Artificial Neural Networks: A Comparative Analysis for the Purposes of Text Classification

MSc Research Project
Data Analytics

Glenn Connell
Student ID: x14441832

School of Computing
National College of Ireland

Supervisor: Anu Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Glenn Connell
Student ID:	x14441832
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Anu Sahni
Submission Due Date:	13/8/2019
Project Title:	Artificial Neural Networks: A Comparative Analysis for the Purposes of Text Classification
Word Count:	5362
Page Count:	17

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	9th August 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Artificial Neural Networks: A Comparative Analysis for the Purposes of Text Classification

Glenn Connell
x14441832

Abstract

Natural Language Processing (NLP) is an area of great interest within both academia and industry, that has, with the advent of Web 2.0, quietly been gathering an even larger following. One of the key components of many NLP applications, text classification, may be considered one of the largest sub-fields within NLP. Traditional text classifiers tended to be restricted in effectiveness due to their reliance upon human-designed features such as dictionaries. Machine learning has since risen to prominence over traditional methods. Within this paper a comparative analysis has been performed comparing two algorithms, Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN), finding CNN to be the superior approach.

1 Introduction

Natural Language Processing (NLP) is an area subject to much research within both academia and the industry. NLP refers to an area of computer science that deals with the study of how computers deal with human language, the eponymous ”natural” language. This particularly refers to the creation of computer programs capable of processing and performing analysis on large quantities of NLP data. There are several key areas of interest within the field of NLP such as natural language generation, speech recognition and text classification.

Text classification, the focus of this paper, refers to the process of classifying text data for categorization into a more organized format, better suited for use in further tasks. For the purposes of this project, and indeed in general, unstructured text is the target of text classification. This is owed to the abundance of sources for unstructured text such as social media and emails. The potential value of unstructured text is obfuscated by its often indecipherable format, however, with a suitable application of text classification valuable insight may be gleaned from the text data. Text classification has risen to prominence in recent times with the advent of Web 2.0 and the subsequent boom of social media platforms Pang et al. (2008). This alongside the dramatic advancements in computational power and ever-increasing storage options, has enabled organizations to fully leverage unstructured text data, for analysis. Several common examples of text classification use cases include topic detection, sentiment analysis, spam detection and language detection.

The process of text classification is undertaken by a text classifier. A common avenue for implementation of text classifiers is machine learning, particularly supervised machine learning. Text classification, as the data fed into the model contains labels in order to facilitate the training of the model, is strictly considered a supervised machine learning task. While a large variety of machine learning algorithms exist that are suited to the task of text classification, for the purposes of this paper two approaches will be the focus. Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). They are two distinct variations of Artificial Neural Networks (ANN) and will be utilized to construct two different predictive models to facilitate the comparative analysis for the purposes of this paper.

CNNs are an offshoot of ANNs inspired by the biological functions of the body. They emulate the structure of the visual cortex within humans and aim to digest input in the same manner in which the brain processes imagery. CNNs, while related to ANNs, differ in 2 major ways. The defining manner in which they differ is the inclusion of a convolutional layer. This layer is responsible for convolving the input data into a feature map, utilizing a kernel. CNN implementations employ an alternating rotation of convolutional layers and pooling layers. Pooling layers are necessary to reduce dimensionality within the feature map and allows for the reduction of overfitting. Within Figure 1 an example of a CNN architecture is detailed.

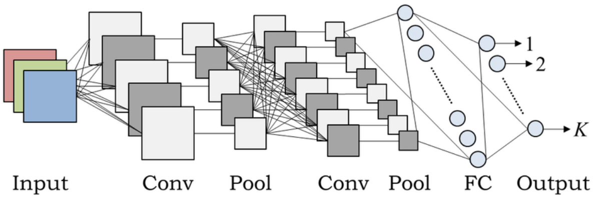

Figure 1: An Example of Convolutional Architecture Hidaka and Kurita (2017)

RNNs diverge from the structure of ANNs by removing the concept of feed-forward from the formula. RNNs, specifically Long Short-Term Memory (LS-TM) Hochreiter and Schmidhuber (1997) implementations, introduce the concept of recall into the architecture. Rather than being restricted to a data flow that can only pass from the input layer through the hidden layers and then out the output layer, RNNs may choose to pass the data back to the hidden layer rather than to pass it forward to the output layer. This can allow for the RNN architecture to overcome the limitations of typical ANNs, as rather than reach an underwhelming result the RNN can repeat transformations to reach a more desirable conclusion. An advancement of the LS-TM architecture is the bidirectional RNN. Rather than only being able to access past data, this architecture can also access future data to reach the desired conclusion. An example of RNN architecture is detailed in Figure 2.

Early research conducted upon both CNN and RNN, that helped define the architectures, initially determined that both approaches were rather limited in scope. However, owing to advances in computation, recent research has revealed the true versatility each architecture. While this research spans a wide variety of topics, even text classification, true comparative analyses between both CNN and RNN methodologies are rare. As such, the findings of this research may be a boon to those uncertain as to which architecture may be the best fit for their selected task.

To combat the lack of comparative analyses in the field of text classification, within this paper a new comparative analysis is conducted. This analysis focuses upon the viability of both the CNN and RNN architectures for the task of text classification, utilizing

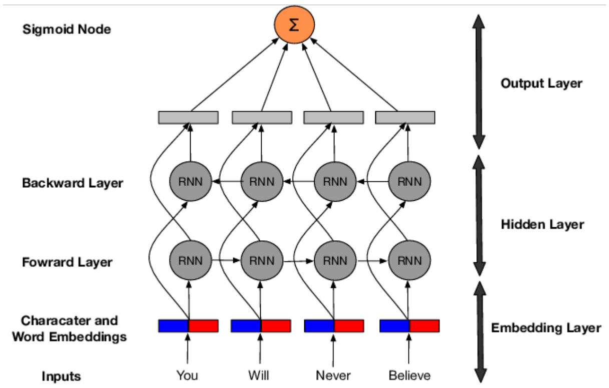

Figure 2: An Example of Recurrent Architecture Anand et al. (2016)

recent techniques in order to provide an environment in which experiment replication may easily take place. Two individual models have been created for the purposes of the analysis. The first created model utilizes a variation of a CNN architecture. The second model utilizes a RNN architecture. Each model has been constructed with Keras [1], in order to maintain a standard interface from which the text classification experiments may be easily reproduced. This approach is constrained by the typical advantages and disadvantages associated with neural networks. While they provide better performance on average compared to traditional classifiers, they are technically more complex and normally require hyperparamter tuning. One of the goals of this research is to avoid extensive hyperparameter tuning in an attempt to keep the research accessible.

The contributions of this paper are two-fold: The primary contribution is the two implemented text classifiers and the results achieved by each. While CNN and RNN text classifiers are not a new concept, comparative analyses based upon the effectiveness of the approaches are rare within the literature. The ancillary contribution of the paper is to provide an accessible, convenient environment from which further comparative research may be undertaken quickly and effectively. The solution discussed within this paper aims to be a platform from which business decisions may be made rather than a platform from which a business solution may be derived.

The remainder of this paper details the process of implementation for each text classifier, contained within section 5. The results received by the models may be observed within section 6. The methodologies and architecture utilized in order to obtain said results may be found in sections 3 & 4 respectively. Also included within the paper is a brief overview of related work within the field, found in section 2.

2 Related Work

In order to begin progress towards addressing the proposed research question, observation must first be made of the existing work within the relevant fields. Initial research centered around the field of text classification and the methodologies utilized commonly within the field.

[1] https://keras.io/

2.1 Text Classification

The work of Ikonomakis et al. (2005) offers an in-depth introduction to the concept of text classification. Included within, is a comprehensive overview of the processes involved in prepping data for text classification, with empirical evidence reinforcing the concepts presented. However, this work shows its age in the complete lack of acknowledgement of the abilities of neural networks for tasks within the field. The work also fails to make any mention of validation techniques. Joulin et al. (2016) once again provides deep insight into the workings of text classifiers and presents methods to increase the speed of model training and testing. In order to do this however, the use of linear classifiers is advocated which in turn comes at the cost of the models generalization capabilities. The research presented by Forman (2003) analyzes and presents a breakdown of several performance metrics and their relevance in text classification tasks. This research provides excellent basis for model evaluation, however, several key metrics are omitted. Inductive transfer, as presented by Mikolov, Sutskever, Chen, Corrado and Dean (2013) and further explored in Howard and Ruder (2018), presents a strong case for the adoption of this concept, however, it may be considered outside the purview of the research conducted within this paper as it tends to receive greater results with convolutional architectures, potentially skewing results.

2.2 Convolutional Neural Networks

Inspired by the early work of (Hubel and Wiesel (1962);Hubel and Wiesel (1968);Hubel and Wiesel (1977)) research conducted by Fukushima (1980) introduced what may be considered as the initial CNN architecture. The architecture described within the paper is quite different from what is considered a modern CNN architecture. The key limiting factor of this architecture is the absence of the back-propagation algorithm LeCun et al. (1989). The introduction of this algorithm paved the way for the increased complexity and effectiveness of modern CNN implementations as it allows for potent loss control. Recent efforts within the field of text classification have been enhanced with the advent of pre-trained word embeddings such as the approach presented by Pennington et al. (2014). Socher et al. (2013) and Collobert et al. (2011) achieved excellent results in various fields of NLP research utilizing this technique.

The research produced by Kim (2014) further reinforces the use of pre-trained word embeddings. The implemented CNN architecture, with little hyperparameter tuning, achieves results comparable to that of the state of the art. An in-depth comparison of CNN versus traditional Bag of Words and its variations is presented in Zhang et al. (2015). Once again the CNN architecture performed admirably while outperforming Bag of Words.

2.3 Recurrent Neural Networks

Similar to the origin of the CNN architecture, the initial mention of RNN architecture details an implementation lacking key features associated modern RNN implementations. Two papers (Pearlmutter (1989);Cleeremans et al. (1989)), outline what may be considered the foundation of modern RNN architecture. However, they fail to address to issue of vanishing gradients which can be devastating to the effectiveness of a model. LSTM as introduced by Hochreiter and Schmidhuber (1997) resolves the vanishing gradient problem by introducing the concept of recall into the RNN architecture. This allows for more layers to be added to the solution, increasing model flexibility. While LS-TM addresses the vanishing gradient problem, exploding gradients may still present a problem for certain implementations. Gated Recurrent Units (GRU) as introduced by Chung et al. (2014), have been shown to achieve results similar to LS-TM on certain datasets. This has been challenged recently, however, as comparisons performed within a paper Weiss et al. (2018), have shown GRU to be outperformed by LS-TM in almost every case. Bidirectional LS-TM implementations have achieved state of the art results in a variety of NLP fields Graves and Jaitly (2014) Fan et al. (2014) Graves et al. (2013), generating renewed interest for its use in text classification.

In summary, while recent research has deemed RNN and CNN approaches to be capable of achieving state of the art results, very little research has been conducted comparing the abilities of each architecture against the other. Lee and Dernoncourt (2016) and Yin et al. (2017) encompass the gamut of available literature comparing the abilities of CNN and RNN for NLP tasks. Each paper provides excellent assessment of the abilities of each architecture, however, neither paper is particularly focused upon text classification. Lee and Dernoncourt (2016) fail to make use of the popular pre-trained word embeddings, proven to increase the performance of models in classification tasks. Yin et al. (2017) provides a broad overview of the abilities of RNN and CNN on various NLP tasks, refraining from focusing on any one task. As such there is a dearth of available literature comparing the abilities of the architectures for text classification, reinforcing the validity of the proposed research question. In addition, of the myriad papers reviewed for this research only 2 provided an open source solution, highlighting a lack of focus on accessibility.

3 Methodology

To contextualize the process that was undertaken for this paper, the pipeline for text classification must first be observed. Text classifiers typically consist of five key components. The first three components consist of the training data, a feature vector and the labels. The training data refers to the text data that is supplied for analysis, typically some form of unstructured data. The feature vector is a descriptor of the training data stored within a vector. The labels are a collection of predefined classes that the model aims to predict. These three components are centered around the dataset and as such are specific to the task at hand. An example you may consider is gaining insight into user sentiment based upon text data gathered from a social media platform with labels detailing particular levels of sentiment: positive, negative and neutral.

The final two components of a text classifier pertain to the machine learning aspects of the classifier, the predictive model and the machine learning algorithm used to construct the model. These final two components may be considered the core facets of the research conducted within this paper, as the results achieved by the analysis hinge on the selected models and their implementation. As mentioned within Section 1, the selected architectures for this analysis are a CNN architecture and a RNN architecture. The procedure undertaken for the purpose of this comparative analysis is detailed below.

3.1 Data

As text classification encompasses a wide variety of tasks, multiple datasets are needed in order to facilitate accurate research. To this end, two sets of data pertaining to two popular text classification tasks have been selected. The first dataset selected for research is the BBC Articles Categorized dataset. This is a further cleaned variant of the data made available by Greene and Cunningham (2006). This data was gathered utilizing Google's BigQuery web service and is now available at [2]. The data consists of of two columns: category and text, where text is a collection of articles and their titles concatenated, while category contains the "genre" in which the article belongs. The data consists of 2126 articles and their respective category, the distribution of which is displayed in Table 1.

Table 1: BBC Articles Category Distribution

Category	**Count**
Sport	511
Business	510
Politics	417
Tech	401
Entertainment	386

The second dataset selected for the purposes of this paper is the Spam Text Message classification dataset. This data was collected by the user "Team AI" from the popular data science website Kaggle, available at [3].This data has been engineered in order to resemble a traditional Spam filtering problem. The data consists of two columns: Type and Message. Type refers to whether the message is categorized as Spam or Ham, while Message refers to a collection of text messages. The dataset consists of 5157 text messages with each being labeled either Spam or Ham. The distribution of these labels may be observed in Table 2

Table 2: Spam and Ham Distribution

Category	**Percentage**
Spam	13
Ham	87

While each dataset has been cleaned prior to its use within these experiments, there has been no quality control upon the contents of the columns. To rectify any potential issues that may arise from this, within each architecture implementation several steps have been taken to ensure that the datasets meet a certain standard. This includes performing checks for URLs and other unwanted or illegal characters, that would otherwise undermine the efficiency of the the text classifiers.

3.2 Environment

The environment utilized for the purposes of the research conducted in this paper was consistent across all facets of the research, in order to facilitate the reproduction of results. All experiments were conducted utilizing the python 3.4 distribution in conjunction with both Anaconda and Jupyter notebooks. Anaconda was instrumental in creating and

[2] https://storage.googleapis.com/dataset-uploader/bbc/bbc-text.csv

[3] https://www.kaggle.com/team-ai/spam-text-message-classification/metadata

maintaining the Python environment and the packages utilized over the course of the experiments. This ease of use was key in enabling the accessible nature of the developed solution, as the modular nature of Python and Keras allowed for intuitive implementation of the two architectures. Jupyter uniquely enables the breakdown of the implemented solution into individual sections for increased accessibility. The cell based infrastructure of Jupyter enhances the ability of future users of the solution to modify it to their own ends.

Global Vectors (GloVe) Pennington et al. (2014), has recently been adapted as one of the premier pre-trained word embeddings, alongside Word2Vec Mikolov, Chen, Corrado and Dean (2013). Pre-trained word embbedings utilize an external corpus in order to efficiently represent semantic information that may not be present within the data currently being fed to the model. This can lead to tremendous increases in training speed and accuracy of the model. Pre-trained word embeddings grow exceptionally more influential as the dataset grows smaller in size, however, it has been shown to still outperform traditional embedding layers even upon larger datasets [4]. As pre-trained word embeddings are so readily available to researchers, GloVe will be utilized within the implemented solution in an attempt to minimize training time costs.

Aside from the software environment another key factor in allowing for the reproduction of results is the specifications of the machine in which the experiments were undertaken. In order maintain accurate results for the comparative analysis, each experiment was undertaken upon the same machine. The machine in question runs on a x64 bit Windows 10 installation. The CPU of the machine is a Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz with 12GB of RAM. The graphical processor in use on the machine during experimentation was the Intel(R) HD Graphics 530.

3.3 Process

The research conducted by Yin et al. (2017) was instrumental in defining the research methodology conducted within this paper. The key components to be obtained via experimentation is the time taken per epoch, the loss and accuracy figures for both training and validation of the various models. The solution developed for the purposes of this paper has been designed to aid in the decision process, specifically decisions regarding which neural network to utilize. It can be seen as a companion to Cross-Industry Standard Process for Data Mining (CRISP-DM), as it is designed as a proving ground for various architectures to aid in assisting the data understanding and business understanding steps of the CRISP-DM methodology. An in-depth overview of the steps taken to implement the models can be observed within Section 5.

4 Design Specification

4.1 CNN Architecture

CNN architectures, similar to many machine learning algorithms, are generally comprised of the same components. What truly effects the capabilities and performance of a CNN implementation is the manner in which it is constructed. Drawing inspiration from both

[4] https://towardsdatascience.com/pre-trained-word-embeddings-or-embedding-layer-a-dilemma-8406959fd76c

Zhang et al. (2015) and [5], a text classifier was constructed for the purposes of this project. Rather than operating on a letter by letter basis or even a single word basis, the architecture of CNNs allows for the observation of expressions such as ”I like”. This can be achieved by utilizing the activation function in concert with the convolutional layer. The output of each kernel may then be concatenated in order to allow the CNN to detect expressions no matter their position within a sentence. The CNN constructed for this paper makes use of alternating layers, initially between convolutional and max pooling layers, finishing with a single flattening layer and two dense layers. Each convolutional layer makes use of the popular ReLU activation function whereas the dense layer utilizes the softmax activation function. The softmax function is utilized in this manner as it produces a probability rather than a simple number, making it ideal for classification tasks. The full architecture alongside the inputs and outputs of each layer may be observed within Figure 3.

4.2 RNN Architecture

In comparison to the CNN architecture shown in Figure 3, the RNN architecture as shown in Figure 4 may look rather simplistic. However, the core concepts behind the RNN implementation are complex. The key feature of RNNs that makes them so versatile is the concept of persistence. Rather than observe each task as an object with no accompanying information, the chain-like structure of RNNs allows for information to be passed from node-to-node. Basic RNNs utilize this to great effect, however, in cases where context is crucial, such as text classification, the problem of long-term dependencies arises. To combat this issue LS-TM Hochreiter and Schmidhuber (1997) variants of RNNs may be used. LS-TMs solve the issue of long-term dependencies by streamlining the manner in which information is passed from node-to-node. By utilizing a new variant of node, known as ”gates”, LS-TMs allow information to be passed from node-to-node with only minimal interaction with the loops running within the nodes. The gates restrict the interactions between nodes and the information stream, allowing for only key information to be passed therefore increasing the efficiency of the network.

The implemented solution draws inspiration from both Li et al. (2015) and [6], in order to achieve the best possible results without in-depth optimization of the network. With the inclusion of a bidirectional layer the network gains access to both past and future information. This provides a tremendous advantage when context is key, as is the case for the experiments within this paper. Once again the softmax activation function is utilized in order to produce a digestible output.

4.3 Keras Framework

The Keras deep learning framework was utilized extensively in order to implement the models produced for this paper. The Keras framework offers an intuitive approach to API implementation that, when utilized with detailed documentation, reduces the cognitive load placed upon the developer. This is a fey factor to consider for the proposed solution, as ideally future users of the solution will implement models of their own design. Thus it is critical that future users be able to quickly digest and comprehend the framework in order to scale-up their solutions in a reasonable time-frame.

[5] http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

[6] https://github.com/keras-team/keras/blob/master/examples/imdb_bidirectional_lstm.py

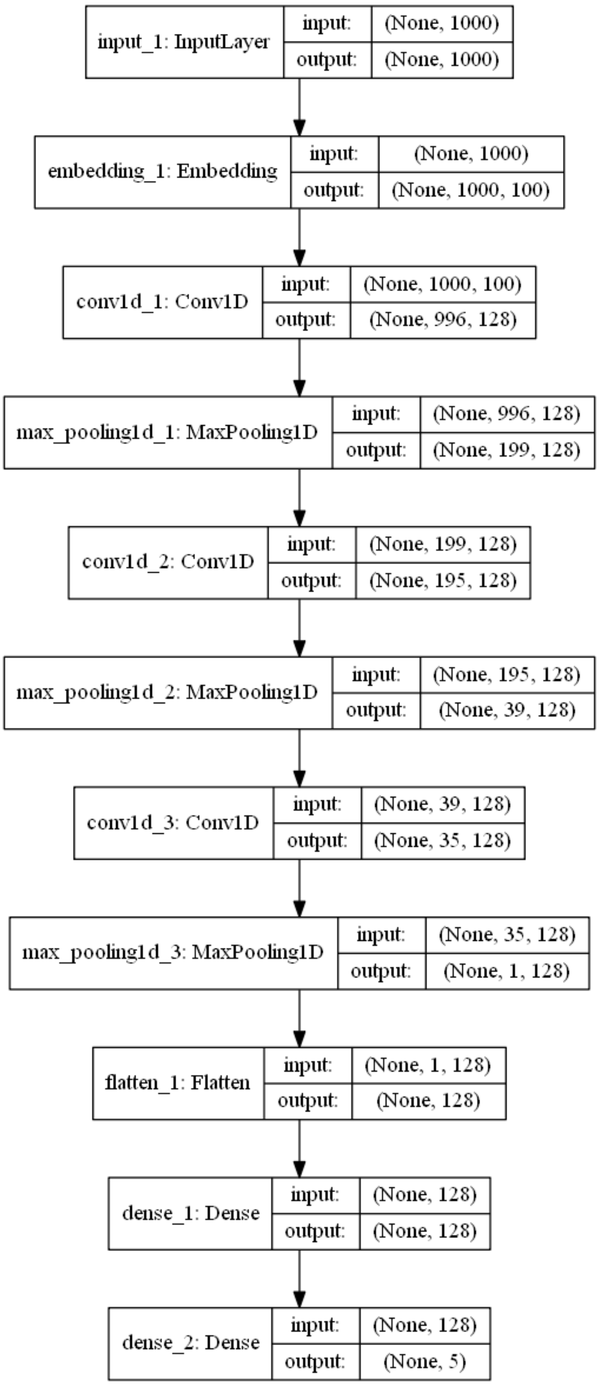

Figure 3: The CNN Architecture Utilized

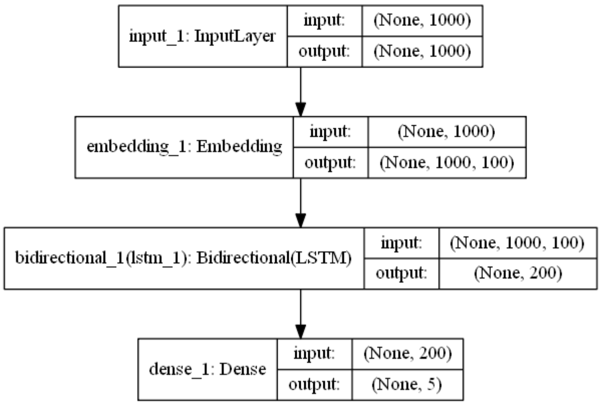

Figure 4: The RNN Architecture Utilized

Keras further aids in this goal with its module-based infrastructure. New models may be swiftly constructed by selecting the desired modules. This feature also enables extensive modification to existing architectures. Keras is also provides a high degree of versatility due to inherent support of multiple backend engines. This allows for models to be developed on one platform and tested on various other platforms. For reference, the models implemented in this paper where created utilizing the Theano [7] backend, however they may be transferred to an implementation utilizing TensorFlow, CNTK or even a MXNet backend.

5 Implementation

The final implementation of of the proposed solution is composed of six individual files. The files consist of the two dataset files and 4 individual Jupyter notebooks. Each Jupyter notebook corresponds to an implemented model, two CNN models and two RNN models, one of each type for each dataset. As each model follows the 2 corresponding architectures, CNN and RNN, the implementation of a single CNN and a single RNN will be discussed.

The first step in each implementation in the importing of the requisite packages and Keras modules. As the datasets are stored within CSV files, they must be imported into the Python environment. To do this the pandas package was imported and utilized. In order to facilitate streamlined handling of the large amount of numerical data the NumPy package was imported. As a text classifier is the ultimate goal, character strings are likely needed for inputs. To address this the pickle package is utilized. For cases in which the data is scraped from the web or perhaps is text data containing HTML, the package Beautiful Soup is imported. As Keras is modular in nature, each of the required modules may be imported as necessary. For performing preprocessing upon the datasets, the Tokenizer and pad_sequences modules are required. The construction of the models themselves comes down to the layers needed within the model, as each layer may be imported as a corresponding module. For the CNN implementation the Embedding, Dense, Input, Flatten, Conv1D, MaxPooling1D, Dropout and Model modules are required. For the RNN implementation the Embedding, Dense, Input, LSTM, GRU, Bi-

[7] http://deeplearning.net/software/theano/

directional and Model modules are required. The final Keras import is ModelCheckpoint, this is helpful for observing results of a model between epochs for simplified comparison. For visualization purposes the versatile matplotlib.pyplot package has been utilized.

Once the package imports have been handled the next steps are importing the data and then transforming the data for use within the models. A function is defined for cleaning the data of some unwanted characters and the data is imported into a data.frame object. Once this is done the data may be passed through Beautiful Soup in order to remove any residual HTML and select undesirable characters. Tokenization is the next step in the process of preparing the data for modeling. This refers to the task of breaking-down the sentence structure of the text data into a form more manageable for the neural networks. The Tokenizer module in Keras is utilized for this purpose. The tokenized data is then passed into a sequence which is then in-turn utilized to form the validation split upon the data, a key technique for training and testing machine learning models. The final step before implementing the model is embedding. Embedding is the process of transforming the 2D tensor input into a 3D tensor that is then passed into the model. An embedding matrix is created utilizing GloVe which is then used to create an embedding layer, that serves as the second layer of the model.

The CNN model is comprised of 11 layers, including the already created embedding layer. Aside from the requisite input and output layer, the core of the model is comprised of 6 alternating convolution and pooling layers followed by a flattening and dense layer. The alternation between convolutional and pooling layers, is necessary to avoid the curse of dimensionality being brought to bear upon the model. The activation function for the majority of the layers is ReLu with the exception of the output dense layer, which utilizes the softmax function to transform the output of the model into probabilities for easier digestion. In contrast the RNN model is comprised of 4 layers, owed to Keras' in-built support for bidirectional LS-TM models. The models are run over 15 epochs with a batch size of 2, with the model check-pointing after each epoch in order to update validation accuracy figures.

Once the models have run their course the final step to perform, is visualization. For the purposes of this solution, two visualization tools have been utilized. The first of which is an in-built util of Keras that allows for the printing of the model architecture. This tool was utilized to generate the model architectures that may be observed in Figures 3 & 4. The second tool utilized is the popular matplotlib package. Matplotlib was instrumental in creating the visualizations presented in Section 6.

6 Evaluation

Below, is documented the experiments designed and undertaken for the purposes of testing the effectiveness of both CNN and RNN architectures with various text classification tasks. Both experiments have been undertaken in the conditions outlined within Section 5, utilizing the architectures discussed within Section 4. To ensure reproducibility of the results obtained, each model has been constructed to meet two key criteria: each model is generalized rather than specialized and the data tensor for each experiment is limited to a length of 1000 in order to avoid inconsistencies.

6.1 Experiment 1: BBC Article Categorization

The first experiment undertaken for this paper was based around the task of classification of an assortment of articles published by the BBC. The models were tasked with classifying text data taken from BBC articles of varying subject matter. The models were supplied a selection of 5 labels, ranging from Sport to Politics, and then tasked with learning the semantic reasoning and the contextual clues behind the individual topics. The results achieved by both model architectures may be observed below. Figures 5 & 6 provide an overview of the training and validation accuracy achieved by both model architectures over the epochs. Figures 7 & 8 show the loss values received by both models. Table 3 provides a comparison of the results achieved by the different model implementations.

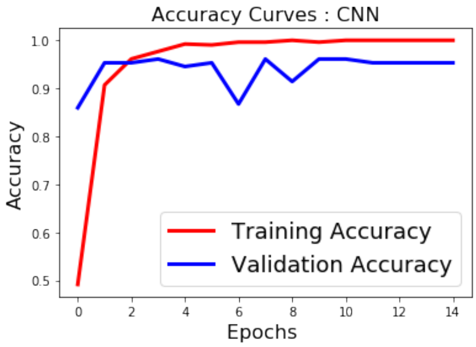

Figure 5: Accuracy Results Achieved with CNN

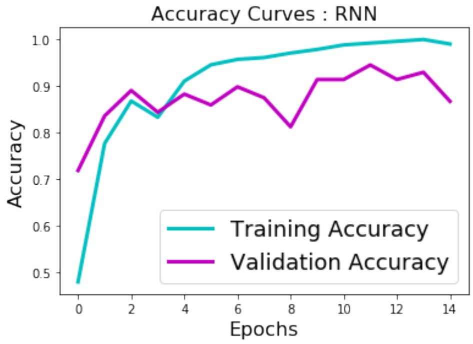

Figure 6: Accuracy Results Achieved with RNN

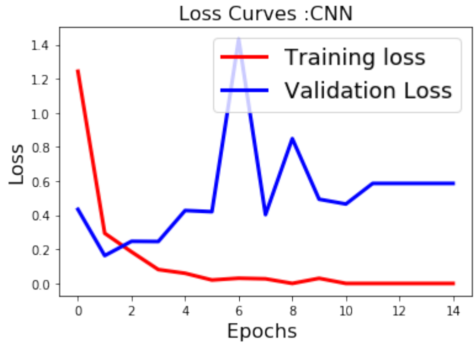

Figure 7: Loss Results Achieved with CNN

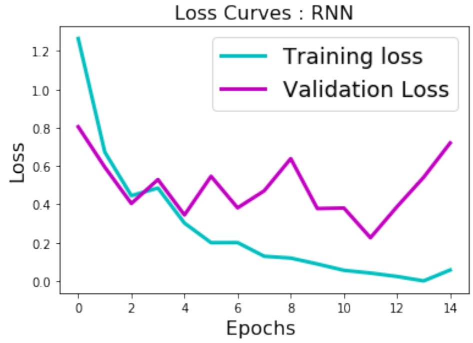

Figure 8: Loss Results Achieved with RNN

Table 3: BBC Article Categorization Results

Algorithm	Time/Epoch	Training Accuracy	Validation Accuracy
CNN	35ms	0.9961	0.96094
RNN	745ms	0.9903	0.94531

6.2 Experiment 2: Spam Detection

The second experiment undertaken for the purposes of this research is the a Spam detection task. Popular within the literature, this task challenges the models with determining whether a message can be counted as Spam, an unwanted message, or Ham, a genuine message. The models have been supplied a selection of data, as described in 2. The accuracy results achieved by the RNN and CNN implementations may be observed within Figures 9 & 10. Similarly the loss curves for both models may be seen in figures 11 & 12. A detailed comparison of the achieved results can be viewed within Table 4.

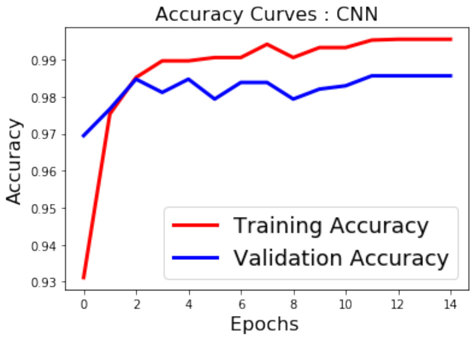

Figure 9: Accuracy Results Achieved with CNN

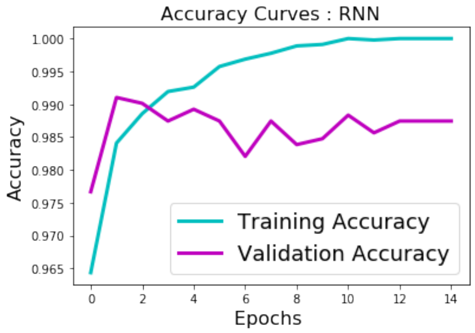

Figure 10: Accuracy Results Achieved with RNN

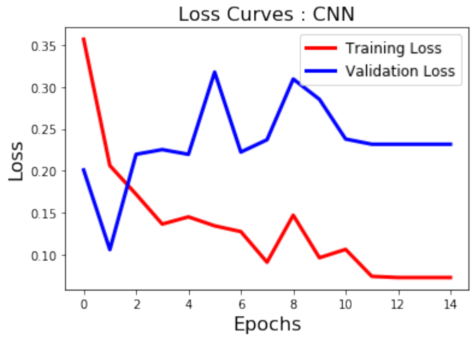

Figure 11: Loss Results Achieved with CNN

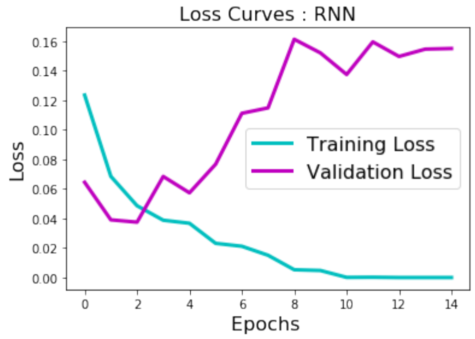

Figure 12: Loss Results Achieved with RNN

Table 4: Spam Detection Results

Algorithm	Time/Epoch	Training Accuracy	Validation Accuracy
CNN	30ms	0.9955	0.98564
RNN	338ms	0.9989	0.99102

6.3 Discussion

Regarding the results achieved over the course of the first experiment: it becomes immediately apparent from observation of the training accuracy curves that the CNN model quickly increases in training accuracy before plateauing at a very high value. The RNN implementation, however, begins sedately before eventually climbing to results comparable to those of the CNN implementation. When observing the validation accuracy of both models a similar trend is evident. The CNN model is quick to achieve high accuracy while staying quite consistent whereas the RNN model fails to reach the same level of consistency over the epochs. The loss curves once again show a similar trend, however the RNN implementation is more competitive in this case, achieving exceptional scores in validation loss. The CNN implementation seems to have suffered from an exploding gradient during epoch 6, but swiftly corrected the issue. These results generally reinforce the research results achieved by Hochreiter and Schmidhuber (1997), Socher et al. (2013). The RNN implementation shows no sign of exploding gradient, owed to its LS-TM architecture, while the CNN is much faster when it comes to training.

The results achieved in the second experiment, reveal quite a different story to the results of the first experiment. The accuracy curves remain somewhat similar to those of the first experiment, however, there is a marked increase in validation accuracy between both models. Again, the RNN implementation seemingly somewhat trails behind the results achieved by the CNN implementation. The true surprise, however, comes with observation of the loss achieved by both models. The RNN model far outstripped the performance of the CNN model in regards to loss, comfortably sitting 0.1 below the loss reported by the CCN model. Upon observation of the results within Table 4, it becomes apparent that the most effective model in terms of accuracy is the RNN model, however the increase is slight and comes at the cost of significantly longer training time. This echoes the sentiments reported by Yin et al. (2017).

In retrospect the 1000 limit upon the max sequence length should be tweaked for future research, as it limited the data the model could draw from to a negative result. Rather than a clear victor outlined by an exceptional performance, the models remain rather close in performance. Perhaps a larger sample size could impact this.

7 Conclusion and Future Work

In this paper two individual neural network architectures, CNN and RNN, were compared and contrasted for their viability for the purposes of text classification. The architectures were implemented in a comprehensive and accessible manner, in order to facilitate use by those new to the field and to ensure those wishing to re-purpose the solution, may do so in a timely manner. It was determined that CNNs achieved high validation accuracy with consistency, however, RNNs outperformed in both validation accuracy and loss on a smaller sample size. The key difference observed between the two architectures is training time, with RNN taking far more time to be trained than CNN. Thus while they may be comparable in performance, CNN is likely more appealing to many due to low time costs. Also observed over the course of the research is that batch size and epochs seem to have a strong effect on the overall effectiveness of the models and as such is likely key to creating an optimum model. While this solution holds possible potential for commercialization, as a new entrant to the market geared toward new entrants to the field. This seems unwise as the market is quite niche and is currently populated with potential competitors such as RapidMiner[8] and Wekka[9].

Future work based upon this research would likely focus upon the inclusion of hyper-

[8] https://rapidminer.com/

[9] https://www.cs.waikato.ac.nz/ml/weka/

parameters into the experimentation. Hyperparameter optimization is a core component to the creation of more sophisticated machine learning models. With appropriate optimization the solution may be introduced to greater sized datasets, in order to gain a proper gauge on the performance of both RNN and CNN architectures on big data.

References

Anand, A., Chakraborty, T. and Park, N. (2016). We used neural networks to detect clickbaits: You won't believe what happened next!

Chung, J., Gulcehre, C., Cho, K. and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling, *arXiv preprint arXiv:1412.3555* .

Cleeremans, A., Servan-Schreiber, D. and McClelland, J. L. (1989). Finite state automata and simple recurrent networks, *Neural computation* **1**(3): 372–381.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011). Natural language processing (almost) from scratch, *Journal of machine learning research* **12**(Aug): 2493–2537.

Fan, Y., Qian, Y., Xie, F.-L. and Soong, F. K. (2014). Tts synthesis with bidirectional lstm based recurrent neural networks, *Fifteenth Annual Conference of the International Speech Communication Association*.

Forman, G. (2003). An extensive empirical study of feature selection metrics for text classification, *Journal of machine learning research* **3**(Mar): 1289–1305.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biological cybernetics* **36**(4): 193–202.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks, *International conference on machine learning*, pp. 1764–1772.

Graves, A., Mohamed, A.-r. and Hinton, G. (2013). Speech recognition with deep recurrent neural networks, *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, pp. 6645–6649.

Greene, D. and Cunningham, P. (2006). Practical solutions to the problem of diagonal dominance in kernel document clustering, *Proceedings of the 23rd international conference on Machine learning*, ACM, pp. 377–384.

Hidaka, A. and Kurita, T. (2017). Consecutive dimensionality reduction by canonical correlation analysis for visualization of convolutional neural networks, Vol. 2017, pp. 160–167.

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

Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification, *arXiv preprint arXiv:1801.06146* .

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *The Journal of physiology* **160**(1): 106–154.

Hubel, D. H. and Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex, *The Journal of physiology* **195**(1): 215–243.

Hubel, D. H. and Wiesel, T. N. (1977). Ferrier lecture: Functional architecture of macaque monkey visual cortex, *Proceedings of the Royal Society of London. Series B, Biological Sciences* pp. 1–59.

Ikonomakis, M., Kotsiantis, S. and Tampakas, V. (2005). Text classification using machine learning techniques., *WSEAS transactions on computers* **4**(8): 966–974.

Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. (2016). Bag of tricks for efficient text classification, *arXiv preprint arXiv:1607.01759* .

Kim, Y. (2014). Convolutional neural networks for sentence classification, *arXiv preprint arXiv:1408.5882* .

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition, *Neural computation* **1**(4): 541–551.

Lee, J. Y. and Dernoncourt, F. (2016). Sequential short-text classification with recurrent and convolutional neural networks, *arXiv preprint arXiv:1603.03827* .

Li, J., Luong, M.-T. and Jurafsky, D. (2015). A hierarchical neural autoencoder for paragraphs and documents, *arXiv preprint arXiv:1506.01057* .

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

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013). Distributed representations of words and phrases and their compositionality, *Advances in neural information processing systems*, pp. 3111–3119.

Pang, B., Lee, L. et al. (2008). Opinion mining and sentiment analysis, *Foundations and Trends® in Information Retrieval* **2**(1–2): 1–135.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks, *Neural Computation* **1**(2): 263–269.

Pennington, J., Socher, R. and Manning, C. (2014). Glove: Global vectors for word representation, *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank, *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642.

Weiss, G., Goldberg, Y. and Yahav, E. (2018). On the practical computational power of finite precision rnns for language recognition, *arXiv preprint arXiv:1805.04908* .

Yin, W., Kann, K., Yu, M. and Schütze, H. (2017). Comparative study of cnn and rnn for natural language processing, *arXiv preprint arXiv:1702.01923* .

Zhang, X., Zhao, J. and LeCun, Y. (2015). Character-level convolutional networks for text classification, *Advances in neural information processing systems*, pp. 649–657.