MSDS 458: Artificial Intelligence and Deep Learning

Assignment 3

October 27, 2025

Building test classifiers using recurrent and convolutional architectures

Yingyu Mao

Abstract

In this study, I presented a comparative analysis of recurrent and convolutional neural network architectures for automated text classification, using the AG News topic classification dataset as a benchmark. I evaluated the performance of Fully Connected Networks (FCNN), Simple RNNs, bidirectional LSTMs, GRUs, and 1D Convolutional Neural Networks (CNNs), including advanced variants with dilation and parallel kernels. The results demonstrated that 1D CNNs achieve the optimal balance: delivering superior test accuracy (91.5%) and significantly faster training times due to their parallelizable architecture compared to recurrent models. I also showed that determining the optimal vocabulary size is crucial for different training corpuses. A data-cleaning pipeline, which identifies mislabeled samples via cross-validation, can also benefit model performance. These findings provided guidance for management to adopt a 1D CNN-based model for scalable, efficient, and accurate automated content categorization to streamline editorial operations.

Keywords: recurrent neural network, long short-term memory, 1D convolutional neural network, text classification

1.0 Introduction

Natural Language Processing (NLP) is a subfield of artificial intelligence concerned with the interactions between computers and human language. Its core objective is to enable machines to read, decipher, understand, and make sense of human languages in a valuable and scalable way.

NLP involves a series of complex tasks, ranging from low-level syntactic analysis (e.g., tokenization, part-of-speech tagging) to high-level semantic understanding (e.g., sentiment analysis, topic modeling). Modern NLP leverages deep learning models trained on vast corpora of text data. These models learn the statistical patterns, structures, and contextual relationships, allowing them to perform tasks such as translation, summarization, and information retrieval with increasing complexity. The ultimate challenge lies in grappling with the inherent ambiguity and nuances in human communication.

Text classification is a type of NLP application. It automates the process of assigning predefined categories or tags to unstructured text. Some of the examples use cases are: sentiment analysis to determine the emotional tone (positive, negative, neutral) of customer reviews, social media mentions, or to understand consumer perceptions, emerging trends for market analysis; topic and intent classification for routing support tickets to the correct department (e.g. Github triage); content categorization to organize news articles, blog posts, and documents into thematic categories for improved navigation and discovery.

At a hypothetical media company that runs an online news media platform, the management is interested in replacing manual editorial operations with modern NLP models to streamline their online content categorization. To test out their idea, I proposed to use benchmark dataset such as AG news topic classification dataset to build some prototype models. In this paper, I will compare the performance of fully connected neural networks with various types of recurrent and convolutional architectures on the text classification task.

2.0 Literature review

Early approaches to text classification relied on manually engineered features. Methods like Naïve Bayes, Support Vector Machines (SVMs), and Logistic Regression were applied to text represented as high-dimensional sparse vectors, most commonly using the "Bag-of-Words" (BoW) or TF-IDF representation (Joachims 1998). While effective for their time, these models ignored word order and contextual meaning, limiting their ability to capture complex linguistic patterns.

The introduction of dense, low-dimensional word vectors (e.g., Word2Vec by Mikolov et al., (2013), and GloVe by Pennington et al., (2014)) was a paradigm shift. These embeddings captured semantic and syntactic relationships between words, allowing models to generalize better. They served as a foundational layer upon which deeper architectures could be built. Deep learning models automatically learn hierarchical feature representations from raw text (or embeddings), eliminating the need for manual feature engineering. Recurrent (RNNs) and convolutional (CNNs) architectures emerged as the most effective sequence modeling NNs.

A standard RNN processes text token-by-token, updating its hidden state h_t at each step t based on the current input x_t and the previous hidden state h_{t-1}. The final hidden state (a nonlinear combination of all states) is then used for output. A breakthrough for RNNs was the development of gated architectures, specifically Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Units (GRU) (Cho, et al. 2014). They use gating mechanisms to selectively remember or forget past information, effectively mitigating the vanishing gradient problem and allowing the network to learn long-range dependencies much more effectively than simple RNNs. Another significant advance was the development of bidirectional RNNs (Schuster and Paliwal 1997), which process the sequence in both forward and backward directions. This allows the model to incorporate context from both past and future tokens for any given position, leading to a richer understanding of context. Unlike generative models that cannot be trained on tokens from the future time steps, text classification benefits from using embeddings learned from bidirectional RNNs.

Inspired by their success in computer vision, CNNs were adapted for text by treating sequences as one-dimensional time series (Kalchbrenner, Grefenstette and Blunsom 2014, Kim 2014). A 1D CNN applies a set of filters (kernels) that slide over the sequence of word embeddings. Each filter specializes in detecting the presence of specific local patterns (such as specific phrases or n-grams). The subsequent pooling layers (e.g., max pooling) reduce the dimensionality while retaining the most important features. Unlike the sequential nature of RNNs, the convolutional operations in a CNN are independent across the sequence and is highly parallelizable. This makes CNNs significantly faster to train on GPUs than RNNs. Because of this computational advantage, multilayers of CNNs can be easily trained on massive amount of text data which allows learning of hierarchical patterns in the text. On the other hand, stacking multiple layers of RNNs significantly slows down training. The use of multiple kernel sizes in CNN simultaneously also allows detection of patterns at different granularities (e.g., 2-word, 3-word, and 4-word phrases) in parallel. Dilated convolution captures dependencies across different ranges, creating a more comprehensive feature representation. While information stored in simple RNN hidden units tends to fade away as time progresses, CNNs are robust to the position changes of key phrases, making them highly adept for tasks like text classification. For these reasons, it would be interesting to compare the performance of different architectures of RNN and CNN on text classification.

RNN and CNN paved the way for modern NLP and inspired the invention of attention mechanism and transformer architecture for learning long-range context in generative language models.

3.0 Methods

3.1 Data preprocessing and exploratory data analysis

The AG's news topic classification subset contains 30,000 training samples and 1,900 testing samples for each of the four prediction classes of topics: "World", "Sports", "Business", "Sci/Tech". All text were first converted to lower case, the punctuations removed, followed by removal of a list of English `STOPWORDS` from the `nltk` module. Token label encodings were first generated for all tokens in the corpus using tensorflow TextVectorization layer. There are 2579419 words and 95827 vocabulary words in the corpus of 127600 news articles. Each news article has between 2 and 95 tokens in it, with a median of about 20 tokens as shown in Figure 1 left.

Inspecting 5 documents and their tokenized vector confirmed all tokens ended up with a unique integer identifier (one instance is shown here):

```
Original text: b'AMD #39;s new dual-core Opteron chip is designed mainly for
corporate computing applications, including databases, Web services, and
financial transactions.'
Vectorized text: [1580    2    4 4038 5740  598  636 4754  591 1593  859  169
8056  160  148  238 5908]
```

It appears that some rare words, such as names, places, or specific events that likely carried no relevant information to the news topic were also identified. I then reduced the maximum token to 3000. The UNK token (label value 1) was used to identify rare words (one instance shown here):

```
Original text: b'AMD #39;s new dual-core Opteron chip is designed mainly for
corporate computing applications, including databases, Web services, and
financial transactions.'
Vectorized text: [1580    2    4    1    1  598  636    1  591 1593  859  169
1  160  148  238    1]
```

The distribution of the vocabulary frequencies is shown in Figure 1 right. Using a 5000 token seems sufficient to capture most of the information in the documents.

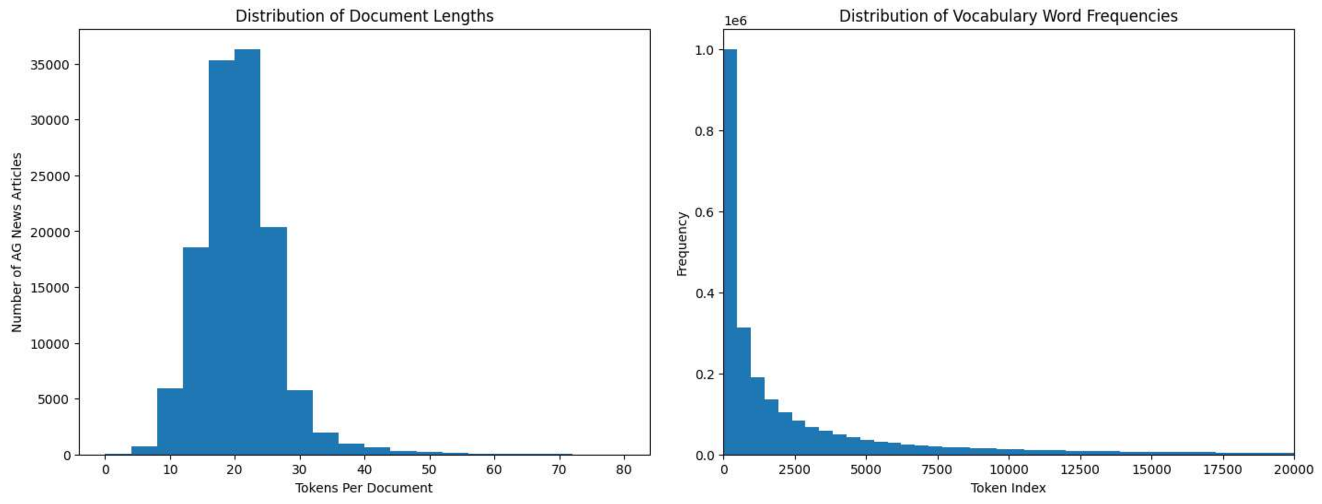


Figure 1 Left: document length distribution. Right: vocabulary frequency distribution.

A quick check of 5 documents reveals that most relevant vocabularies to predict the class label were tokenized (one instance shown here):

```
Original text: b'AMD #39;s new dual-core Opteron chip is designed mainly for
corporate computing applications, including databases, Web services, and
financial transactions.'
Vectorized text: [1580    2    4 4038    1  598  636 4754  591 1593  859  169
1  160  148  238    1]
```

To further reduce noisy tokens that are particularly frequent in this corpus, I then manually selected about 20 words that were seemingly irrelevant to the class labels from the top 50 most frequent vocabularies and added them to the `STOPWORDS` list to generate new tokens for the corpus. The selected 20 words are:

```
['said','new','us','reuters','ap','two','first','monday','wednesday','tuesd
ay','thursday','friday','one','yesterday','last','york','would','sunday','y
ears','week','three','could','saturday','night']
```

3.2 Experimental design

To first determine the best data preprocessing pipeline for downstream training, I compared the performance of a simple RNN model trained on all vectorized datasets generated with different tokenization rules.

Experiment 1: Test different tokenization methods with a simple RNN model

Vectorization layer name	Tokenization rule	NN model
3000	`max_tokens = 3000`	Input: variable length of vectorized text Embedding: 256 dimensions SimpleRNN: 16 units Dense: 4 units
5000	`max_tokens = 5000`	
5000 with added stop words	`max_tokens = 5000` added 20 manually selected frequent words	
10000	`max_tokens = 10000`	

Experiments	**Architecture**
Base model: 3-layer FCNN	Input: variable length of vectorized text Embedding: 256 dimensions GlobalAveragePooling1D Dense-1: 64 units Dense-2: 32 units Dense-3: 4 units
1-layer simple RNN of different units	Input: variable length of vectorized text Embedding: 256 dimensions SimpleRNN: 8/32 units Dense: 4 units
Bidirectional simple RNN	Input: variable length of vectorized text Embedding: 256 dimensions Bidirectional SimpleRNN: 32 units Dense: 4 units
Bidirectional RNN with pretrained embedding layer	Input: variable length of vectorized text GloVe embedding: 100 dimensions Bidirectional SimpleRNN: 32 units Dense: 4 units
Tuned embedding dimension and regularization	Input: variable length of vectorized text Embedding: 50 dimensions, L2 kernel regularizer Bidirectional SimpleRNN: 32 units, L2 kernel and recurrent regularizer Dense: 4 units
1-layer LSTM	Input: variable length of vectorized text Embedding: 256 dimensions Bidirectional LSTM: 32 units Dense: 4 units
1-layer GRU	Input: variable length of vectorized text Embedding: 256 dimensions Bidirectional GRU: 32 units Dense: 4 units
1D CNN	Input: variable length of vectorized text Embedding: 256 dimensions Conv1D: 32 filters Dropout: 0.5 MaxPooling1D: pool size = 2 GlobalMaxPooling1D Dense-1: 256 units Dense-2: 4 units

Cross validation to identify documents with wrong class labels	1D CNN architecture was used for cross validation
Dilated 1D CNN	Input: variable length of vectorized text Embedding: 256 dimensions Parallel Conv1D: 32 filters, dilation rate = [1,2,4,8] GlobalMaxPooling1D Dense: 4 units
Parallel 1D CNN with different kernel size and dilation rate (CNN-DK)	Input: variable length of vectorized text Embedding: 256 dimensions Parallel Conv1D: 32 filters, kernel size = [2,3,4] of dilation rate = [1,2,4,8] GlobalMaxPooling1D Dense: 4 units

All layers used default kernel initiation. All Dense and Conv1D layers used ReLU activation, while RNN units used tanh activation for output and sigmoid activation for recurrence. The output dense layer used Softmax activation. I compared the training time (on Colab default CPU) and per class prediction performance of different models.

4.0 Model Training Results

All models were trained using tensorflow default Nadam optimizer, a variant of Adam in combination with the Nesterov trick. The loss function used was categorical cross entropy.

Early stopping was used to prevent overfitting:

```
tf.keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=3)
```

4.1 Experiment 1: Test different tokenization methods with a simple RNN model

The performance of simple RNN trained with different vectorized datasets are summarized below:

Vectorization layer name	Best model test accuracy	Training time
3000 tokens	0.874	4min 51s
5000 tokens	0.881	5min 13s
5000 tokens with added stop words	0.866	5min 17s
10000 tokens	0.876	6min 7s

As the number of tokens increased, the more complex the data and the training time became longer. The increase of token number only benefited the model performance to certain extent: 10000-token dataset performs worse than 5000-token. As expected, noisy tokens can arise from the two ends of the vocabulary frequency distribution: common words appear in every document that offer no distinguishing features for document labels; extremely rare words that refer to specific events, names, or places also carry no relevance information to the class labels. It seems that 5000 token is a sweet spot that filters out these types of noisy tokens.

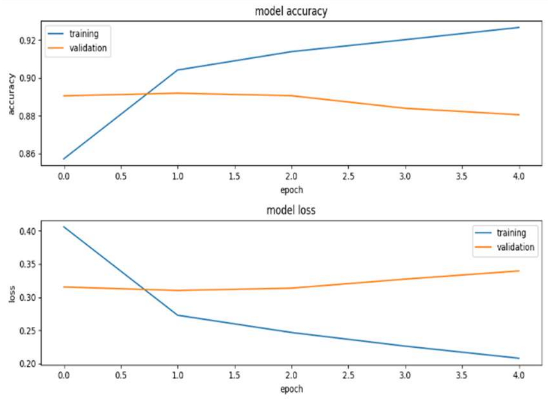
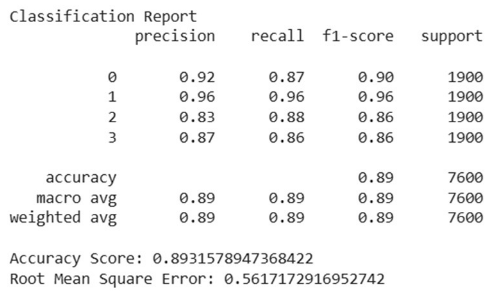
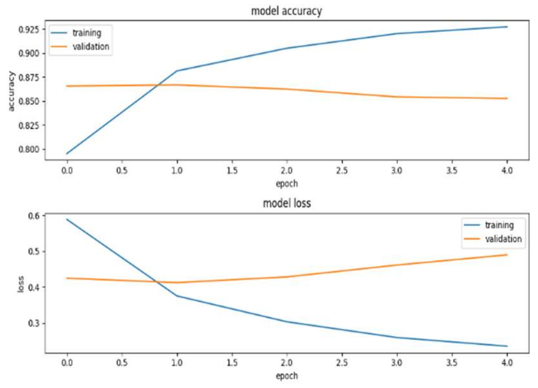
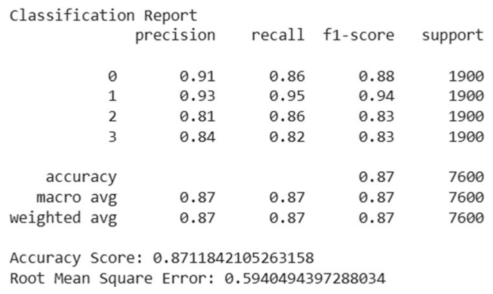
Interestingly, manually deleting some seemingly irrelevant frequent tokens resulted in worse model performance. It could be because some implicit dependencies were disrupted by this type of operation. It also suggests that subjective and hard filtering of token-level noise might not be beneficial for subsequent learning.

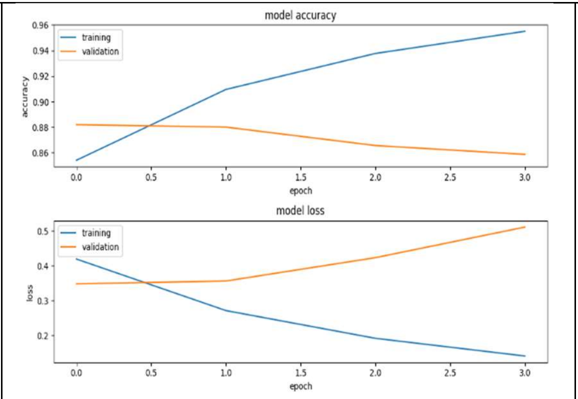
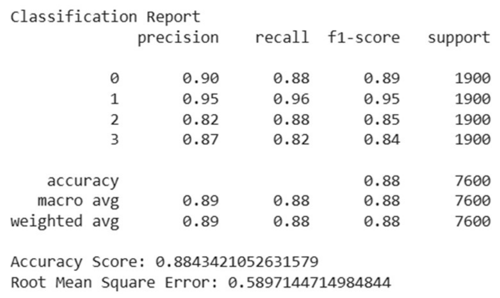
The 5000-token vectorization layer was selected for subsequent training.

4.2 Experiment 2: simple RNN with different units

A baseline fully connected network was first built. It started with the same input and embedding layer followed by a `GlobalAveragePooling1D` operation, which is taking a weighted sum of the token embeddings (the context) in a document where the weights are equal, differing from the nonlinear transformation of the context in RNNs and 1D CNN.
Based on the 16-unit model of simple RNN, I altered the unit number to 8 and 32. Less hidden dimension resulted in worse performance, while higher hidden dimension resulted in better performance (Table 1). Overall, they performed worse than simply averaging the embeddings of the tokens. The performance of the validation set also deteriorated over time, indicating overfitting.

Table 1 Performance of simple RNNs

Experiments	**Training time**	**Training history**	**Performance on test set**
Base model: 3-layer FCNN	2min 40s		Classification Report precision recall f1-score support 0 0.92 0.87 0.90 1900 1 0.96 0.96 0.96 1900 2 0.83 0.88 0.86 1900 3 0.87 0.86 0.86 1900 accuracy 0.89 7600 macro avg 0.89 0.89 0.89 7600 weighted avg 0.89 0.89 0.89 7600 Accuracy Score: 0.8931578947368422 Root Mean Square Error: 0.5617172916952742
1-layer simple RNN 8 units	5min 33s		Classification Report precision recall f1-score support 0 0.91 0.86 0.88 1900 1 0.93 0.95 0.94 1900 2 0.81 0.86 0.83 1900 3 0.84 0.82 0.83 1900 accuracy 0.87 7600 macro avg 0.87 0.87 0.87 7600 weighted avg 0.87 0.87 0.87 7600 Accuracy Score: 0.8711842105263158 Root Mean Square Error: 0.5940494397288034

1-layer simple RNN 32 units	5min 19s		

```
Classification Report
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      1900
           1       0.95      0.96      0.95      1900
           2       0.82      0.88      0.85      1900
           3       0.87      0.82      0.84      1900

    accuracy                           0.88      7600
   macro avg       0.89      0.88      0.88      7600
weighted avg       0.89      0.88      0.88      7600

Accuracy Score: 0.8843421052631579
Root Mean Square Error: 0.5897144714984844
```

4.3 Experiment 3-5: Bidirectional simple RNNs

Adding bidirectional operation to the simple RNN improved its prediction performance but increased training time (Table 2), as two sets of RNN units learned the context in both forward and backward directions.

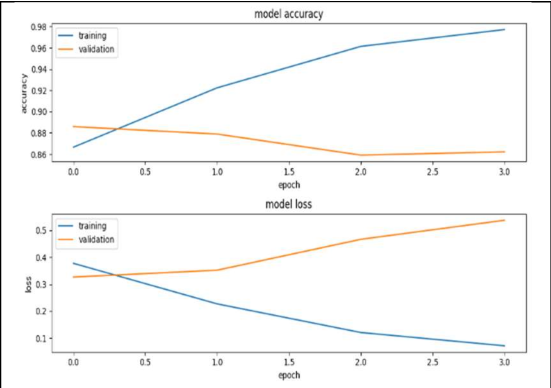
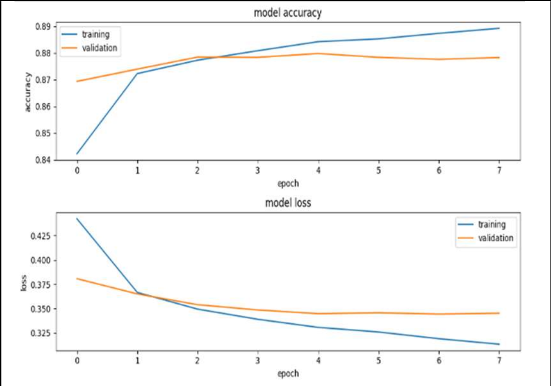
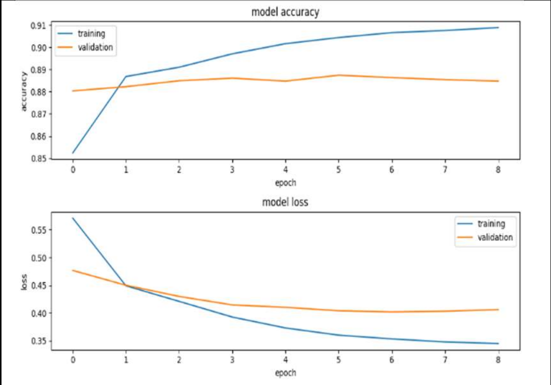
To understand how token embeddings affect downstream learning, a pretrained word embedding matrix, GloVe, was used instead of training the embedding layer from scratch. With 100 dimensions, the pretrained word embedding resulted in worse test accuracy (Table 2). However, the learning trajectory of the validation set appeared to converge better than using the vanilla embedding layer, indicating less tendency to overfit.

Next, I added L2 regularizations to the embedding layer, the kernel, and the recurrent weight matrix of the simple RNN units. The regularization strength and embedding dimensions were tuned using `keras tuner` hyperband search. The resulting new architecture did not result in better prediction performance but increased training time (Table 2). The accuracy of the validation set held steady during training, which means the regularization was effective. However, since the prediction performance is comparable to its early-stopping-regularized counterpart, implementing early stopping is easier than tuning L2 regularizations.

Interestingly, none of the simple RNNs trained so far exceeded the performance of a FCNN that learns from simple average of token embeddings within a document. This could be because the embedding dimension we chose was relatively large (256 for only 5000 tokens, and much larger than the median document length), which led to relatively sparse representations. When averaged over 80 tokens, each dimension was still able to keep some information contributed by each token to the context.

Table 2 Performance of bidirectional simple RNNs

Experiments	**Training time**	**Training history**	**Performance on test set**

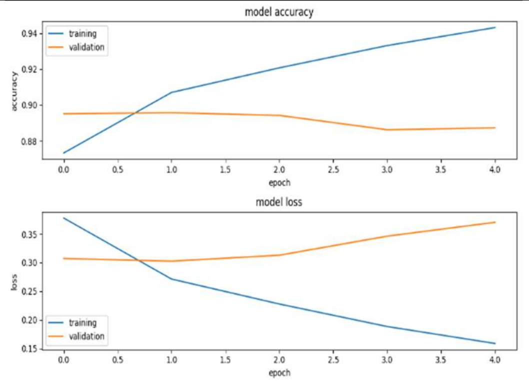
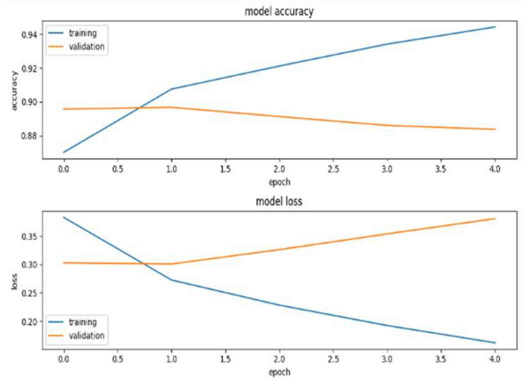
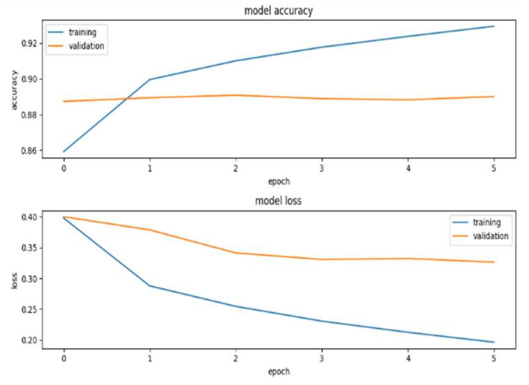
Bidirectional simple RNN	7min 35s		Classification Report precision recall f1-score support 0 0.91 0.89 0.90 1900 1 0.95 0.96 0.95 1900 2 0.82 0.88 0.85 1900 3 0.86 0.82 0.84 1900 accuracy 0.89 7600 macro avg 0.89 0.89 0.89 7600 weighted avg 0.89 0.89 0.89 7600 Accuracy Score: 0.8860526315789473 Root Mean Square Error: 0.5855721448652635
Bidirectional RNN with pretrained embedding layer	9min 4s		Classification Report precision recall f1-score support 0 0.90 0.86 0.88 1900 1 0.94 0.95 0.95 1900 2 0.84 0.83 0.83 1900 3 0.81 0.85 0.83 1900 accuracy 0.87 7600 macro avg 0.87 0.87 0.87 7600 weighted avg 0.87 0.87 0.87 7600 Accuracy Score: 0.8732894736842105 Root Mean Square Error: 0.631101793440813
Tuned embedding dimension and regularization	11min 21s		Classification Report precision recall f1-score support 0 0.91 0.89 0.90 1900 1 0.95 0.95 0.95 1900 2 0.82 0.88 0.85 1900 3 0.87 0.82 0.84 1900 accuracy 0.88 7600 macro avg 0.89 0.88 0.88 7600 weighted avg 0.89 0.88 0.88 7600 Accuracy Score: 0.8847368421052632 Root Mean Square Error: 0.5689325184293088

4.4 Experiment 6-8: Bidirectional LSTM, GRU and 1D CNN

To further improve the bidirectional architecture, I used LSTM and GRU units in place of the simple RNN to allow the model to learn long-range context. The added long-term memory increased training time. Their prediction performance started to match that of a FCNN that simply averaged the embeddings of all tokens in a document.

Like bidirectional RNNs, 1D CNN learns the context independent of the direction of the text sequence. This benefits the learning for text classification. The 1D CNN with only one Conv1D layer with max-pooling had better test accuracy than all the RNNs tested so far. In addition, its training time was only half of that of LSTM/GRU, because CNNs are parallelizable, independent of time steps.

Table 3 Performance of bidirectional LSTM, GRU, and 1D CNN.

Experiments	**Training time**	**Training history**	**Performance on test set**
1-layer LSTM	10min 48s		`Classification Report` `precision recall f1-score support` `0 0.93 0.87 0.90 1900` `1 0.94 0.97 0.95 1900` `2 0.84 0.88 0.86 1900` `3 0.87 0.85 0.86 1900` `accuracy 0.89 7600` `macro avg 0.89 0.89 0.89 7600` `weighted avg 0.89 0.89 0.89 7600` `Accuracy Score: 0.8936842105263157` `Root Mean Square Error: 0.55310130413115`
1-layer GRU	10min 52s		`Classification Report` `precision recall f1-score support` `0 0.93 0.87 0.90 1900` `1 0.94 0.97 0.95 1900` `2 0.84 0.88 0.86 1900` `3 0.87 0.85 0.86 1900` `accuracy 0.89 7600` `macro avg 0.89 0.89 0.89 7600` `weighted avg 0.89 0.89 0.89 7600` `Accuracy Score: 0.8936842105263157` `Root Mean Square Error: 0.55310130413115`
1D CNN	4min 43s		`Classification Report` `precision recall f1-score support` `0 0.90 0.90 0.90 1900` `1 0.95 0.96 0.96 1900` `2 0.87 0.85 0.86 1900` `3 0.86 0.87 0.86 1900` `accuracy 0.90 7600` `macro avg 0.90 0.90 0.90 7600` `weighted avg 0.90 0.90 0.90 7600` `Accuracy Score: 0.8957894736842106` `Root Mean Square Error: 0.5735393346764044`

For all models trained so far, the prediction accuracy seems to plateau at around 0.895. The confusion matrix of the test set (Figure 2) reveals that most of the misclassification happened in categories ‘Business’ and ‘Sci/Tech’.

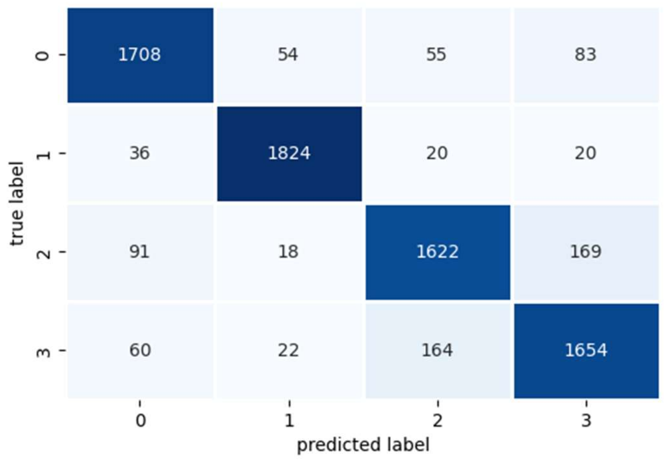


Figure 2 confusion matrix of the test set predicted by 1D CNN

After checking 20 examples of the misclassified documents from these two classes, I noticed most of them were either mislabeled or carried insufficient information (one document only had 5 words) or were ambiguous even for a human to determine its category. Some of the mislabeled examples are shown below but more can be found in the python notebook:

```
Examples of misclassified Business and Sci/Tech articles:
--- Example 1 ---
True Class: 3 (Sci/Tech)
Predicted Class: 2 (Business)
Article: b'Geneva - Worldwide sales of industrial robots surged to record
levels in the first half of 2004 after equipment prices fell while labour
costs grew, the United Nations Economic Commission for Europe said in a
report to be released today.' # Because the mentions of prices, labor cost,
industry, this sounds more like a business article to me.
--------------------
--- Example 2 ---
True Class: 3 (Sci/Tech)
Predicted Class: 2 (Business)
Article: b'Individually theyve been unstoppable in their respective
industries. Theyre both legends that have survived the dot com burst and came
out winners.' # The keywords industry, dot com burst, make this more like a
business article.
--------------------
--- Example 3 ---
True Class: 3 (Sci/Tech)
Predicted Class: 2 (Business)
Article: b'Ivan, Frances and Charley delivered three staggering blows to the
Gulf Coast and Florida, as well as Caribbean island nations, all in just five
weeks.' # It's not clear Ivan, Frances and Charley are hurricane names even
to a human reader. That would make this article in the 'World' category
instead of 'Sci/Tech'.
```

4.5 Experiment 9: Cross validation to identify documents with wrong class labels in the corpus

No model can learn the desired patterns on mislabeled data. To rectify this, I explored options to identify mislabeled or ambiguous documents in the entire corpus. Inspired by Lin et al.'s work on identifying noisy tokens using cross entropy loss to set up a cutoff (Lin, et al. 2024), I identified documents with high loss after 3 rounds of 10-fold cross validation using the best-performing model so far, the 1D CNN model. The maximum loss distribution of all documents is shown in Figure 3.

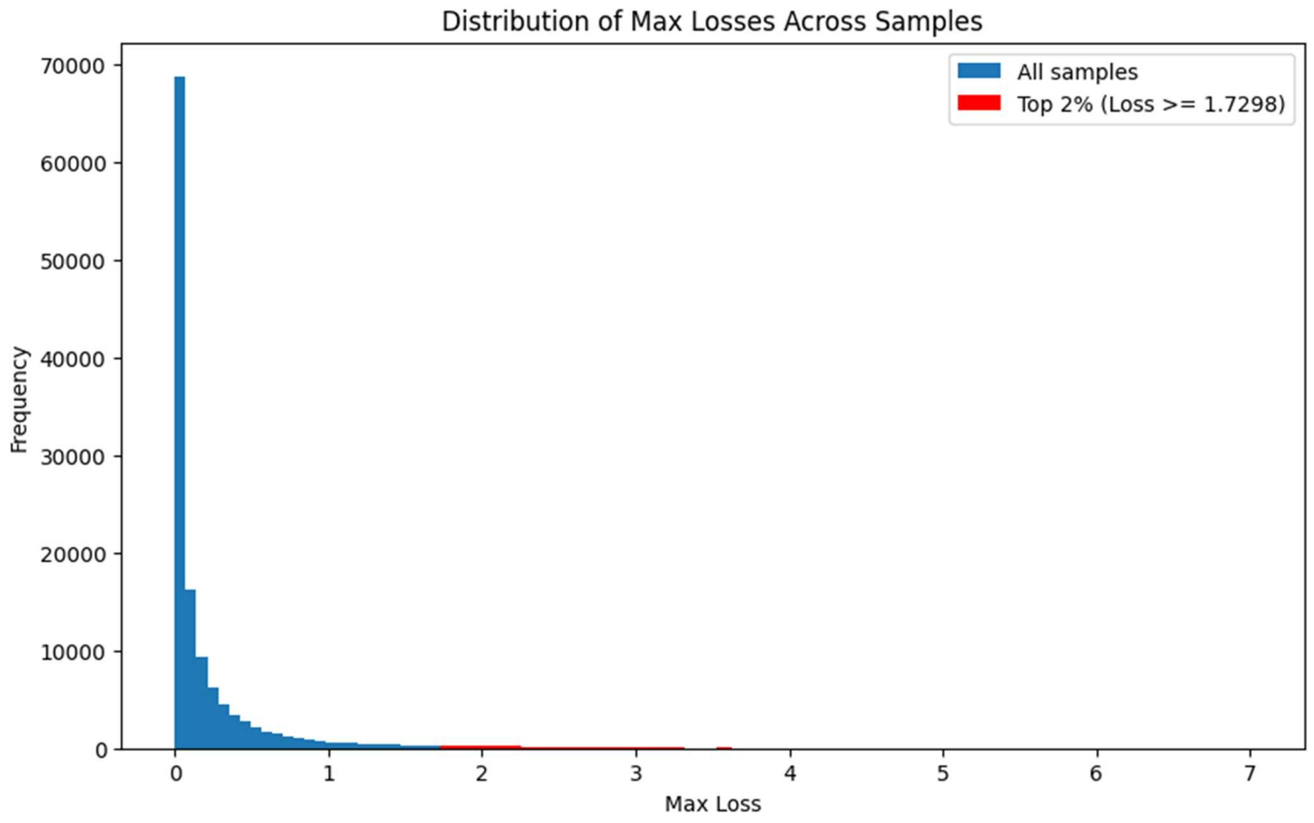


Figure 3 Distribution of the maximum losses of all documents.

The lower bound for the loss of the top 2% of documents with high maximum losses was 1.73, which corresponds to about 17.7% predicted probability. This means that documents with maximum loss larger than 1.73 received a minimum predicted probability less than 17.7% for their true class. Using this rule, I found that documents with the highest maximum loss are mostly mislabeled documents. Some examples are shown below:

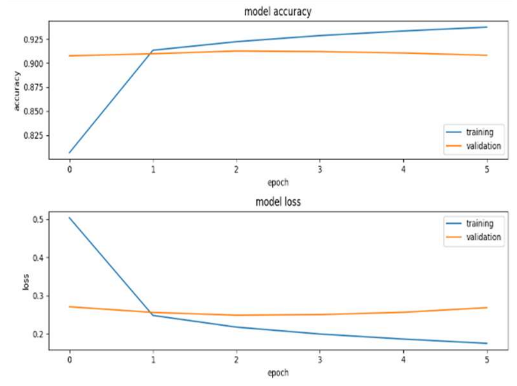
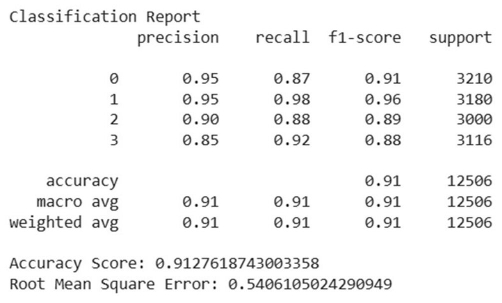
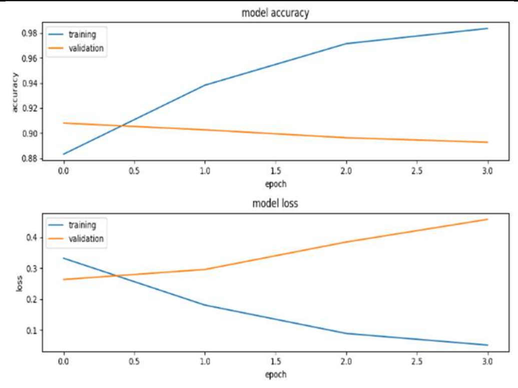
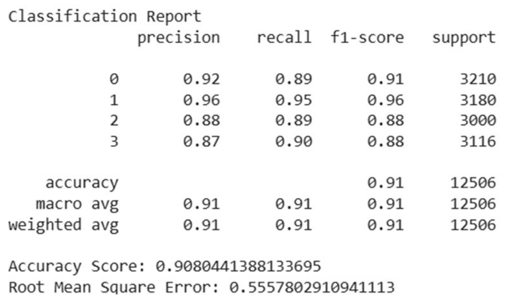
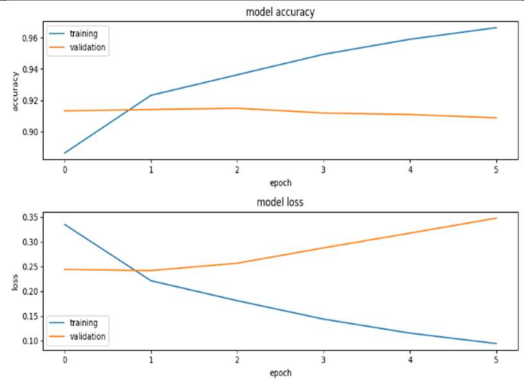
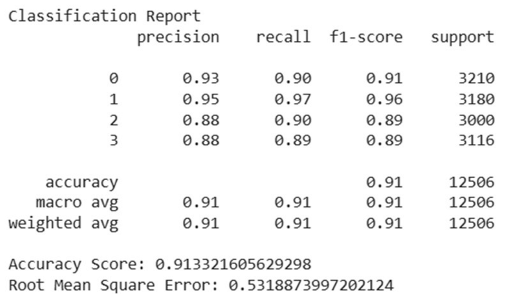
```
--- Sample (Loss: 7.0066, True Label: 3 (Sci/Tech)
Article: The Philippines Saturday expressed quot;deepest sympathy quot; to
the families of the dead in the Russian school siege on Friday, in which 322
people were killed when Russian troops stormed
--- Sample (Loss: 6.8475, True Label: 3 (Sci/Tech)
Article: Asian stocks had the biggest gain in seven months on optimism the
US economy is growing after the Federal Reserve raised interest rates and a
government report showed an unexpected increase in retail sales last month.
--- Sample (Loss: 6.6154, True Label: 3 (Sci/Tech)
Article: AP - NBA star Kobe Bryant arrived at his sexual assault trial Monday
as attorneys in the case who spent the weekend poring over questionnaires
prepared to question potential jurors individually.
```

At the lower end of these 2% documents (2552 samples), there are still some mislabeled documents (examples in python). I then discarded these documents and split the corpus again into training/validation/test sets with a ratio of 80:10:10. The FCNN, bidirectional RNNs, and 1D CNN model were retrained with the new corpus, and their performance is listed in Table 4. No mislabeled documents were found after inspecting 20 misclassified documents by 1D CNN trained with the new corpus.

4.6 Experiment 10: Dilated CNN with multiple kernel sizes

The kernel size of 1D CNN controls the level of granularity each neuron focuses on, while dilation rate captures dependencies between tokens at different distances. I tested parallel 1D CNN architectures with either varying dilation rates or both varying dilation rates and kernel sizes (CNN-DK). The dilated CNN had similar performance as the baseline 1D CNN, while CNN-DK improved the performance on top of the dilated CNN, surpassing that of the bidirectional GRU model.

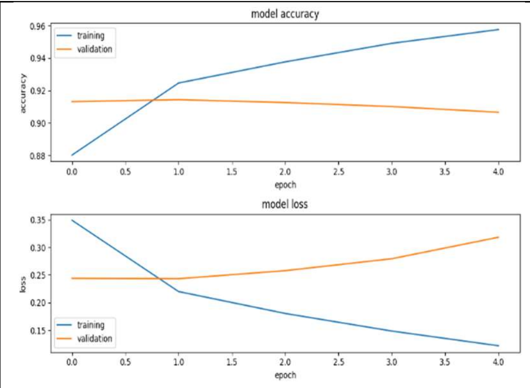
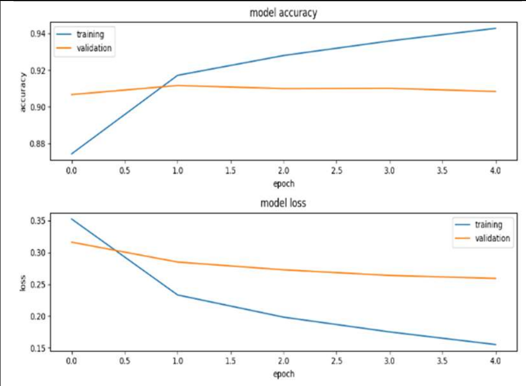
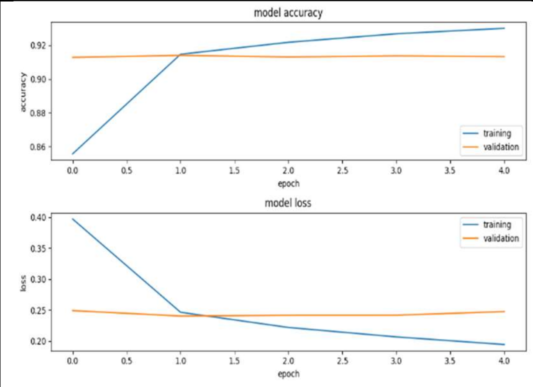
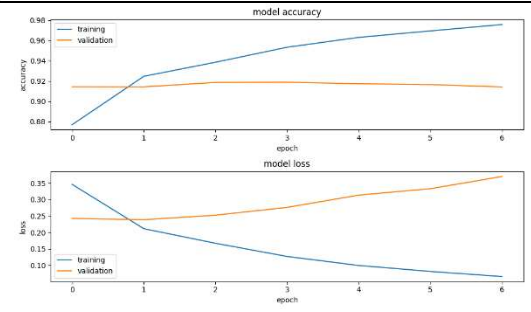
Table 4 Performance of all models on cleaned corpus.

Experiments	**Training time**	**Training history**	**Performance on test set**
FCNN	3min 45s		
Bidirectional simple RNN	9min 2s		
Bidirectional LSTM	26min 38s		

```
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.87      0.91      3210
           1       0.95      0.98      0.96      3180
           2       0.90      0.88      0.89      3000
           3       0.85      0.92      0.88      3116

    accuracy                           0.91     12506
   macro avg       0.91      0.91      0.91     12506
weighted avg       0.91      0.91      0.91     12506

Accuracy Score: 0.9127618743003358
Root Mean Square Error: 0.5406105024290949
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.92      0.89      0.91      3210
           1       0.96      0.95      0.96      3180
           2       0.88      0.89      0.88      3000
           3       0.87      0.90      0.88      3116

    accuracy                           0.91     12506
   macro avg       0.91      0.91      0.91     12506
weighted avg       0.91      0.91      0.91     12506

Accuracy Score: 0.9080441388133695
Root Mean Square Error: 0.5557802910941113
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.90      0.91      3210
           1       0.95      0.97      0.96      3180
           2       0.88      0.90      0.89      3000
           3       0.88      0.89      0.89      3116

    accuracy                           0.91     12506
   macro avg       0.91      0.91      0.91     12506
weighted avg       0.91      0.91      0.91     12506

Accuracy Score: 0.913321605629298
Root Mean Square Error: 0.5318873997202124
```

Bidirectional GRU	20min 16s		Classification Report precision recall f1-score support 0 0.93 0.90 0.91 3210 1 0.95 0.97 0.96 3180 2 0.89 0.90 0.89 3000 3 0.89 0.89 0.89 3116 accuracy 0.91 12506 macro avg 0.91 0.91 0.91 12506 weighted avg 0.91 0.91 0.91 12506 Accuracy Score: 0.9141212218135295 Root Mean Square Error: 0.5318122266013691
1D CNN	11min 39s		Classification Report precision recall f1-score support 0 0.93 0.89 0.91 3210 1 0.94 0.97 0.96 3180 2 0.87 0.90 0.89 3000 3 0.88 0.87 0.88 3116 accuracy 0.91 12506 macro avg 0.91 0.91 0.91 12506 weighted avg 0.91 0.91 0.91 12506 Accuracy Score: 0.9085239085239085 Root Mean Square Error: 0.5420138246268986
Dilated 1D CNN	24min 27s		Classification Report precision recall f1-score support 0 0.94 0.88 0.91 3210 1 0.95 0.98 0.96 3180 2 0.86 0.91 0.88 3000 3 0.89 0.87 0.88 3116 accuracy 0.91 12506 macro avg 0.91 0.91 0.91 12506 weighted avg 0.91 0.91 0.91 12506 Accuracy Score: 0.9095634095634095 Root Mean Square Error: 0.5448096250863766
Parallel 1D CNN with different kernel size and dilation rate (CNN-DK)	4min 26s after switching 16GB T4 GPU		Classification Report precision recall f1-score support 0 0.94 0.89 0.91 3210 1 0.95 0.98 0.96 3180 2 0.88 0.91 0.89 3000 3 0.89 0.88 0.89 3116 accuracy 0.92 12506 macro avg 0.92 0.92 0.91 12506 weighted avg 0.92 0.92 0.92 12506 Accuracy Score: 0.9153206460898768 Root Mean Square Error: 0.5326385473896539

5.0 Conclusion

I evaluated neural network architectures to automate news categorization for the AG News dataset. The goal was to identify a model that balances high accuracy with computational efficiency for a production environment.

Key findings indicate that 1D Convolutional Neural Networks (CNNs) are the most suitable architecture. A standard 1D CNN achieved top accuracy (90.8%) and trained twice as fast as recurrent models like LSTMs due to its parallelizable nature. Advanced variants using parallel kernels with different dilation rates (CNN-DK) achieved even higher performance (91.5%). While bidirectional LSTMs and GRUs were effective, their longer training times make them less ideal for training more complex models. I also found that a 5,000-token vocabulary was optimal for this corpus, and that manual stop-word removal was counterproductive.

A significant discovery was that model performance plateaued due to mislabeled data in the corpus. I implemented a cross-validation method to identify these problematic samples and retraining on a cleaned dataset consistently improved all models.

Recommendations for Management:

- Adopt a 1D CNN model (standard or parallel CNN-DK or CNN with deep hierarchy) for the initial deployment. It offers the best balance of speed and accuracy for text classification.
- Implement a data-cleaning pipeline to continuously identify and review mislabeled articles in new corpus, ensuring long-term model quality.
- Allocate GPU resources for training and inference to capitalize on the CNN's high parallelizability and reduce operational latency.

This CNN-based approach will effectively automate content categorization, reduce editorial workload, and improve content discoverability on their online platform.

References

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." *arXiv* 1406.1078.

Hochreiter, S., and J. Schmidhuber. 1997. "Long short-term memory." *Neural computation* 9(8), 1735-1780.

Joachims, T. 1998. "Text categorization with support vector machines: Learning with many relevant features." *In European conference on machine learning* 137-142.

Kalchbrenner, N., E. Grefenstette, and P. Blunsom. 2014. "A convolutional neural network for modelling sentences." *arXiv* 1404.2188.

Kim, Y. 2014. "Convolutional neural networks for sentence classification." *arXiv* 1408.5882.

Lin, Zhenghao, Zhibin Gou, Yeyun Gong, Xiao Liu, Yelong Shen, Ruochen Xu, Chen Lin, et al. 2024. "Rho-1: Not All Tokens Are What You Need." *arXiv* 2404.07965.

Mikolov, T., K. Chen, G. Corrado, and J. Dean. 2013. "Efficient estimation of word representations in vector space." *arXiv* 1301.3781.

Pennington, J., R. Socher, and C. Manning. 2014. "GloVe: Global Vectors for Word Representation." *In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* 1532-1543.

Schuster, M., and K. K. Paliwal. 1997. "Bidirectional recurrent neural networks." *IEEE transactions on Signal Processing* 45(11), 2673-2681.