ABSTRACT

This paper is continued with creating deep neural networks to apply in natural language processing (NLP), and to analyze how various factors affect the fitting and ultimate test set performance of these networks. Alternative network structures/topologies between sequence models (i.e., recurrent neural networks) and one-dimensional convolutional neural network are explored. The key assignment objective is for us to gain further practical experience with building deep neural networks, especially with commonly used NLP algorithms. As an extension to the modeling exercise, management recommendation is provided for developing a sequential model as a conversational agent to assist in customer support function.

1. INTRODUCTION

In this exercise, various neural network topologies derived from Deep Neural Networks (DNN) and Recurrent Neural Networks (RNN) are built, fitted, and tested in the Python coding framework. Attempts were made to compare and evaluate alternative network structures/topologies, including 1) text/corpus preprocessing, 2) recurrent neural networks with and without long short-term memory, and 3) recurrent neural networks versus one-dimensional convolutional neural network. RNN, in particular, have been proven very effective in areas of NLP used to process, understand and generate natural language texts. While DNN and CNN operations were explored in previous assignments, RNN as a sequence model is newly introduced through this module assignment. RNN adopts the same principle in a simplified manner of how human intelligence processes information incrementally, while deciphering meaning based on new information built on past information. The RNN works by "iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far" (Chollet, 2017). The RNN is essentially a for loop that recalls quantities computed in past steps, and the simple RNN architecture utilized in this assignment Experiment B is illustrated in figure 1. The simple RNN takes in a batch of sequence as its input with shape (batch_size, timesteps, input_features) for each cell. Furthermore, variants of Long Short-Term Memory (LSTM) architecture as an improved recurrent network are explored in Experiment C and they are compared with the simple RNN counterparts. The LSTM algorithm was first developed by Hochreiter and Schmidhuber (1997) as a solution to the vanishing gradient problem that is present when RNN models attempt to learn long-term dependencies. The LSTM layer is a variant of the simple RNN layer, and includes a way to carry information forward and retain information across many timesteps as shown by the "carry track" in figure 2. The mechanism of LSTM is briefly discussed in the METHODS section. Across the different network topologies in all experiments, the final output of the recurrent or convolution layer is passed forward to the final dense, classification layer.

The RNN architecture/hyperparameter variations are examined in this paper for each of their training requirement and performance on the AG's news topic classification data - a subset of AG Corpus (AG's Corpus of News Articles, n.d.) – sourced from Google TensorFlow. The data subset is constructed by choosing 4 largest classes, namely "World", "Sports", "Business", and "Science/Technology" from the original corpus. Each class comprises 30,000 training samples and 1,900 testing samples, with a total 120,000 articles as training samples and 7,600 articles as testing samples. Extensive exploratory data analysis (EDA) was conducted on the corpus in Experiment A, which will be further detailed in the Methods and Results sections of this paper.

After the EDA was completed, 11 modeling experiments were conducted: Experiment set B1 through B5 are variations of simple RNN architecture by tweaking hyperparameters; similarly, Experiment set C1 through C5 are variations of LSTM architecture by tweaking hyperparameters. The remaining Experiment D tests out a one-dimensional (1D) convolutional neural network's ability in sequence processing. The 1D CNN operation is examined in further details in the METHODS section with a focus on text classification application.

In addition, the models in Experiment B4, C4, and D with the "best performance" among those tested are further analyzed through constructing a confusion matrix of the predictions for the test set. For each of these "best models", T-distributed Stochastic Neighbor Embedding (t-SNE) is utilized to reduce the AG Corpus subset dimensionality to allow for visualization of each label class on a 2D graph (van der Maaten and Hinton, 2007).

This assignment allows us to foray deeper into the techniques of building more advanced deep learning networks intended for sequence and temporal processing. More importantly, the experience gained from building the sequence models is supplemented by applying the models into real-world use case of developing a conversational chatbot in a customer service and support capacity.

2. LITERATURE REVIEW

The fundamental operation theories of deep learning models for text, timeseries, and sequence data processing are presented in chapter 6 of Chollet's (2017) book. The initial idea of low-dimensional word embedding as text preprocessing was by Bengio et al. (2003). The theoretical reasons for the vanishing gradient problem are reviewed by Bengio, Simard, and Paolo (1994), leading up to the LSTM algorithm developed by Hochreiter and Schmidhuber (1997).

3. METHODS

The EDA of the input text corpus is performed with a custom "Vectorization" class, in which the adapt method consists of the following steps (figure 3):

- standardize each sample (usually lowercasing + punctuation stripping)
- split each sample into substrings (usually words)
- recombine substrings into tokens (usually n-grams)
- index tokens (associate a unique int value with each token)
- transform each sample using this index, either into a vector of integers or a dense float vector

The Title and Description columns were combined into a Title_Description column in the training and test datasets before text vectorization, after which the distribution of tokens and encodings were visualized in bar charts built with Python Matplotlib functions.

All sequence model variants covering simple RNN, LSTM, and 1D CNN were built in the Python environment using Keras layers packages imported from Tensorflow. To facilitate a fair comparison, certain parameters were held constant across different models as listed in Table 1. In the one-recurrent layer structure, the RNN and LSTM models consist of 32 output features corresponding to the 32 input features in the one-hot encoded or embedded text input, whereas the stacked RNN and LSTM models consist of a second recurrent layer of 64 output features. On the other hand, the 1D CNN structure is built with two 1D convolution layers of 32 nodes stacked in between 1D max pooling layers. All models are completed with a dense classification layer with 4 nodes that correspond to the four topic classes. Variations of the RNN and LSTM models also incorporate Dropout regularization. Other Python packages including NumPy, Pandas, and Matplotlib are used to help with data manipulation and visualization of the confusion matrix and t-SNE graphs.

The AG Corpus data subset used by the sequence models was imported from TensorFlow datasets directly using the tfds.load function. The training and validation set were split 95% to 5% at a batch size of 100. 40 common words, known as "stop words", listed in the Natural Language Toolkit (NLTK) were removed from the input datasets along with text standardization utilizing a custom function. Another important step before the experiments were run is to turn the tokens into input vectors (figure 4). This was achieved in the experiments either through one-hot encoding in Experiments B1/B2/C1/C2 or using learned-word embeddings in Experiments B3/B4/B5/C3/C4/C5/D. The one-hot encoder is sourced from TensorFlow built-in utilities that returns a one-hot tensor restricted to 1000 most common words/tokens in the dataset, which helps to avoid with very large input vector spaces. On the other hand, the Keras Embedding layer is inserted after the input layer to learn word vectors in a new embedding space as the model is trained. For this assignment, the Embedding layer takes 1000 tokens as input and returns associated vectors in a 256-dimension space. The word representation differences between one-hot word vectors and word embeddings are summarized and illustrated in figure 5. All models are controlled by early stopping with patience parameter of 3 and callback of the best model parameters.

The last technique applied to the recurrent neural networks is the bidirectional training. As RNN variants are notably order and time dependent, reversing the training timesteps can completely change the representations the RNN extracts from the sequence. As explained by

Chollet (2017), “bidirectional RNNs exploit the order sensitivity of RNNs: it consists of using two regular RNNs, each of which processes the input sequence in one direction (chronologically and antichronologically), and then merging their representations. By processing a sequence both ways, a bidirectional RNN can catch patterns that may be overlooked by a unidirectional RNN.” In Experiments B2-B5 and C4-C5, the Keras Bidirectional layer was used to take in a recurrent layer instance (i.e., simple RNN or LSTM).

As for the 1D CNN structure in Experiment D, it follows the same operation mechanism explored in the previous assignment. However, the 1D variant in this assignment extracts local 1D patches (subsequences) from sequences input instead of image tensors (figure 6). 1D convolution is effective in recognizing local patterns in a sequence as explained in Chollet (2017), “Because the same input transformation is performed on every patch, a pattern learned at a certain position in a sentence can later be recognized at a different position, making 1D convnets translation invariant (for temporal translations).” In Experiment D, the Keras Conv1D layer was used to instantiate convolution twice in a stacked manner. The features vector size is set to 5 and 32 nodes in each convolution layer.

The algorithm used to optimize the model weights and biases as per the given data was defined to be stochastic gradient descent (i.e., RMSProp). The loss function to be minimized by the optimizer was selected as “categorical cross entropy” commonly used in a multi-class classification problem. Across all experiments the training batch size was set to constant at 100. The “evaluate” method was used to evaluate the trained model performance – accuracy and loss - on the test set.

The model training and validation performance of the models in Experiments B4, C4, and D was also inspected by plotting the evolution of accuracy and loss in figure 7 as each epoch completes. To further the analysis, a 4x4 confusion matrix was constructed in a heatmap format together with the classification results of the first 20 predictions (figure 8) for these experiments to visualize the occurrence each predicted value against the true value of the label.

A comparison was summarized in Table 2 between models from every experiment to compare their model attributes/hyperparameters, model performance, and model processing time/requirement.

4. RESULTS

From the EDA conducted in Experiment A, the training and test corpus contain balanced classes of texts across 4 news categories. The corpus has been summarized to contain 4,773,046 words across the 127,600 articles. The distribution of tokens per news article was plotted as a positive skewed Gaussian curve between 4 and 177 tokens (figure 9). Three different levels: top 100, 500, and 1000 words were vectorized in a simple comparison. As observed from the distribution plot of percentage of non-tokenized words, it is observed that as the vector size grew from 100 to 1000, the percentage improved from a mean of 60-70% to 30-40% (figure 10). It can be inferred that vector size will have a big impact on the model learning representation.

The model characteristics, performance (training, validation, test accuracy and loss) of each model across Experiment B1 to D are listed in Table 2. It also shows the number of epochs for each model when the training stopped. This table lends insight to understanding how the different sequence model architecture and hyperparameters affect the performance in text classification.

From Table 1, there are several key observations. Firstly, even though the simple RNN structure can achieve final accuracy comparable to LSTM, the training time of a simple RNN recurrent layer is almost ten-fold of that of an LSTM recurrent layer, which is evidenced in the total model training time between Experiment set B and C. Secondly, the bidirectional implementation of both RNN/LSTM resulted in significantly improved model performance than the unidirectional models. Thirdly, adding regularization (dropout) to the RNN/LSTM models in Experiments B4/C4 or stacking them with a second recurrent layer in Experiments B5/C5 did not yield any noticeable performance improvement but at the cost of training time. It is suspected that the one-layer bidirectional models have "learnt all features there are to learn" before overtraining, and without modifying the input sizes (e.g., batch size, token size, vector size) the model prediction accuracy are capped around 86%.

An observation worth noting is the opposite effect of different word vectorization on training time of the simple RNN model versus LSTM model. The training time of each epoch in experiment B3 increased more than 50% than that of B2, but from Experiment C2 to C3 the training efficiency improved slightly with no loss to the model performance. In practice, the undesirable consequence with the RNN model likely does not warrant further investigation, as simple RNN models are rarely used due to their subpar performance. Bidirectional LSTM/GRU and transformer architectures have dominated the NLP field as the state of the art.

Out of the 11 experiments from Table 1, the 1D CNN model has the best tradeoff between model accuracy and training time resource. Without tweaking the input size, the 1D CNN has performed better than all the recurrent model variants. This has all but confirmed the assertion by Chollet (2017) that 1D CNN is known to be a fast alternative to RNNs for simple tasks such as text classification.

Lastly, useful insights were gained from the comparative analysis of the confusion matrices (figure 11) and t-SNE graphs (figure 12) of Experiments B4, C4, and D. Despite the different topologies, these models have reached a very similar accuracy. Furthermore, the models have developed similar capabilities to predict between each class as their confusion matrices follow the same trend. All three models tend to mis-classify to a higher degree between "Business" and "Sci/Tech" labels, which can be verified in the t-SNE graphs as "Business" and "Sci/Tech" classes have a good amount of mapping overlaps in all three models, whereas the "World" and "Sports" classes are more isolated in their own cluster.

5. CONCLUSION

By conducting an extensive EDA on text preprocessing and comparing different sequence model architectures, it is concluded that the recurrent neural network is in general superior in prediction tasks involving sequence and timeseries. The notion of machine learning requires empirical strategies to get to an optimal solution is reinforced, as every problem is unique and there is no one-size-fits-all model approach. There are multiple aspects of the model and data parameters that can be finetuned to break through the model performance metrics attained in this exercise.

Chatbots and conversational agents can utilize machine learning approach, and one of the widely leveraged technique is the Sequnce-to-sequence learning (Sutskever et al., 2014). Essentially, it is a combination of two or more RNNs in a specific architecture to transform one sequence to another - an encoder network condenses an input sequence into a vector, and a decoder network unfolds that vector into a new sequence. The trained model can generate a new output given new input, so that it can be used to address complex prediction problems such as machine translation, image captioning, text summarization, question-answering, etc. For building chatbots, the model will translate the user messages (i.e., sequence) to the chatbot answer (i.e., sequence). The attention mechanism can be further leveraged to improve the decoder to learn to focus over a specific range of the input sequence, perhaps those deemed relevant to the business needs. Recent studies on chatbots built on this type of model have shown that by training on an extensive conversational dataset, the chatbot can learn to answer question, extract relevant information, deduce rudimentary reasoning, and even learn human user traits such as personality and tone. However, this type of model has its own weaknesses as pointed out by Mnasri (2019), and it is suggested that a model aggregation/integration approach is needed to develop a well-rounded chatbot that can handle both generic and task-oriented functions.

Appendix

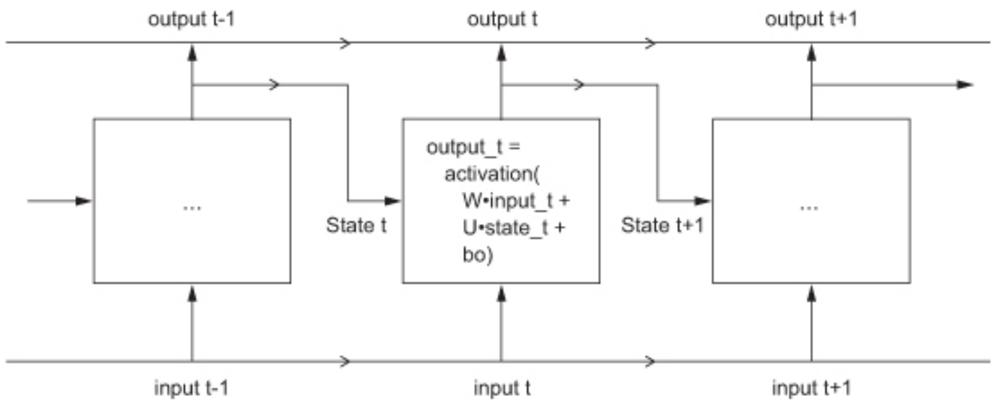


Figure 1. Simple RNN Architecture (adapted from Chollet, 2017)

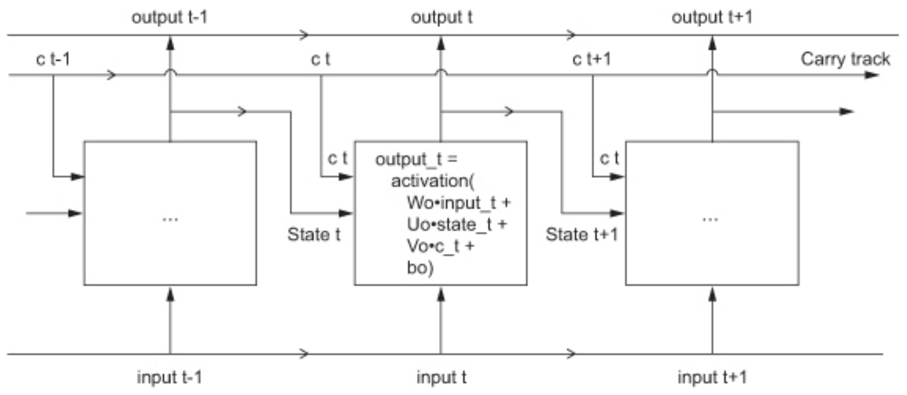


Figure 2. LSTM Architecture (adapted from Chollet, 2017)

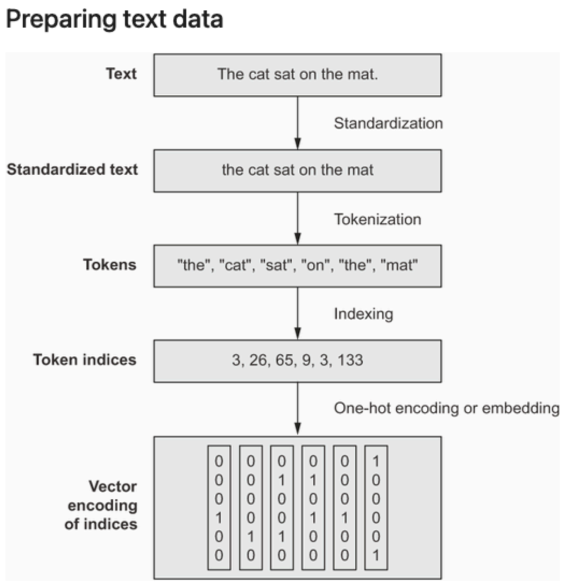


Figure 3. EDA: Text data preprocessing steps (adapted from https://github.com/djp840/MSDS_458_Public/blob/master/MSDS458_Assignment_03/MSDS458_Assignment_03_part00_EDA_v6_20220906.ipynb)

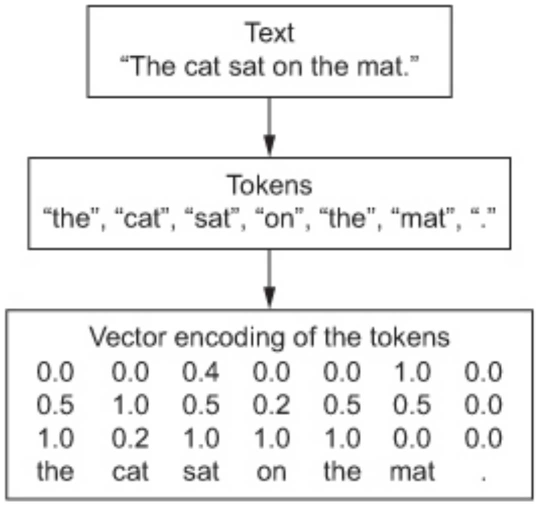

Text
"The cat sat on the mat."
Tokens
"the", "cat", "sat", "on", "the", "mat", "."
Vector encoding of the tokens
the cat sat on the mat .

Figure 4. Text tokenization and vectorization (adapted from Chollet, 2017)

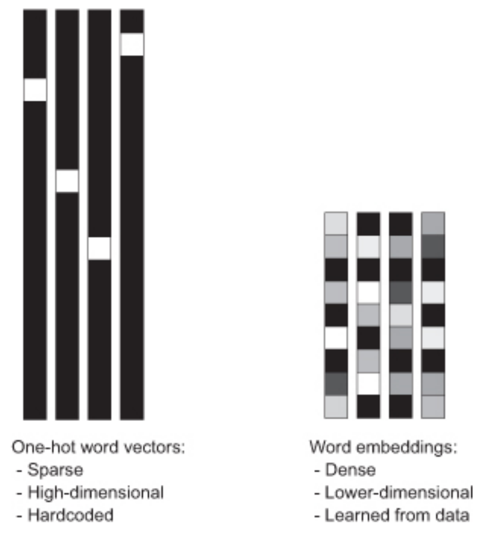


Figure 5. Differences between one-hot encoding and word embedding (adapted from Chollet, 2017)

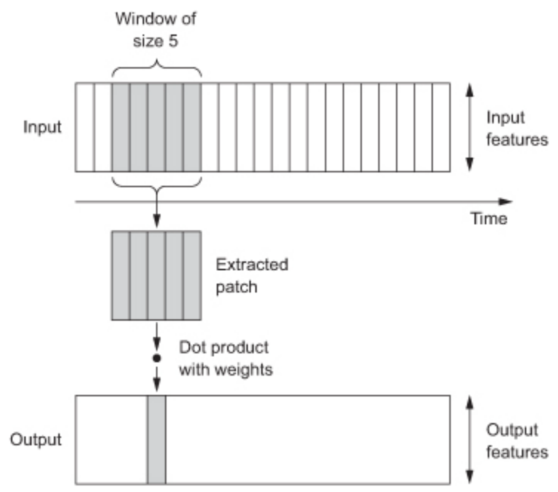


Figure 6. 1D CNN in sequence processing (adapted from Chollet, 2017)

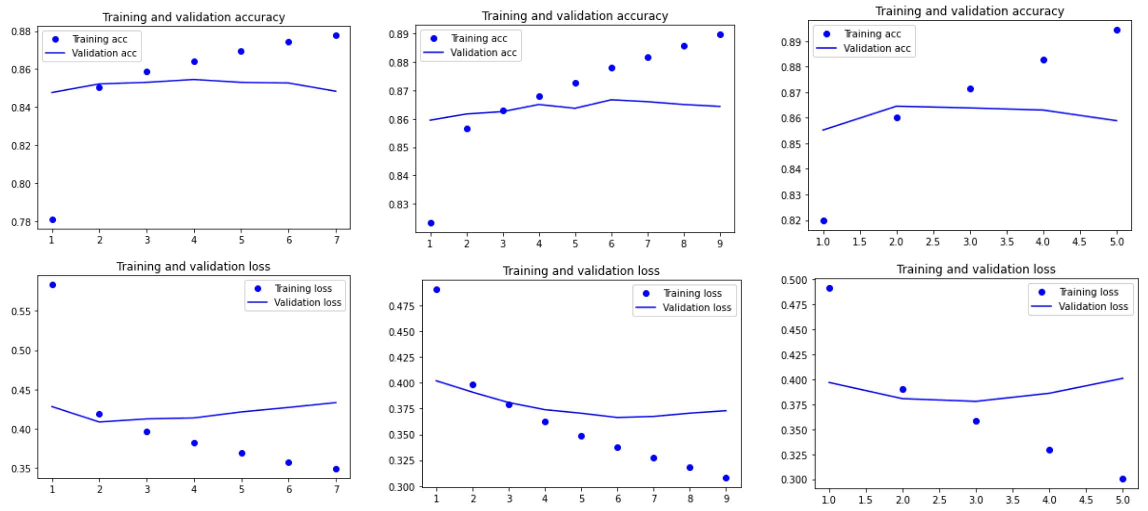
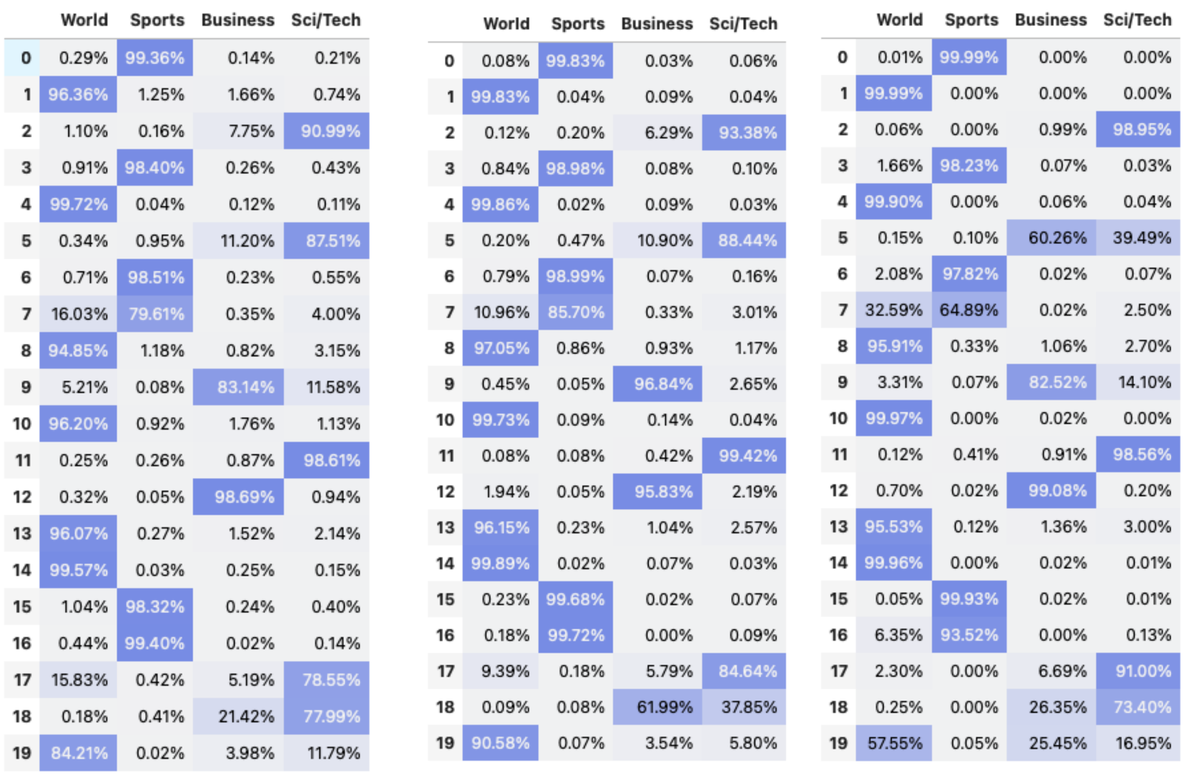


Figure 7. Evolution of model training/validation accuracy and loss per epoch in (left) Experiment B4; (middle) Experiment C4; (right) Experiment D



	World	Sports	Business	Sci/Tech
0	0.29%	99.36%	0.14%	0.21%
1	96.36%	1.25%	1.66%	0.74%
2	1.10%	0.16%	7.75%	90.99%
3	0.91%	98.40%	0.26%	0.43%
4	99.72%	0.04%	0.12%	0.11%
5	0.34%	0.95%	11.20%	87.51%
6	0.71%	98.51%	0.23%	0.55%
7	16.03%	79.61%	0.35%	4.00%
8	94.85%	1.18%	0.82%	3.15%
9	5.21%	0.08%	83.14%	11.58%
10	96.20%	0.92%	1.76%	1.13%
11	0.25%	0.26%	0.87%	98.61%
12	0.32%	0.05%	98.69%	0.94%
13	96.07%	0.27%	1.52%	2.14%
14	99.57%	0.03%	0.25%	0.15%
15	1.04%	98.32%	0.24%	0.40%
16	0.44%	99.40%	0.02%	0.14%
17	15.83%	0.42%	5.19%	78.55%
18	0.18%	0.41%	21.42%	77.99%
19	84.21%	0.02%	3.98%	11.79%

	World	Sports	Business	Sci/Tech
0	0.08%	99.83%	0.03%	0.06%
1	99.83%	0.04%	0.09%	0.04%
2	0.12%	0.20%	6.29%	93.38%
3	0.84%	98.98%	0.08%	0.10%
4	99.86%	0.02%	0.09%	0.03%
5	0.20%	0.47%	10.90%	88.44%
6	0.79%	98.99%	0.07%	0.16%
7	10.96%	85.70%	0.33%	3.01%
8	97.05%	0.86%	0.93%	1.17%
9	0.45%	0.05%	96.84%	2.65%
10	99.73%	0.09%	0.14%	0.04%
11	0.08%	0.08%	0.42%	99.42%
12	1.94%	0.05%	95.83%	2.19%
13	96.15%	0.23%	1.04%	2.57%
14	99.89%	0.02%	0.07%	0.03%
15	0.23%	99.68%	0.02%	0.07%
16	0.18%	99.72%	0.00%	0.09%
17	9.39%	0.18%	5.79%	84.64%
18	0.09%	0.08%	61.99%	37.85%
19	90.58%	0.07%	3.54%	5.80%

	World	Sports	Business	Sci/Tech
0	0.01%	99.99%	0.00%	0.00%
1	99.99%	0.00%	0.00%	0.00%
2	0.06%	0.00%	0.99%	98.95%
3	1.66%	98.23%	0.07%	0.03%
4	99.90%	0.00%	0.06%	0.04%
5	0.15%	0.10%	60.26%	39.49%
6	2.08%	97.82%	0.02%	0.07%
7	32.59%	64.89%	0.02%	2.50%
8	95.91%	0.33%	1.06%	2.70%
9	3.31%	0.07%	82.52%	14.10%
10	99.97%	0.00%	0.02%	0.00%
11	0.12%	0.41%	0.91%	98.56%
12	0.70%	0.02%	99.08%	0.20%
13	95.53%	0.12%	1.36%	3.00%
14	99.96%	0.00%	0.02%	0.01%
15	0.05%	99.93%	0.02%	0.01%
16	6.35%	93.52%	0.00%	0.13%
17	2.30%	0.00%	6.69%	91.00%
18	0.25%	0.00%	26.35%	73.40%
19	57.55%	0.05%	25.45%	16.95%

Figure 8. Top 20 test set class predictions of (left) Experiment B4; (middle) Experiment C4; (right) Experiment D

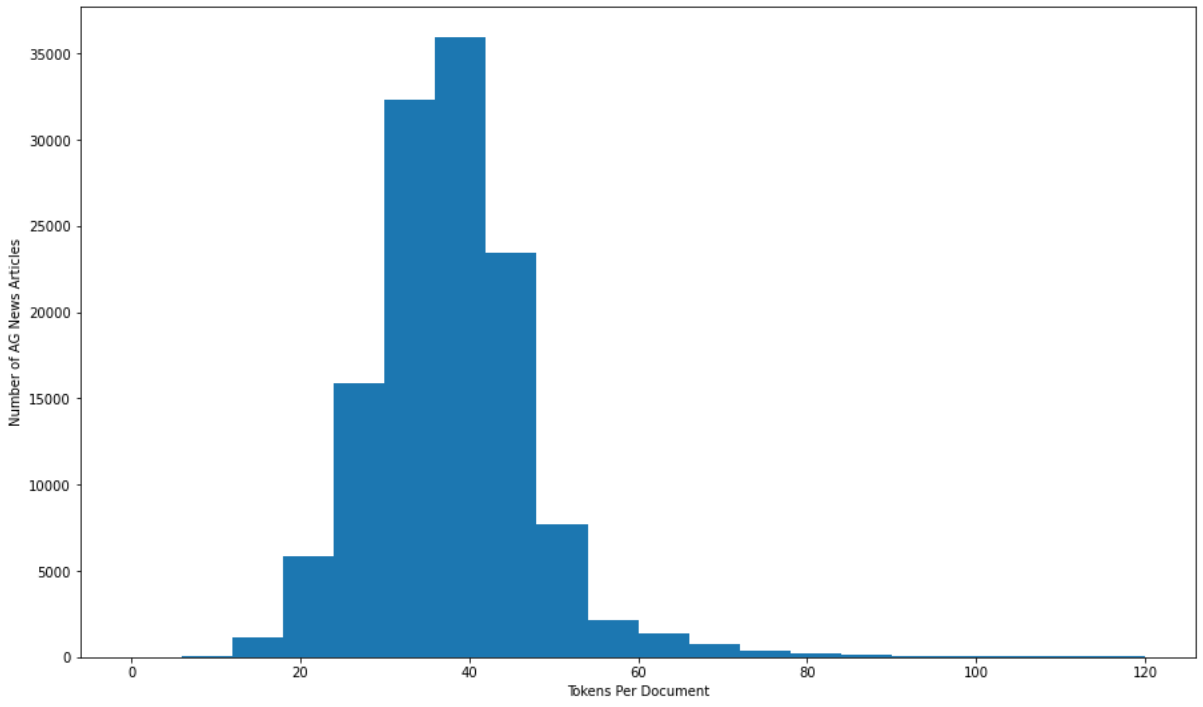
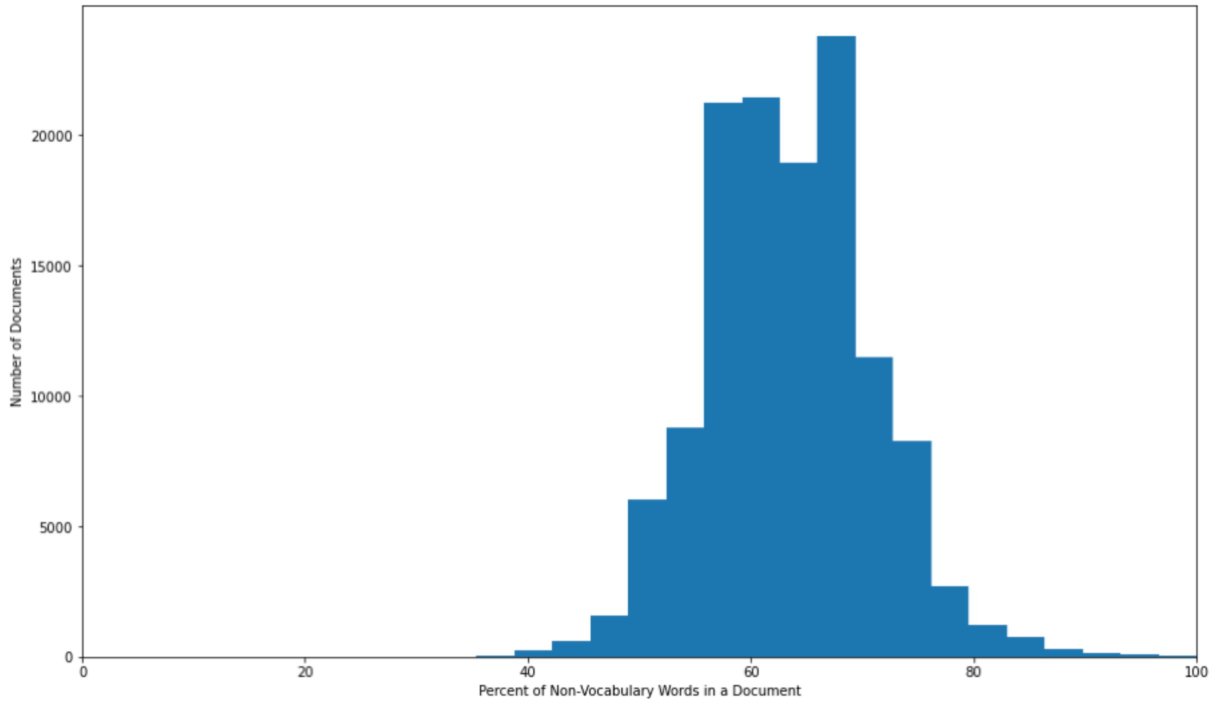


Figure 9. EDA: Distribution plot of tokens per documents in the AG News Corpus subset



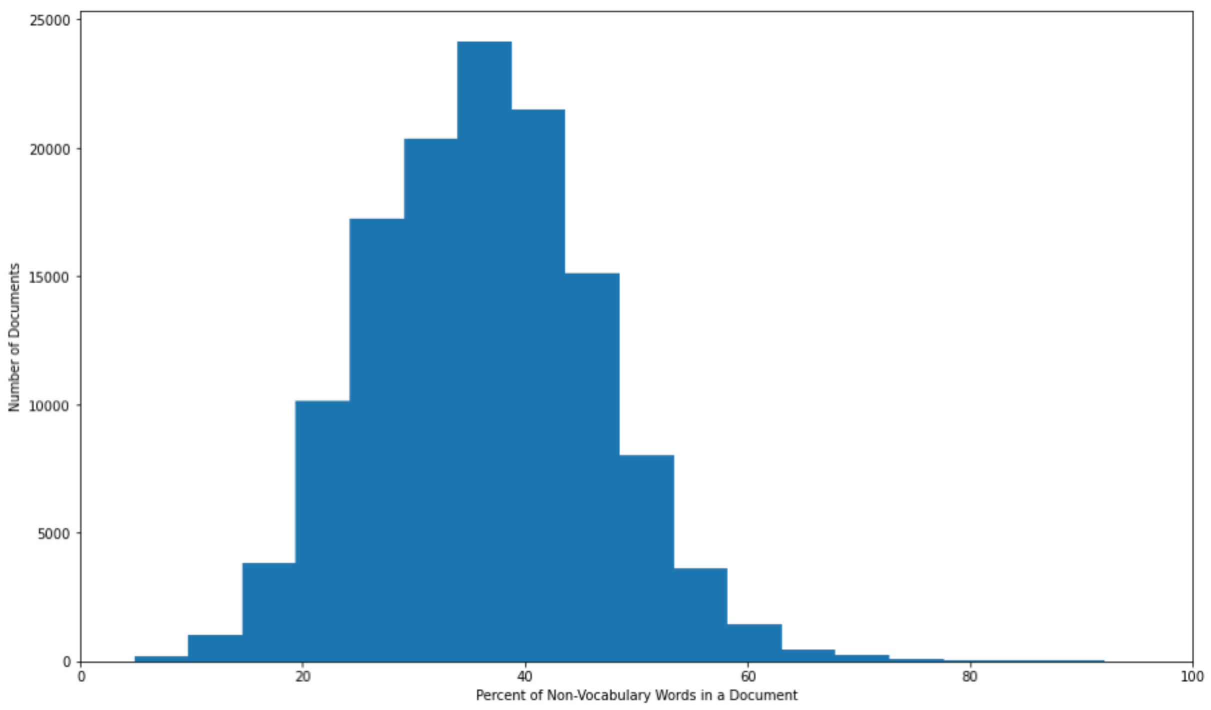
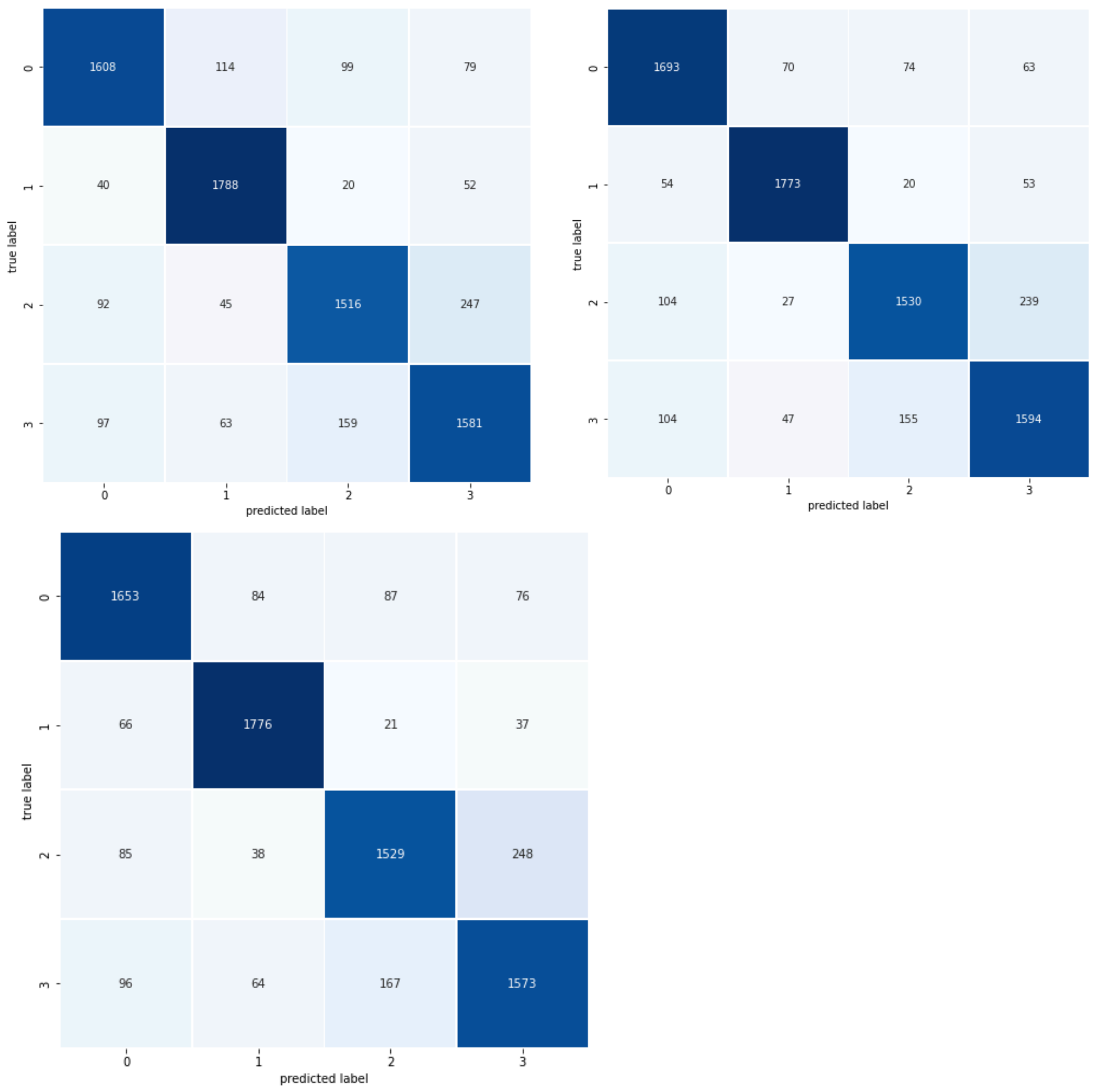


Figure 10. (top) EDA: Distribution plot of percentage of non-vocabulary words AG News articles leveragle 100-word encoder. (bottom) EDA: Distribution plot of percentage of non-vocabulary words AG News articles leveragle 1000-word encoder.

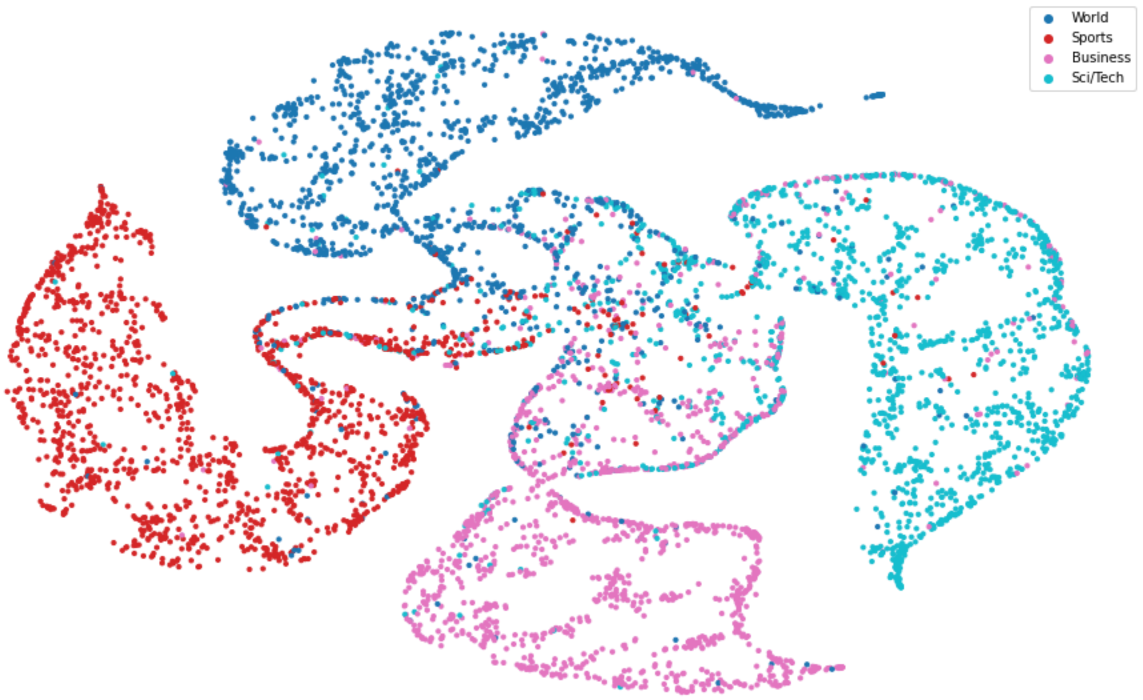


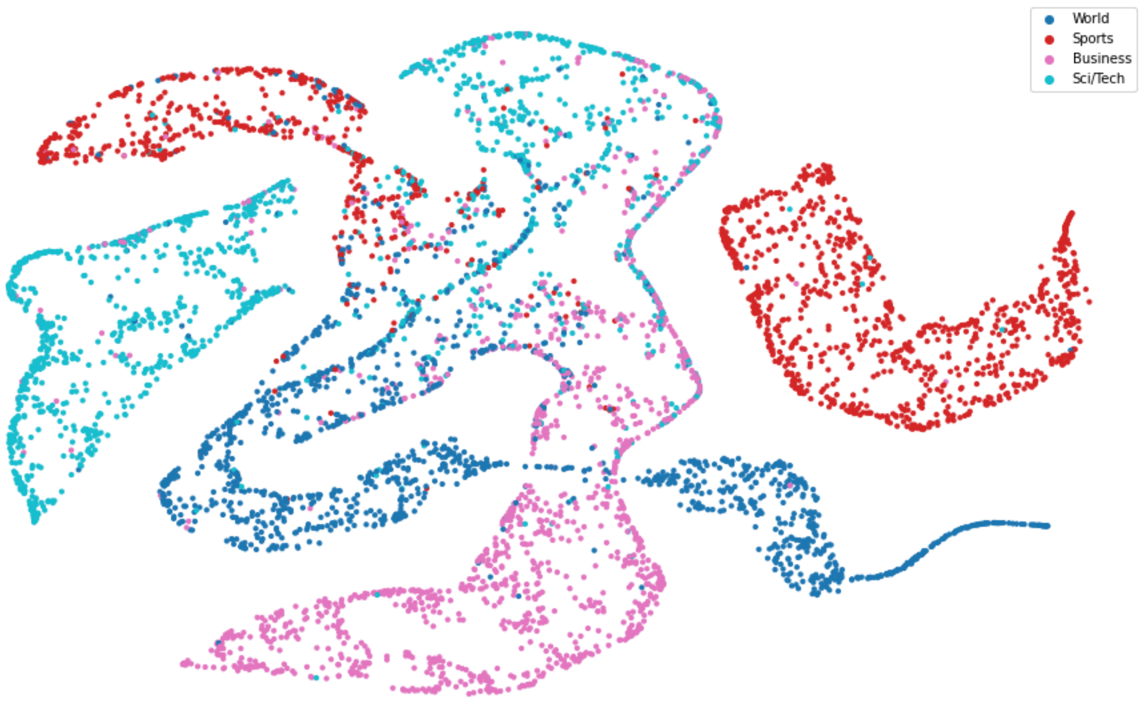
true label \ predicted label	0	1	2	3
0	1608	114	99	79
1	40	1788	20	52
2	92	45	1516	247
3	97	63	159	1581

true label \ predicted label	0	1	2	3
0	1693	70	74	63
1	54	1773	20	53
2	104	27	1530	239
3	104	47	155	1594

true label \ predicted label	0	1	2	3
0	1653	84	87	76
1	66	1776	21	37
2	85	38	1529	248
3	96	64	167	1573

Figure 11. 4x4 confusion matrix of (top left) Experiment B4; (top right) Experiment C4; (bottom left) Experiment D


World
Sports
Business
Sci/Tech


World
Sports
Business
Sci/Tech

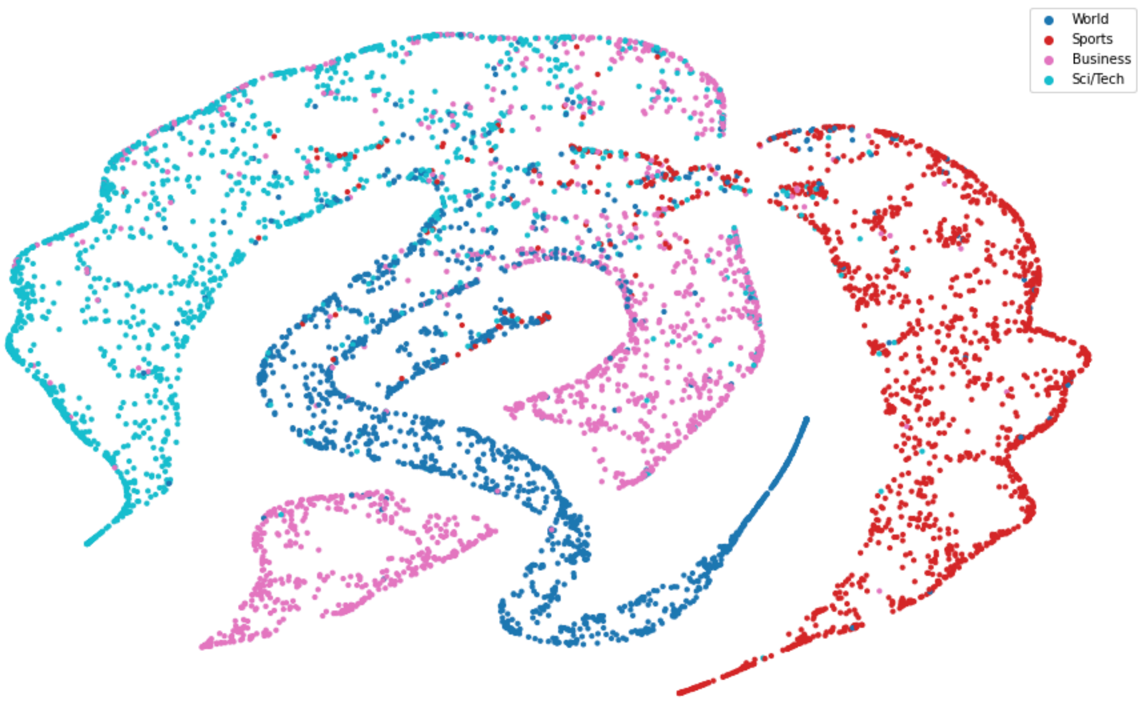


Figure 11. t-SNE graphs of (top) Experiment B4; (middle) Experiment C4; (bottom) Experiment D

Max epochs	Batch size	Max token size	Max vector length
10	100	1000	150

Table 1. Common model parameters across experiments

	Model	Regularization	Dropout Rate	Epochs	Trainable parameters	Time to train	Training accuracy	Training loss	Validation accuracy	Validation loss	Test accuracy	Test loss
Exp B1	Simple RNN (one-hot encoding)	No	-	10	33,188	855.746 s	52.3%	1.128	53.5%	1.152	63.9%	0.937
Exp B2	Bi-directional Simple RNN (one-hot encoding)	No	-	6	66,372	909.792 s	86.7%	0.374	85.2%	0.425	85.6%	0.413
Exp B3	Bi-directional Simple RNN (word embedding)	No	-	10	274,756	1974.726 s	89.4%	0.309	84.9%	0.457	85.2%	0.420
Exp B4	Bi-directional Simple RNN (word embedding)	Yes	0.1	7	274,756	1365.193 s	87.8%	0.349	84.8%	0.433	85.4%	0.404
Exp B5	Stacked Bi-directional Simple RNN (word embedding)	Yes	0.1	6	291,524	2326.033 s	88.3%	0.335	84.8%	0.443	85.7%	0.414
Exp C1	LSTM (one-hot encoding)	No	-	4	99,396	55.043 s	25.1%	1.386	25.2%	1.387	25.0%	1.387
Exp C2	Bi-directional LSTM (one-hot encoding)	No	-	10	264,708	219.181 s	88.3%	0.325	86.4%	0.381	86.4%	0.386
Exp C3	Bi-directional LSTM (word embedding)	No	-	8	311,940	171.144 s	88.6%	0.317	86.3%	0.381	86.7%	0.381
Exp C4	Bi-directional LSTM (word embedding)	Yes	0.1	9	330,244	189.154 s	89.0%	0.308	86.4%	0.373	86.7%	0.377
Exp C5	Stacked Bi-directional LSTM (word embedding)	Yes	0.1	10	396,548	384.326 s	89.8%	0.285	86.4%	0.389	86.7%	0.381
Exp D	Stacked 1D CNN (word embedding)	No	-	5	302,276	35.021 s	89.4%	0.301	85.9%	0.401	85.9%	0.396

Table 2. Comparison of models in Experiment B1-D

References

Chollet, F. (2021, December 21). *Deep Learning with Python, Second Edition* (2nd ed.). Manning.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Laurens van der Maaten, & Geoffrey E. Hinton. (2007, December 31). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605. http://isplab.tudelft.nl/sites/default/files/vandermaaten08a.pdf

BengioYoshua, DucharmeRéjean, VincentPascal, & JanvinChristian. (2003). A neural probabilistic language model. Journal of Machine Learning Research. https://doi.org/10.5555/944919.944966

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2), 157–166. https://doi.org/10.1109/72.279181

Ilya Sutskever, Oriol Vinyals, & Quoc V. Le. (2014). Sequence to Sequence Learning with Neural Networks. Neural Information Processing Systems, 27, 3104–3112. http://cs224d.stanford.edu/papers/seq2seq.pdf

Maali Mnasri. (2019). Recent advances in conversational NLP : Towards the standardization of Chatbot building. ArXiv: Computation and Language.