Assignment 3: Language Modeling with RNN

Vivian Xia

Northwestern University

MSDS458: Artificial Intelligence & Deep Learning

Syamala Srinivasan

February 21, 2022

Abstract

Text has an order associated with it that provides the context for the next word. The sequence of these words are important patterns to consider and remember when predicting the next word. The context allows for the understanding of the sequence of words. So when building a model that uses language as its input, the model should capture those patterns and memorize them to be used as context for its predictions. 1-D CNN and RNN model architectures allow the temporal correlations to be extracted as well as consider the time associated with each of the features. 1-D CNN uses a convolutional and max pooling layer to extract features in the dimension of time. RNN uses two inputs, one being the current word and the other being the hidden cell state or temporal correlations, so the memory of patterns is being considered for each time step. LSTM is a type of RNN, so it also has a memory that is used in every time step, but it separates the memory into short and long term memory to only remember relevant information and forget the irrelevant ones.

Multiple models will be built in order to classify a subset of the articles from AG news into four topics of World, Sports, Business, and Sci/Tech. Before building the models, the text is explored including experiments on different vocabulary sizes and the resulting number of known and unknown tokens in each document. The built models and their performance will be compared to evaluate which model should be implemented to best classify the articles.

Introduction

Text is a type of sequence data where the order of the words holds context and patterns that are relevant to the later words. Because the order of the data matters, the model must also take that order into consideration. Deep learning models such as recurrent neural networks and one-dimensional neural networks have an architecture that are able to capture the correlations of

the sequences in the text. A type of recurrent neural networks is long short term memory models that, like traditional recurrent neural networks, is able to save the memory of the correlations. Long short term memory, however, separates memory into short and long term memory, which elevates the issue that simple recurrent neural networks loses long term memory.

The goal is to build these three different model architectures to classify a subset of AG news articles into its corresponding topic labels. Not only does this problem require a model to be built but also to process the text into vectors to provide the text with meaningful representation as well as build a vocabulary with it. These experiments will explore different sized vocabularies to see the implications of using a smaller versus larger vocabulary for this dataset. The construction of different models will allow for the comparison of model architectures as well as observe the performance of the different models via its accuracy and processing time. Each model's performance will supplement the understanding of the concepts behind each type of model. By analyzing the performance of each model, a final model can then be recommended for this multi-classification problem.

To experiment with the different models using sequential data, a subset of the AG news dataset will be used. The AG news articles are sourced from ComeToMyHead, an academic news search engine. The overall ag_new dataset contains more than one million news articles (Gulli, n.d.). For this application, a subset of 127,600 news articles will be used. The data is evenly distributed among 4 topic classes of World, Sports, Business, and Sci/Tech. The goal of the model is to use the input to predict and classify the text into one of the four topics.

Literature Review

For natural language processing applications, the use of recurrent neural networks, RNN, is commonly used due to its model architecture that is specifically designed for sequential

information like text. Unlike traditional deep neural networks, RNN introduces the concept of memory, allowing past information to be stored and used as additional factors into predicting the output. By memorizing the "time-related contextual information," the network can take into account the sequence of the data (Majid & Santoso, 2021). The RNN enables loops in each iteration and links between hidden components, allowing the temporal correlations to be captured between data (Giri et al., 2021).

Another type of neural network, convolutional neural networks or CNN are also a common model to be used in sequential data applications. Regardless of an RNN or CNN model, the input text needs to be vectorized to represent the meaning of the words (Majid & Santoso, 2021). Word embeddings are the "heart of the project" and often used for more meaningful representation of the text (Giri et al., 2021).

After preprocessing the text, the model is built. In one study that aimed to detect offense in memes using NLP and deep learning, CNN and LSTM were selected to model the sequential data. LSTM or long short term memory is a type of RNN that also stores memory of past contextual information but divides that memory into short and long term memory. The LSTM was able to use the input conversation text to extract patterns that categorized text based on its sentiment (Giri et al., 2021).

As mentioned above, CNN is another type of model that is able to extract patterns or features from the sequential data. CNN was originally invented for computer vision but has also been effective in NLP applications. Similar to implementing it in computer vision, CNN uses convolutional and max pooling layers to extract features to create a feature map and take the most important feature from each map. A study done on multiple experiments with CNN built on top of word embeddings shows that a simple CNN with one layer of convolution performs well

in multiple classification applications. The study's conclusions emphasize the importance of word vectors in a model's ability to capture patterns to predict the outputs (Kim, 2014).

Methods

In order to classify the classes based off text inputs, the libraries packaging, numpy, pandas, collections, time, matplotlib, seaborn, sklearn, and tensorflow were used to build recurrent neural networks. The library packaging was imported in to use the version package to check and verify the tensorflow and keras version that the notebook requires. The libraries numpy and pandas were used to format the data. The collections library was used in exploratory data analysis such as checking if the data is evenly distributed among the classes. The time library is used to measure how long processing time for each model to complete training. Matplotlib, seaborn, sklearn were used to visualize the performance of the model. Sklearn was also used to measure the accuracy of the model. The source of the dataset was from tensorflow_datasets library. The library tensorflow was used to build and train the models.

The ag_news_subset dataset is downloaded into the notebook from tensorflow. The 127,600 articles in the dataset are split into a training set of 120,000 and testing set of 7,600. For the purposes of exploratory data analysis, the training and test data are concatenated into one dataset to explore the data in its entirety. Because this is a curated dataset from tensorflow, the metadata can be retrieved to provide information on the dataset including its source, features, and description. The first ten documents of the dataset are outputted to shows examples of the text and its corresponding labels, which are provided in both numeric and text formats. The topic labels do seem to fit each of its corresponding descriptions.

The labels are further reviewed by checking the metadata for the number of classes and the class names in the dataset as well as its corresponding numeric class label. There are 4

classes made up of “World,” “Sports,” “Business,” and “Sci/Tech” that are in the dataset. The numeric labels of 0, 1, 2, 3 represent “World,” “Sports,” “Business,” and “Sci/Tech” respectively. Among these four classes, the total 127,600 articles are evenly distributed at 31,900 articles per class.

For the computer to understand the text, the text must be encoded numerically as vectors. To convert into vectors, the TextVectorization layer and the adapt function are used. The TextVectorization layer standardizes the text by changing all the text into lowercase, removing punctuation, splitting the text into substrings then recombining them into tokens to build a vocabulary. This layer also has parameters that will be experimented with in later experiments including the maximum number of tokens in the vocabulary (Chollet, 2021). This layer’s default output_mode is integer indices, which will encode the tokens using an integer index (“tf.keras.layers.TextVectorization,” n.d.). The adapt function is used to index the vocabulary by giving each token a unique integer value, transforming each token using the index into a vector. During indexing, all sequences are adjusted to be the same length as the longest sequence. The shorter sequences are padded using zeroes to match the length of the longest sequence so that the data batches are contiguous. Padding, however, is not helpful in training the model and can cause the information stored in its internal state to fade out as it iterates through the meaningless inputs (Chollet, 2021). The encoder uses 0 for padding, 1 for unknown words, 2 for the common words, and other indexes are used to denote known vocabulary words.

Because natural language processing requires extensive exploratory data analysis and preprocessing of the text, Experiment A will test different vocabulary sizes and edit the output sequence length. For each experiment, the vocabulary will be built of the specified number of tokens, and the tokens will be sorted by frequency within the vocabulary. For each experiment,

the distribution of tokens per document will be analyzed as well as visualized in a histogram. If the vocabulary size is too small, there will not be enough words for the model to train on as well as miss important words. If the size is too large, there could be too much padding in the sequences and will not be helpful to the model **(**Srinivasan, 2022). The output sequence length parameter pads or truncates the sequences to a specific length. The default of output sequence length is None. One of the experiments will observe the effect of fixing a value to that parameter.

After exploratory data analysis, the data is preprocessed to begin modeling. The data is, again, loaded in but split into training, validation, and test data sets with 114000, 6000, and 7600 articles respectively. For the training set, the data is distributed about evenly among the four classes. A buffer size of 10000 is used to shuffle each of the datasets to help with overfitting. A random text and label pair from the first 10000 elements in the buffer will be replaced by a new element, maintaining the buffer size. A batch size of 64 is also set to create batches for each of the datasets.

The text is encoded using the vocabulary size of 1000. The most frequent and least frequent twenty tokens are observed. The twenty most frequent have many stop words including “the,” “a,” “to,” “of,” etc. The least frequent have more meaningful words such as “singapore,” “crisis,” “build,” “lawsuit,” etc. The first three example texts from the training dataset and its corresponding encoded indices are observed.

The encoded data of 1000-dimenstional vectors is used as inputs to model the topic classification of each document in the corpus. The first model that will be designed is a recurrent neural network, RNN. This network is designed to capture temporal and sequential correlations. Text is a sequence of words where one word depends on the past words. RNN can keep a memory of those past words, which are used as an input for the current step. The RNN

architecture is made up of multiple RNN cells. Each cell contains two inputs with a hidden layer and two outputs. In each layer, there are any number of neurons where each neuron captures a pattern or temporal correlation. The two inputs are the current word and the hidden state of memory of the correlations up until the last word. The two outputs are the predicted cell state and the hidden state with the memory with this current cell state. The memory is what gives the context for the cell state, so the memory is also used as an input for every step. The weight vectors from the input and hidden layer, hidden to output layer, and the hidden units for timestamps are the same since the data is all related, so the weights can be reused. To get the activated values in the hidden layer, the activation function is applied to the sum of two affine transformations, one being the weight matrix for the hidden layer and the current word input vector, and the other the weight matrix of the hidden units for timestamps and the patterns from the past words. This allows the memory and the current input to be brought together to get the activated values. Those values are then connected through a dense layer to connect the nodes to the preceding nodes. Another dense layer is then used to classify the outputs. Loss is calculated and back propagated through each time step, tweaking the weights. Because the same weights are used throughout time, the vanishing or exploding gradient is an issue in RNN. RNN is also unable to keep long term memory, so temporal correlations that deal with longer lags may be lost during training (Srinivasan, 2022).

The first model that will be built is a simple RNN. It is built using a Sequential class. The first layer is the encoder that was mentioned earlier that converts the text into sequences of token indices. The second layer is the embedding layer that uses a shallow network to generate embedding vectors. The resulting dense and low-dimensional embedding vectors use context to give semantic meaning to each word so that the similar vectors will have similar meanings. The

indices are used as a lookup for each word embedding. This layer takes the parameters of the size of the vocabulary as the input dimensions, the size of the embedded vector space as the output dimension. The embedding layer also has a parameter to generate a mask that skips the iterations where there is padding. The next layer is the simple RNN layer that takes 64 neurons. A dense layer is used to connect the nodes from the bidirectional layer to the nodes in this layer. Another dense layer with a SoftMax activation function is used to return a value between 0 and 1 to represent the probability of the text's classification in each of the four topics. The model is then compiled using an Adam optimizer and Sparse Categorical Cross Entropy loss function. The metric is specified as accuracy. The model is trained using the training set with a fixed number of epochs of 150. The regularization technique of early stopping is used to prevent overfitting. The model will stop training after the validation accuracy does not improve after two epochs. The performance will be analyzed as well as visualized in the form of line graphs, confusion matrices, and prediction probabilities gradient. The line graphs will plot the performance metrics of loss and accuracy of the training and validation. The confusion matrix will show the classification rates of the classes based on the predicted and true labels. The prediction probabilities gradient for fifteen of the test set articles will show how confident the model was in predicting each article's class. The less distribution of color there is in the gradient, the better the temporal correlations discriminate the classes.

Another type of RNN model is a bidirectional RNN. A bidirectional RNN runs two recurrent layers on the input, one time reading the input forward from left to right and another time reading the input backward from right to left. The outputs are concatenated at each time step. By processing the sequences forwards and backwards, the RNN can catch patterns that would not have been found if the model only read the input in one order (Chollet, 2021). It has

the same architecture of the simple RNN except the bidirectional layer wrapper with simple RNN layer. The bidirectional layer can be implemented with simple RNN or LSTM.

As mentioned above, simple RNN has many issues including unstable gradient and an inability to store long term memory. Long Short Term Memory or LSTM networks address those issues. LSTM not only has the input and output gate that traditional RNN has but also a forget gate. This model can identify key words and forget irrelevant words. It addresses which information to forget from the previous cell state using the forget gate, to save to this cell state or should be forgotten using the input gate, and to save to go to the next cell state using the output gate. These gates divide memory into long and short term memory. The gates use sigmoid or tanh as activation function to allow for binary decision-making of whether to forget or keep the information. This architecture allows for the relevant memory of past words and the current word information to be captured as a hidden state and cell state (Srinivasan, 2022).

Another model architecture that can be used with sequential data is 1-dimensional convolutional neural networks, 1-D CNN. CNN is designed to capture spatial correlations. However, it can also extract features from sequential data, which is 1-dimensional, using auto or serial correlation (Srinivasan, 2022). The convolution layers slide a kernel window to extract features or correlations from the embeddings. The kernel scans a list of word embedding in sequence to look at each word embedding as well as the surrounding words to output a value that captures the correlation or pattern about that sequential word grouping. The kernel moves in one dimension, time, to get a feature vector, which will then have max pooling performed to extract the important features over time (Camacho, 2019).

Results

Experiment A: EDA

Experiment A.1 – Vocabulary Size: All

The TextVectorization layer did not limit the vocabulary, so all the tokens in the text are added to the vocabulary. The entire vocabulary set contains 95,976 tokens. The twenty most frequent tokens are observed. These 20 tokens include padding and "[UNK]," unknown words, as the two most frequent tokens. The other frequent tokens are stop words such as "the," "a," as," and "is" that hold little to no information and is not important in this classification application.

There are 3.9 million words present in the corpus. Each news article has between 3 and 173 tokens. The histogram, Figure 3, shows the distribution of tokens in each document. There is a normal distribution, but there is a little right skew as seen by the right tail. Most documents have around 30 tokens each. The right tail shows that there are not many documents with more than 60 tokens each, which would result in input sequences with a lot of padding.

Experiment A.2 – Vocabulary Size: 1000

This experiment will limit the vocabulary to the 1000 most frequent tokens by setting the parameter max_tokens in the TextVectorization layer. The most common twenty tokens are the same as that of the preceding experiment, so there are many unknown words and filler words. This was further explored by using the encoder on example texts from the dataset. In each example, the text is shown and its corresponding label. Each text is encoded with the indices created from the TextVectorization and adapt function. The 1's represent unknown words, so words such as "dual-core," "mainly," "computing," "databases" from the first example are not in the vocabulary. Because the vocabulary does not contain all the tokens, the process is not completely reversible. The original text has the words "computing" and "databases" but, converting it back, the words are unknown because they are not in the vocabulary. The

vocabulary also contains lower case and no punctuation, which also will differ from the original text when reversed back to text from indices.

The percentages of unknown words for each document are observed. There are only 12 documents that have no unknown words. There are 22 documents that have 100% unknown words in their documents. Most documents have around 30-40% unknown words, which can also be seen from the histogram in Figure 4, showing the percent of non-vocabulary words per document. Limiting the vocabulary size to 1000 tokens resulted in 2.6 million known words that are present in the corpus, a decrease from the 3.9 million using the entire vocabulary.

Experiment A.3 – Vocabulary Size: 500

As seen from Experiment A.1, most documents had only 30 to 40 tokens each using the entire vocabulary, so this experiment will decrease the number of tokens in the vocabulary to 500 tokens. The encoded example texts show more 1's than the preceding experiment did, so there are more unknown words in each text. There are not many important known words in the texts, rather there are many words such as in "is" and "a" that are identified. From Experiment A.1 and A.2, it was observed that the twenty most frequent tokens contained many stop words. Experiment A.2 with a vocabulary size of 1000 may give better prediction results than this experiment's vocabulary size of 500 since the larger vocabulary size of 1000 can offset the lack of information of the stop words.

Despite decreasing the vocabulary size by half compared to the preceding experiment, there are still some low percentages of unknown words in a document for a few documents. However, there are more documents than that of the last experiment's that have 100% of unknown words in their documents. The histogram, Figure 5, shows most documents have 35-55% unknown words in their documents. The decrease in vocabulary size from the preceding

experiment increased the percent of unknown words in each document by around 10%. The number of words present in the corpus using this vocabulary is 2.2 million, which is a decrease from the preceding experiment by 400,000 words.

Experiment A.4 – Output Sequence Length: 40 & Vocabulary Size: 1000

As seen by Experiment A.1, most documents had about 30 tokens each. For this experiment, the output_sequence_length parameter in TextVectorization is set at a fixed value of 40 tokens. Because the tokens are limited to 40, longer sentences than the 40 tokens are truncated to a length of 40 tokens and shorter sentences are padded with zeroes

to match the length of 40 tokens (Chollet, 2021). The vocabulary size is set at 1000 tokens. The example texts show the same amount of 1's or unknown words as Experiment A.2. There are more known words that hold meaning and offset the stop words. The encoded text also shows the effect of fixing the output sequence length to 40. The padding of zeroes in each sequence to fit the length of 40 can be seen. The first two sequences have more padding than the third sequence, but this parameter will prevent the encoding index to pad sequences to the longest sequence in each batch which may go up to 173 tokens, as observed in Experiment A.1.

By setting the output sequence length, the percent of unknown words in each document decreased from Experiment A.2 that did not use the output sequence length but had the same vocabulary size. Most of the documents have 20% unknown words each rather than 30-40% from Experiment A.2. By truncating the longer sequences to 40 tokens, the number of unknown words decreased and there is less of right tail in this histogram, Figure 6. Experiment A.2 had some documents that had 100% unknown words. This experiment does not have any documents

with more than 90% unknown words. The number of words in this corpus decreased from 2.6 million in Experiment A.2 to 2.5 million in this experiment.

Experiment B: RNN

Experiment B.1 – Single Layer Simple RNN

A single layer simple RNN is built using the encoder with a vocabulary size of 1000 tokens, an output dimension of a 64-dimensional embedding vector space, use of masking, one simple RNN layer with 64 nodes that represents 64 cells, one dense layer with a ReLu activation function and 64 nodes, and one dense layer with a Softmax activation function and 4 nodes to represent the 4 classes. Early stopping is used during training to prevent overfitting.

The training and validation accuracy were very similar in value, so there does not seem be overfitting. The test accuracy is 0.85, representing that the model does a good job in classifying the articles to its corresponding topic. The accuracy and loss performance metric plots, Figure 7, also show that there is no overfitting. The training and validation graphs go in the same direction. The validation loss starts to flatten out at the end of training, so the early stopping regularization technique does a good job in stopping training before the validation loss graph starts to go the opposite direction of the training loss graph. The plots also show that the model did a good job as the loss plot shows the graphs close to 0 and the accuracy plot shows the graph close to 1. The confusion matrix, Figure 8, also shows the model did a very good job classifying class 1 or Sports correctly. The other classes have a correct classification rate of approximately 0.80, which is still good. The largest error rate is 0.14 where class 2, Business, is misclassified as class 3, Sci/Tech. The prediction probabilities, Figure 9, are also visualized and there is little to none color distribution along each column, so the model did extract features that were able to discriminate among the classes well.

Experiment B.2 – Single Layer Bidirectional Simple RNN

This model used a bidirectional layer wrapper with a simple RNN layer. Otherwise, the model architecture is the same as Experiment B.1. With the bidirectional layer, the parameters increase from Experiment B.1's 76,676 to 89,028. The accuracy of the testing set is only 0.01 greater than the accuracy of Experiment B.1's, so it does not seem like the model captured many new temporal correlations from going backwards that the unidirectional could not capture already from Experiment B.1.The processing time to train this model took a little less than two times as long as that of Experiment B.1. Because of the only slight increase in accuracy, the unidirectional simple RNN would be a better model to implement than this model.

The performance metric plots show that both the loss and accuracy graphs are moving in the right direction. The validation loss is flattening out while the training loss is continuing to go down, so the early stopping was helpful in preventing overfitting. Similarly, the validation accuracy flattens out while the training accuracy continues to improve. The overall accuracy and loss scores are good. The confusion matrix shows that there is a light gray and white diagonal, so the classification rate was also good among each class. Similar to Experiment B.1, the class 1 or Sports has the best classification score and the other classes had about 0.82 accuracy. The prediction probabilities visual does not have much color distribution, so the model did a good job discriminating between classes. Overall, the model did a good job, but it took a very long time to process for only a 0.01 increase in test accuracy.

Experiment C: LSTM

Experiment C.1 – Single Layer LSTM

This model is similar in structure to Experiment B.1 except the simple RNN layer was replaced by a LSTM layer. The model had more parameters to train but finished training six time

as fast as Experiment B.1's did. The accuracy was also the same as that of Experiment B.1's accuracy with a test accuracy of 0.85, which is good. The performance metrics, Figure 10, show that there is no sign of overfitting. The training and validation graphs are very similar to each other. The confusion matrix also looks the same as the last two models' with a white and light gray diagonal and high classification rates for the classes. Similarly, the predictions visual shows that the model is able to discriminate well among the four topics.

Experiment C.2 – Single Layer Bidirectional LSTM

Like Experiment C.1, this model will use a single layer LSTM but with a bidirectional layer wrapper. The architecture is otherwise the same as the preceding model. The accuracy scores for all three data sets were the same as the preceding model's, showing that looking at the sequence backwards did not find any correlations that the model could not capture by just going forwards. The processing time took longer with less epochs than that of the preceding model, but the accuracy scores were the same. Therefore, the unidirectional LSTM is better for implementation than the bidirectional LSTM.

The performance plots show that there is no overfitting in the model since training and validation graphs are similar to each other and go in the same direction. The confusion matrix has good classification rates for class 0 and 1 or World and Sports, respectively. The rates for class 2 and 3 or Business and Sci/Tech are also good at a classification rate of 0.79, respectively. The prediction probabilities visual shows that the model does a good job discriminating between the four classes because there is not much of a color distribution in each column.

Experiment C.3 – Multi-Layer Bidirectional LSTM

This model uses three bidirectional LSTM layers as well as a dropout layer after the first Dense layer to drop 0.50 of the nodes at random for each iteration to prevent overfitting. The

first bidirectional LSTM layer uses 128 cells and returns the full sequence of successive outputs for each timestep to be passed to the next bidirectional LSTM layer of 64 cells. Similarly, the layer returns the full sequence to the next LSTM layer which has 32 cells. Otherwise, the model architecture is the same with the same encoder, embedding layer, and two dense layers. Despite the increase in model complexity, the accuracy for all three datasets were the same as that of the last two preceding models. The test accuracy was also 0.85, so there was no increase in accuracy compared to using a single layer, whether that be with or without the bidirectional layer. Therefore, the model from Experiment C.1 is a better implementation model than Experiment C.2 or this model.

The performance plots show that there is no overfitting as the training and validation graphs are similar to each other as well as go in the same direction. The confusion matrix shows that the classification of each class is good with classification rates around 0.85. The prediction probabilities visual supports the model accuracy as well.

Experiment C.4 – Multi-Layer Bidirectional LSTM with increased model complexity

This model uses the same architecture as Experiment C.3 but with increased model complexity. The number of nodes in the bidirectional LSTM and first dense layers are increased. The first bidirectional LSTM layer will have 256 nodes, the second will have 128 nodes, the third will have 64 nodes. The first dense layer will have 82 nodes. The test accuracy improves by 0.01 compared to the preceding model's test accuracy. However, the processing time took longer as well as many more parameters needed to be trained. The preceding model with less nodes would be better for implementation since the accuracy did not improve much. From the performance metric plots, the training and validation graphs are similar in value, so there is no

overfitting. The confusion matrix and prediction probabilities visual shows that the classification rates for each class are good and the model is able to discriminate between the two classes.

Experiment C.5 – Single Layer LSTM with vocabulary size of 2000

This model uses a vocabulary size of 2000 instead of 1000. Th most common 2000 tokens were encoded and used in the model. The 20 least frequent words in the vocabulary seem do not seem like they are too specific to one text with words such as “wins,” “chinas,” “defeated,” “seasons,” so these words do seem like they can help in identifying the topic. The model architecture stayed the same as that of Experiment C.1’s with one layer of unidirectional LSTM using 64 nodes and two dense layers. The only difference in the architecture is that a dropout layer was added in between the dense layers to prevent overfitting.

The test accuracy is 0.88, which is better than the other models experimented so far. It also was had a faster training time than the other experiments. The performance plots show that the training and validation graphs are similar to each other. The validation graphs start to flatten out towards the end of training, so the early stopping stopped training and prevented overfitting. The confusion matrix and prediction probabilities show that the model was able to extract meaningful correlations that did a good job classifying each text in its corresponding class. The increase in vocabulary size allowed the model to increase in accuracy without having to add extra layers and nodes.

Experiment D: 1-D CNN

Experiment D.1 – 1-D CNN

The model architecture is similar to the other experiments with the use of a Sequential class and an encoder then embedding layer. Instead of an RNN layer, the next layer is a one-dimensional convolutional layer with 64 filters, kernel size which is the length of the convolution

window of 6, and ReLu activation function. The next layer is a dense layer with 32 nodes and activation function ReLu. The dropout layer drops 0.50 of the nodes at random for each iteration. The one-dimensional global max pooling layer is used to extract the important correlations found in the text. The last dense layer has 4 nodes for the 4 classes the model is discriminating.

The 1-D CNN test accuracy does just as well as the other models with the 1000 token vocabulary size at 0.85. Despite having more parameters than the simple RNN models, this experiment's processing time was much faster than Experiment B's models. It was also faster than the other models as well, but it did have less parameters to train. The performance metric plots show that there is a gap between the training and validation loss graphs. The gap seems to be more than 0.11, so there does seem to be some overfitting. On the other hand, the training and validation accuracy graph are similar to each other. There may be overfitting so using more regularization techniques may help. The confusion matrix supports that the model does a good job classifying the classes correctly. The prediction visual shows some distribution of hues for the first 15 samples from the test set, so the model does not do a very good job discriminating between the topics, despite being able to classify it correctly.

Summary: Recommended Model

Based on the test accuracy score, the best model is Experiment C.5's model consisting of a single layer unidirectional LSTM with a vocabulary size of 2000 tokens. The test accuracy is 0.88 which is very good, and based off the visualizations, this model does not overfit and does a good job discriminating between the four topic classes. Not only is the test accuracy score better than the other models, but it also had the fastest processing time other than for the 1-D CNN model. The LSTM is more efficient than the Simple RNN architecture, despite having more parameters to train. The increase in vocabulary size increased the test accuracy score by 0.03

compared to the scores of the other models that used a vocabulary size of 1000 which, for the most part, had a score of 0.85. The bidirectional models did not seem to extract any patterns that the unidirectional models could not since the test accuracy stayed the same between Experiments B.1. and B.2 as well as C.1 and C.2, so there does not seem to be a need to add the extra parameters for this particular case. And with added model complexity by adding more layers, the models do not seem to improve, so more than a single layer does not seem to be recommended. Overall, the single layer unidirectional LSTM with a vocabulary size of 2000 tokens is the recommended model for implementation. It may also be a good idea to do more experimentation using a larger vocabulary size to see if the model can improve even more.

Conclusion

In order to model sequential data, the sequence patterns or correlations need to be memorized and captured. RNN and 1-D CNN model architectures are designed to capture such correlations. RNN, in particular, use memory as an additional input to consider the past context when predicting the current cell state. LSTM is a type of RNN that is able to address the issues with simple RNN such as vanishing/exploding gradient and lost long term memory. All three types of models were implemented using the text to compare the architectures and also its ability to classify the text into four classes. The exploratory data analysis enforced the importance of natural language processing when dealing with sequential data such as text. The number of tokens, as seen by the experiments, does impact the accuracy of the model. The EDA prepared the text to be encoded and vectorized into a word embedding due to the first two layers of the models. These two layers were uniformly used as the first steps in building each of the models, regardless the type.

The simple RNN model did a good job in classifying the text. The LSTM, however, did just as good of a job but much faster. The bidirectional wrapper did not have much of an impact in accuracy for this application with this dataset. There were also multi-layer LSTM models with three bidirectional LSTM layers that were experimented, but they failed to improve the accuracy. The hyperparameter that did make the most difference was increasing the vocabulary size, showing that exploring and experimenting with the text itself is also an important part of the process in language modeling. 1-D CNN also did a good job in classifying the text into its respective topics. It also took less time than the other models, so this model architecture is also a good alternative to RNN modeling with sequential data. Overall, simple RNN, LSTM, and 1-D CNN models were all able to successfully classify the articles into its respective topic class. However, the single layer LSTM with vocabulary size of 2000 did the best and should be implemented.

References

Camacho, C. (2019). *CNNs for Text Classification*. Cezanne Camacho. Retrieved 2022, from https://cezannec.github.io/CNN_Text_Classification/#:~:text=1D%20Convolutions&text=To%20process%20an%20entire%20sequence,in%20only%20one%20dimension%3A%20time.

Chollet François. (2021). *Deep learning with python*. *O'Reilly*. Manning Publications. Retrieved 2022.

Giri, R. K., Gupta, S. C., & Gupta, U. K. (2021). An approach to detect offence in Memes using Natural Language Processing (NLP) and Deep learning. *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 1–5. https://doi.org/10.1109/iccci50826.2021.9402406

Gulli, A. (n.d.). *AG's corpus of news articles*. Retrieved 2022, from http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

Kim, Y. (2014). (rep.). *Convolutional Neural Networks for Sentence Classification*. arXiv. Retrieved 2022, from https://arxiv.org/abs/1408.5882.

Majid, R., & Santoso, H. A. (2021). Conversations sentiment and intent categorization using context RNN for emotion recognition. *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 46–50. https://doi.org/10.1109/icaccs51430.2021.9441740

tf.keras.layers.TextVectorization. TensorFlow. (n.d.). Retrieved 2022, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization

Srinivasan, S. (2022). *MSDS 458 AI/Deep Learning: Sync Session #4* [Zoom Cloud Recordings].

Appendix

Figure 1. Experiment A: EDA descriptions and processing times.

Experiment A: EDA	Description	Encoding Process Time
1	Vocabulary size: All	2min 59s
2	Vocabulary size: 1000	2min 49s
3	Vocabulary size: 500	2mins 51s
4	Output sequence length:40 & Vocabulary size: 1000	3mins 00s

Figure 2. Experiment model results and times.

Experiment	Description	Training	Validation	Testing	Process Time (seconds)	Epochs	Total Parameters	Hyperparameters
B.1	Single Layer Simple RNN	0.87	0.86	0.85	2338.52	12	76,676	Vocabulary size = 1000 Simple RNN = 64 First Dense = 64
B.2	Single Layer Bidirectional Simple RNN	0.88	0.86	0.86	4146.00	11	89,028	Vocabulary size = 1000 Bidirectional Simple RNN = 64 First Dense = 64
C.1	Single Layer LSTM	0.87	0.86	0.85	377.49	17	101444	Vocabulary size = 1000 LSTSM = 64 First Dense = 64
C.2	Single Layer Bidirectional LSTM	0.87	0.86	0.85	433.98	13	138,564	Vocabulary size = 1000 Bidirectional LSTM = 64 First Dense = 64
C.3	Multi-Layer Bidirectional LSTM (3 layers)	0.87	0.86	0.85	689.63	11	471,620	Vocabulary size = 1000 Bidirectional LSTM = 128 Bidirectional LSTM = 64 Bidirectional LSTM = 32 First Dense = 64 Dropout = 0.5
C.4	Multi-Layer Bidirectional LSTM (3 layers) with increased model complexity	0.87	0.86	0.86	872.13	12	1,553,054	Vocabulary size = 1000 Bidirectional LSTM = 256 Bidirectional LSTM = 128 Bidirectional LSTM = 64 First Dense = 82 Dropout = 0.5
C.5	Single Layer LSTM with vocabulary size of 2000	0.89	0.88	0.88	204.3	8	165,444	Vocabulary size = 2000 LSTM = 64 First Dense = 64 Dropout = 0.5
C.5	Single Layer LSTM with vocabulary size of 2000	0.89	0.88	0.88	204.3	8	165,444	Vocabulary size = 2000 LSTM = 64 First Dense = 64 Dropout = 0.5
D.1	1-D CNN	0.86	0.85	0.85	97.05	10	90,852	Vocabulary size = 1000 1-D CNN = 64 filters, 6 kernel size First Dense = 32 Dropout = 0.5 GlobalMaxPool1D

Figure 3. Experiment A.1 histogram.

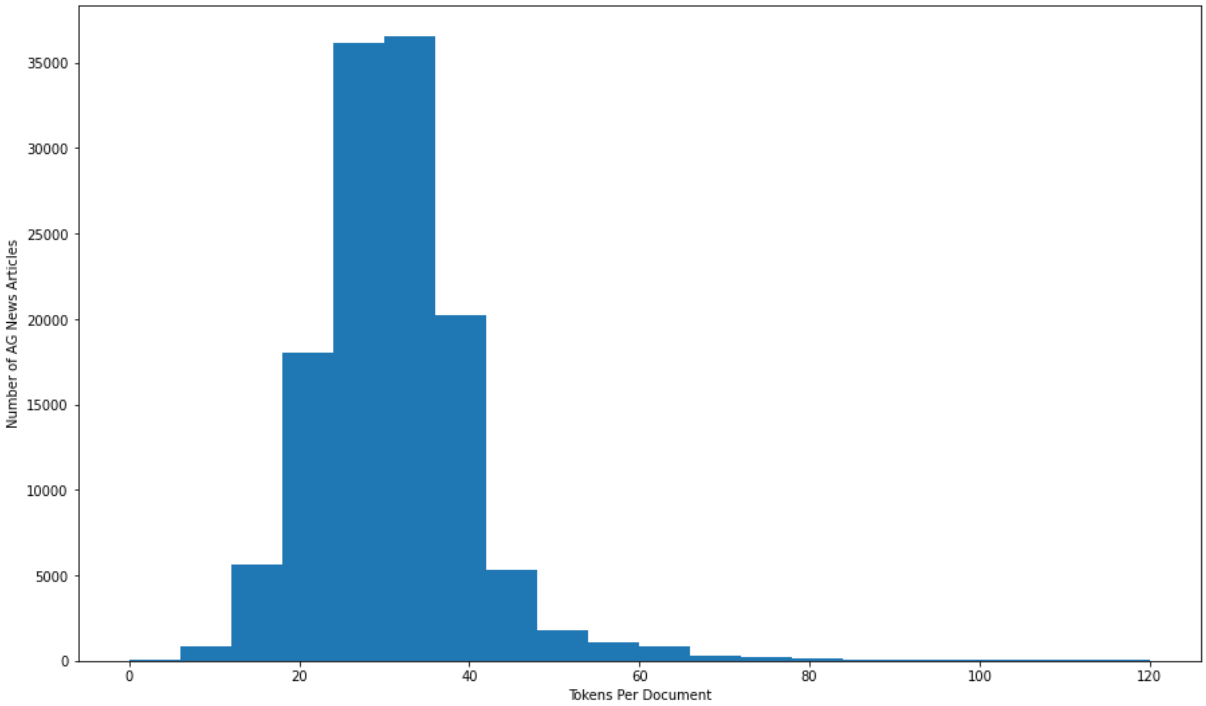

Figure 4. Experiment A.2 percent of non-vocabulary words in a document histogram.

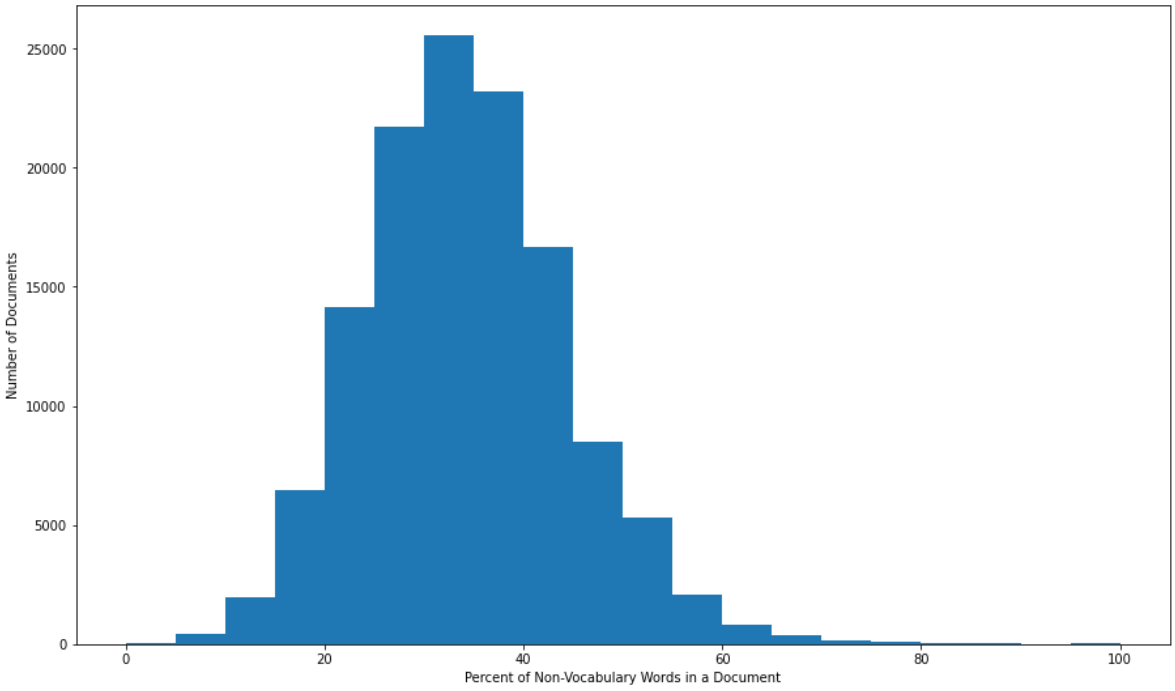

Figure 5. Experiment A.3 percent of non-vocabulary words in a document histogram.

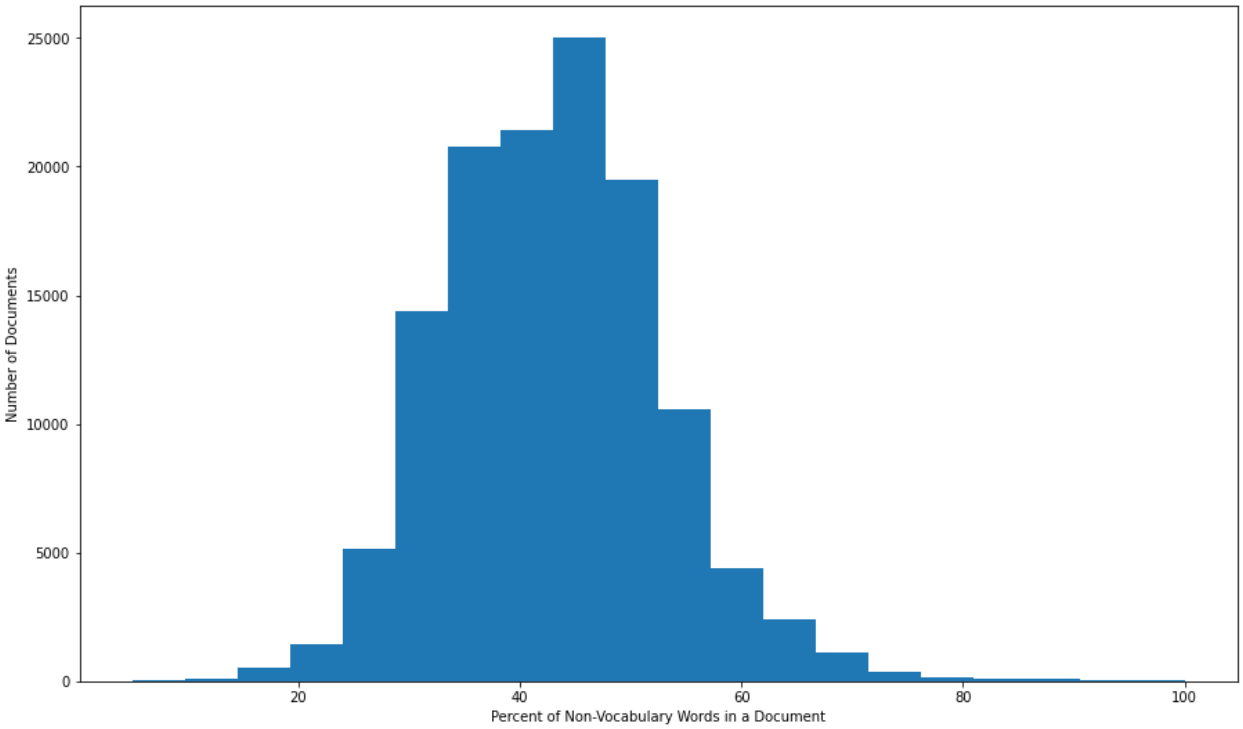

Figure 6. Experiment A.4 percent of non-vocabulary words in a document histogram.

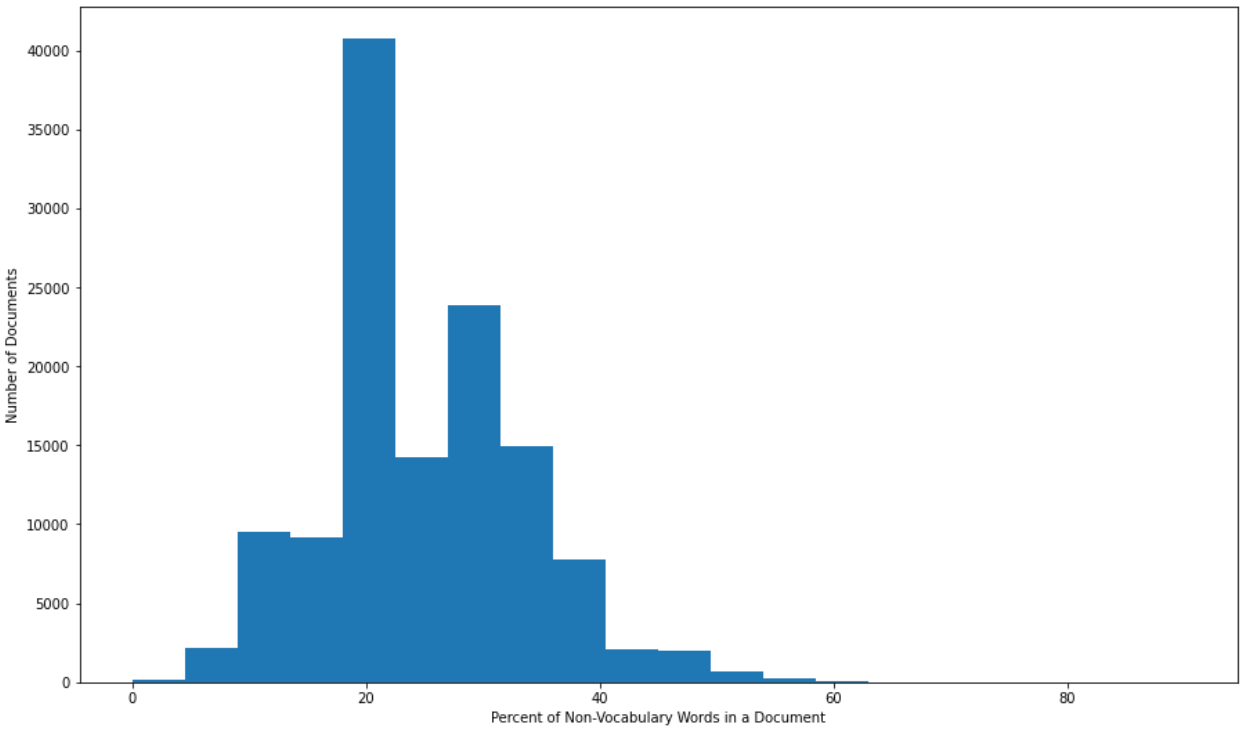

Figure 7. Experiment B.1 performance metrics graph.

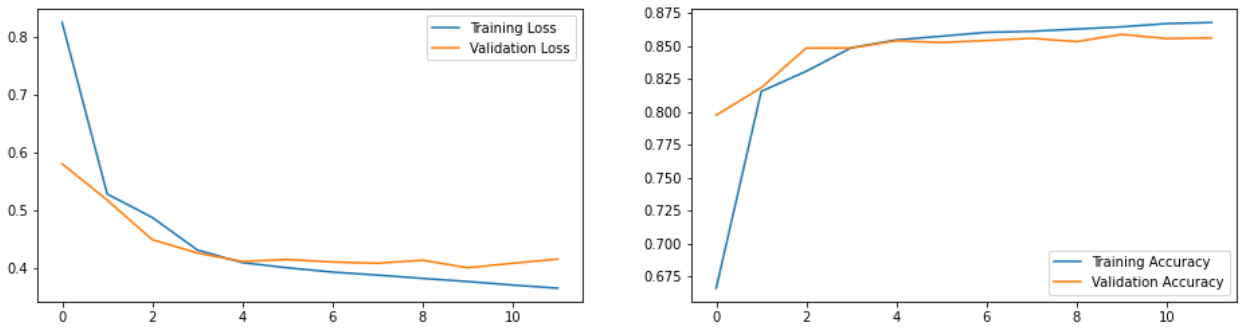

Figure 8. Experiment B.1 confusion matrix.

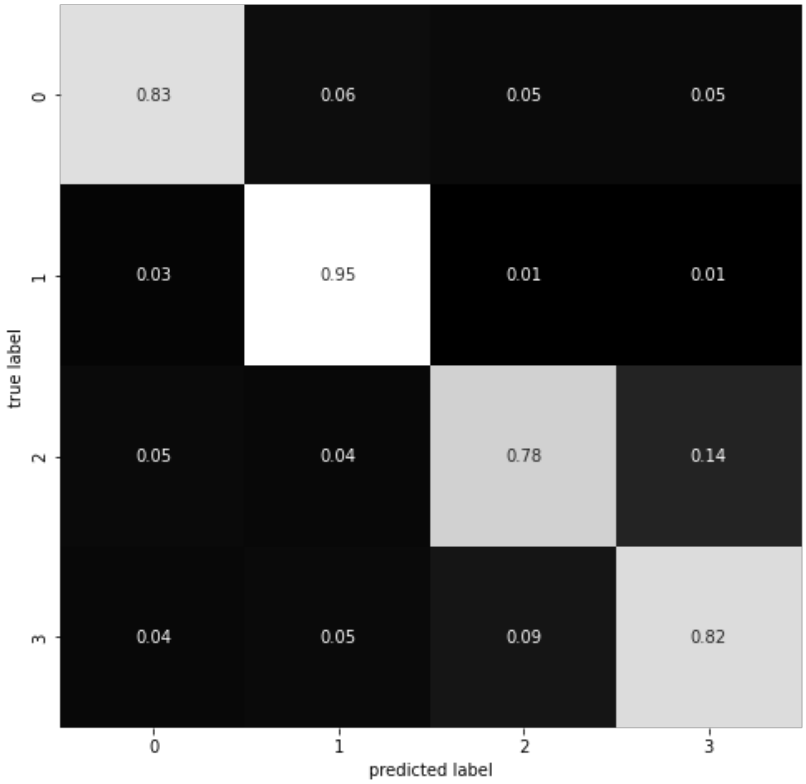

Figure 9. Experiment B.1 prediction probabilities for test set images.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
World	0.14%	99.91%	0.07%	1.02%	99.28%	0.65%	0.58%	15.17%	92.16%	7.47%	99.82%	0.84%	0.46%	84.06%	99.14%
Sports	99.84%	0.00%	0.00%	98.89%	0.02%	0.19%	99.20%	82.58%	0.43%	0.07%	0.00%	0.73%	0.01%	1.12%	0.01%
Business	0.01%	0.06%	1.86%	0.03%	0.16%	11.51%	0.03%	0.10%	0.80%	75.92%	0.08%	1.83%	99.34%	3.32%	0.29%
Sci/Tech	0.01%	0.02%	98.07%	0.06%	0.53%	87.64%	0.19%	2.14%	6.61%	16.54%	0.09%	96.60%	0.19%	11.50%	0.57%

Figure 10. Experiment B.3 performance metrics graph.

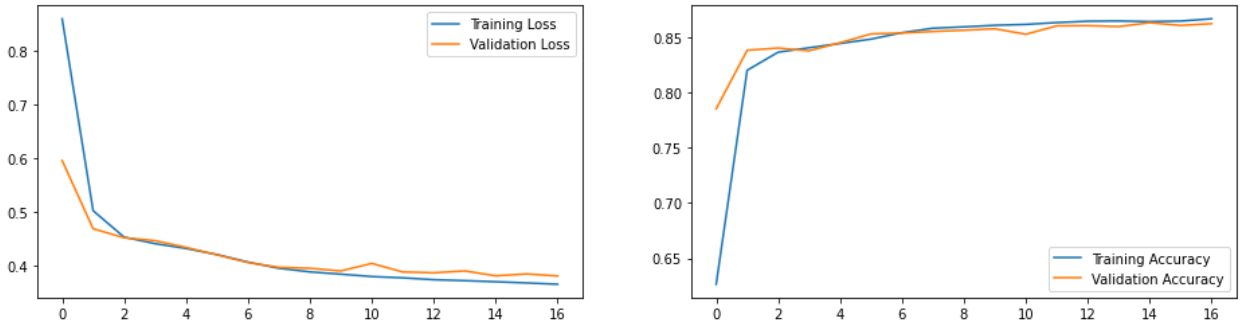

Figure 11. Experiment D.1 performance metrics graph.

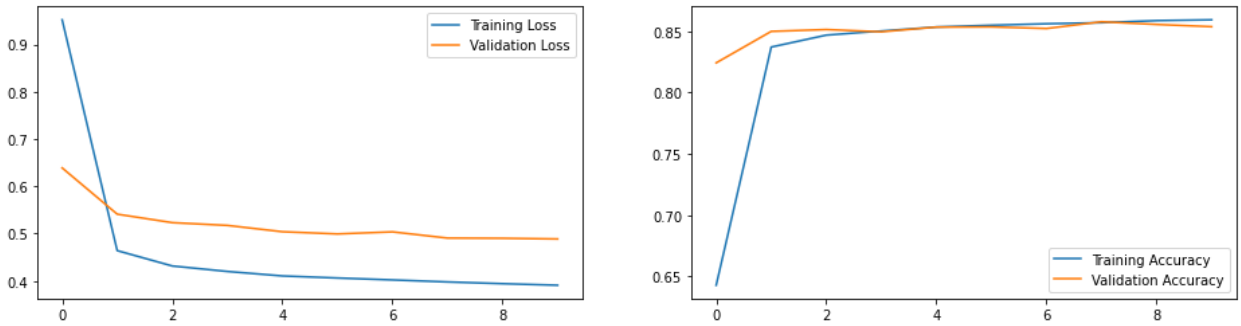

Figure 12. Experiment D.1 prediction probabilities for test set images.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
World	1.64%	95.23%	1.69%	8.34%	92.71%	6.09%	4.51%	34.73%	63.41%	12.09%	98.68%	6.14%	10.67%	76.56%	91.54%
Sports	97.37%	0.38%	0.17%	86.53%	1.25%	1.84%	89.77%	55.09%	3.18%	1.85%	0.09%	8.72%	1.83%	2.84%	0.32%
Business	0.44%	3.21%	9.65%	2.61%	2.58%	32.58%	2.70%	0.97%	11.13%	60.40%	0.51%	5.30%	73.53%	11.97%	5.22%
Sci/Tech	0.56%	1.18%	88.48%	2.53%	3.46%	59.50%	3.02%	9.22%	22.28%	25.66%	0.72%	79.84%	13.97%	8.64%	2.91%