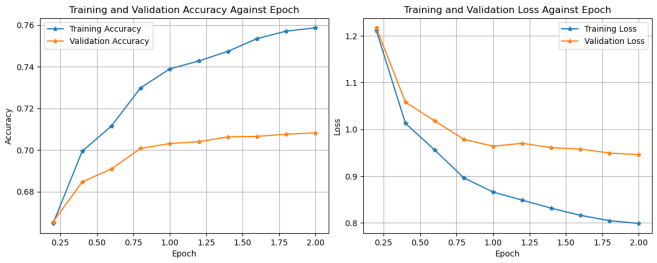
To facilitate parallel processing, the batch sizes for training and evaluation are set by *train_batch_size* and *eval_batch_size* respectively. Although larger batches are beneficial to the processing time, this project opted for smaller batch sizes due to the lack of computational resources. The optimised hyperparameters were obtained through a series of experiments, and are listed in Table 9.

Parameter	Symbol	Optimal
No. Epoch	*num_train_epochs*	2
Learning Rate	*lr*	2e-5
Weight Decay	*weight_decay*	0.01
Batch Size	*train_batch_size*	64
	eval_batch_size	

Table 9: Optimal Hyperparameters of the DistilBERT model.

4.3.3. Transfer learning and evaluation

The *Trainer* and *TrainingArguments* classes from HuggingFace were defined to initiate the model. In order to provide detailed training and validation records, logging was done at the end of each step (1/10 of a epoch). A custom callback was also included as HuggingFace does not generate validation results by default.



(a) Accuracy against Epoch (b) Loss against Epoch

Fig. 8: Training and validation plots of the DistilBERT model.

Epoch	T. Accu/ Loss	V. Accu/ Loss	Time
2	0.759/ 0.799	0.708/ 0.946	21:12

Table 10: Training and validation results of the DistilBERT model.

The accuracy/ loss against epoch plots and numerical results are provided in Figure 8 and Table 10 respectively. As the model does not see any significant improvement beyond 2 epochs, the training loop was manually capped for stopping.

4.3.4. Prediction

Predictions were then made on the fine-tuned model with the testing set. Much like the RNNs, the performance metrics and plots were generated and presented in Section 5 for further analysis.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1. Evaluation metrics

To better describe the model performances, the metrics in Table 11 were used and their equations are listed below. Note that TP, TN, FP, FN and n correspond to true positive, true negative, false positive, false negative and total number of samples respectively. These measurements are vital for analysing the model, which can be further visualised with the help of confusion matrices and metrics plots.

$$Accuracy = \frac{TP + TN}{n}$$
$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$
$$F1Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Metric	Purpose
Accuracy	Measurement of percentage of correct predictions over total samples.
Precision	Measurement of percentage of true positives over the sum of predicted positives.
Recall	Measurement of percentage of true positives over the sum of actual positives.
F1 Score	Harmonic mean of precision and recall.

Table 11: Purposes of the evaluation metrics.

On top of the numerical metrics, confusion matrices were also used to provide class-based performance analysis. To understand the layout of a multi-class confusion matrix, Figure 9 from [14] was used. The diagonal cells from top left to bottom right indicate the number of true positives of each class. The sum of values horizontal or vertical to the true positive cells are false negatives or false positives respectively. The rest of the cells represent the true negatives.

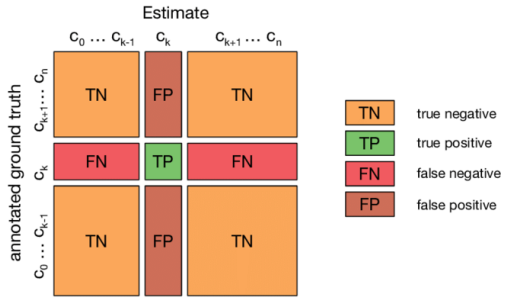


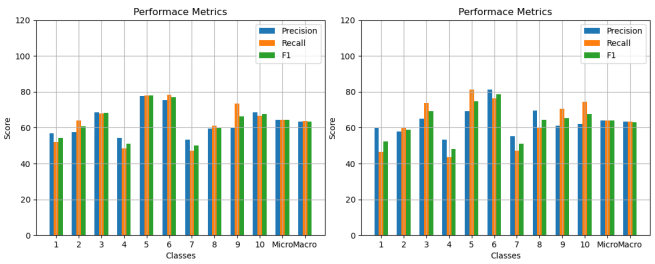
Fig. 9: Layout of a multi-class confusion matrix [14].

5.2. Model results

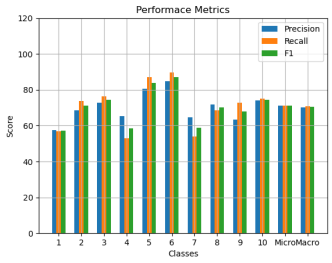
Model	Test Accuracy	Test Loss
LSTM	0.643	1.817
GRU	0.641	1.817
DistilBERT	0.713	0.941

Table 12: Test results of the models.

The prediction on the testing set yielded the results in Table 12. While the performances of the RNNs were almost identical with merely 0.2% difference in accuracy, the DistilBERT model outperforms both of them by approximately 7%. Comparing their performance metrics in Figure 10, it is notable that the DistilBERT model has better overall scores than the RNN models. This indicates that the transformer from transfer learning has superior performance over the self-trained RNN models in text classification tasks. However, significant discrepancies in class-metrics were still observed even with the more powerful DistilBERT model.



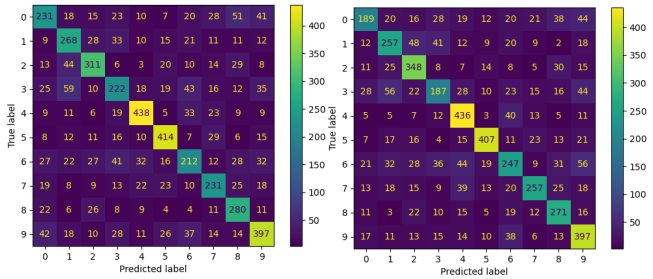
(a) LSTM (b) GRU



(c) DistilBERT

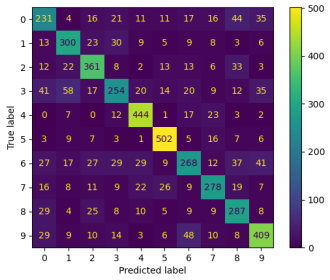
Fig. 10: Performance metrics of the models.

According to Figure 10 and 11, class 0, 3 and 6 (Society & Culture, Education & Reference and Business & Finance) have evidently worse results across all 3 models compared to the rest. While the scores of other classes managed to achieve above 60%, or even peak at 80%, the mentioned classes struggled to reach 60% even with the best performing transfromer model. This is likely because of the correlation of word patterns between different classes. Take the DistilBERT model as an example, the common false negative misclassifications are listed in Table 13.



(a) LSTM

(b) GRU



(c) DistilBERT

Fig. 11: Confusion matrices of the models.

True	Predicted
Society & Culture	Family & Relationship
Education & Reference	Science & Mathematics
Business & Finance	Politics & Governement

Table 13: Most common misclassifications of the DistilBERT model.

It is not suprising that the model struggles to distinguish between the above classes as their context are closely related. For example, Society and culture often involve family issues, while education and science topics overlap in mathematics. Furthermore, business often involves politics and is strongly tied to governments' policies. During model training, it is very likely that certain ambiguous sentence patterns were captured to represent classes with overlapped meanings, thus aggravated the classification accurcies.

Another important factor for model evaluation is their training time. While the DistilBERT model demonstrated superior performance compared to LSTM or GRU-based RNN, it also required the longest training time of 21 minutes over 2 epochs. In contrast, the LSTM model completed 19 epochs in 190 seconds, and the GRU model consumed even less time of 175 seconds for 17 epochs.

This is mainly because the GRU model has the simplest architecture and fastest convergenece. Although the DistilBERT is a lightweight variant of BERT, it is still fundamentally a large language model that requires decent computational power for transfer learning. Therefore, depending on the available time and computational resources, the GRU-based RNN might be the more practical option for text classification tasks dispite being less accurate.

6. CONCLUSION

In conclusion, all models in this project successfully classified input questions titles from the Yahoo Dataset into 10 unique classes. While the two variants of RNN (LSTM and GRU) achieved almost identical test accuracies of 64.3% and 64.1% respectively, the simpler network structure of GRU allowed for faster convergence and only required 175 seconds of training time. In contrast, the DistilBERT model demonstrated superior test accuracy of 71.3%, but the training time prolonged to over 21 minutes due to the size of the pre-trained model.

Overall, the model performances were satisfactory, but improvements could be made over several aspects in the future. First, to address the misclassification problem mentioned in Section 5, the data size can be increased to provide more class-unique word patterns. Longer embeddings can also be used to learn more complex context to avoid representing multiple classes with overlapped patterns.

Furthermore, better utilisation of the Yahoo Dataset can be achieved by including the question contents and answers as only question titles were used in this project due to computational limitations. Even though the questions' classes can usually be identified by only analysing the titles, some entries are rather lackluster and do not carry much meaning. Concatenating the titles with contents or even the answers can substantially enhance the amount of capturable patterns, thus improve model accuracy.

As for preprocessing, more thorough data cleaning can be conducted to ensure that the dataset only contains English. Although the Yahoo Dataset claimed to have plain English text only, some sampled data turned out to be non-english that could have negatively impacted the model performances. Several attempts were made to filter the non-english text but to no avail due to libraries like SpaCy requiring exceedingly long time to process the dataset word by word.

In closing, accuracy should not be the only factor when selecting the appropriate architecture for text classification tasks. Although the RNN models had inferior prediction accuracies than the DistilBERT model, they were remarkably less computational intensive, and required much shorter training time. Additionally, it was much simpler and efficient to fine-tune the RNN models in PyTorch as each layer was defined individually with adjustable parameters.

In scenarios where time and computational costs are not confined, transfer learning a complex transformer model can undeniably provide the best classification results. However, if project schedule and resources are constrained, training RNNs, especially GRU-based models could be the ideal alternative.

7. REFERENCES

[1] Xiang Zhang and Acharki Yassir, "Yahoo! answers topic classification dataset," Kaggle, 2007.

[2] Zulqarnain, Rozaida Ghazali, Yana Mazwin, and Muhammad Rehan, "A comparative review on deep learning models for text classification," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, 2020.

[3] Santiago González-Carvajal and Eduardo C. Garrido-Merchán, "Comparing BERT against traditional machine learning text classification," *CoRR*, vol. abs/2005.13012, 2020.

[4] Alex Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, 2020.

[5] Ezat Ahmadzadeh, Hyunil Kim, Ongee Jeong, Namki Kim, and Inkyu Moon, "A deep bidirectional lstm-gru network model for automated ciphertext classification," *IEEE Access*, vol. 10, 2022.

[6] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze, "Comparative study of cnn and rnn for natural language processing," 2017.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics*, 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[10] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing" Huang, "How to fine-tune bert for text classification?," in *Chinese Computational Linguistics*, Maosong Sun, Xuanjing Huang, Heng Ji, Zhiyuan Liu, and Yang Liu, Eds., 2019.

[11] Diederik P Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[12] S Zargar, "Introduction to sequence learning models: Rnn, lstm, gru," *Department of Mechanical and Aerospace Engineering, North Carolina State University*, 2021.

[13] Denis Rothman, *Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more*, Packt Publishing Ltd, 2021.

[14] Frank Krüger, *Activity, Context, and Plan Recognition with Computational Causal Behaviour Models*, Ph.D. thesis, 12 2016.