T.C
ALTINBAS UNIVERSITY
Information Technology

DEEP LEARNING TEXT CLASSIFICATION USING CNN, RNN & HAN

Abdulrahman Maadh Abdulaleem

Master Thesis

Supervisor

Prof. Dr. Oguz BAYAT

Istanbul 2019

DEEP LEARNING TEXT CLASSIFICATION USING CNN, RNN & HAN

by

ABDULRAHMAN MAADH ABDULALEEM

INFORMATION TECHNOLOGY

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAŞ UNIVERSITY

2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Abdulrahman Maadh Abdulaleem

DEDICATION

I would like to dedicate this work to my lovely family and specially my mother, for their invaluable efforts when I felt hopeless and weak in solving problems.

ACKNOWLEDGEMENTS

In the name of Allah: the Beneficent and the Merciftl_ Praise and Gratitude be to Allah for giving me strength and guidance: so that this thesis can be finished accordingly I would like to thank my supervisors: Prof_ Dr. OGUZ BAYAT and Asst. Prof. Dr_ TAREQ ABED MOHAMMED Please let me express my deep sense of gratitude and appreciation to both of you for the knowledge: guidance and unconditional support you hawe given me. I wish you all the best and further success and achievements in your life.

ABSTRACT

DEEP LEARNING TEXT CLASSIFICATION USING CNN, RNN & HAN

Abdulrahman Maadh Abdulaleem

M.S, Information Technology, Altınbas University,

Supervisor: prof .Dr. Oguz BAYAT

Co-Supervisor: Dr.,Tareq MOHAMMED

Date: july,2019

Page: 60

Increasingly large document collections require improved information processing methods for searching, retrieving, and organizing text. Central to these information processing methods is document classification, which has become an important application for supervised learning. Recently the performance of traditional supervised classifiers has degraded as the number of documents has increased. This is because along with growth in the number of documents has come an increase in the number of categories.

Keywords: Deep learning, Text classification ,CNN, RNN, HAN, NLP.

TABLE OF CONTENTS

Pages

LIST OF TABLE

Pages

LIST OF FIGURES

Pages

LIST OF ABBREVIATIONS

DL: Deep learning.

NLP: Nature Languages Process.

ANNs: Artificial Neural Networks.

CNN: Convolutional Neural Network.

RNN: Recurrent Neural Network.

HAN: Hierarchical Attention Network .

1. INTRODUCTION

1.1 INTRODUCTION

Nowadays, the amount of information that online available is numerous and increasing rabidly, this amount of data open an world of science and make a lot of attention by researchers. Part of this data is the text data like newspapers, academic researches, articles and books, emails and all kind of text information that need to be organized and automatically processed.

Text classification can be applied in everyday of our life, from news arranged under topics and articles to the email spam detection.

The Nature Languages Process (NLP) is the techniques involve categorization of textual data at the foundation. Dictionaries, knowledge bases and special tree kernels are often used in Traditional text classifiers. In traditional supervised ML methods, a model is created based on a training set in which there are documents with tagged labels and the model is trained on this dataset and later used to predict labels for new documents. Depending on the classification algorithm or strategy used, the classifier might also provide a confidence measure to indicate how confident it is that the classification label is correct. In traditional text classification, a common methodology to do a pre-processing of text to make it suitable to be fed to classifier is TF-IDF (term frequency - inverse document frequency). The TF-IDF weighting for a word increases with the number of times the word appears in the document but decreases based on how frequently the word appears in the entire document set. Most text categorization algorithms represent a document collection as a Bag of Words (BOW).

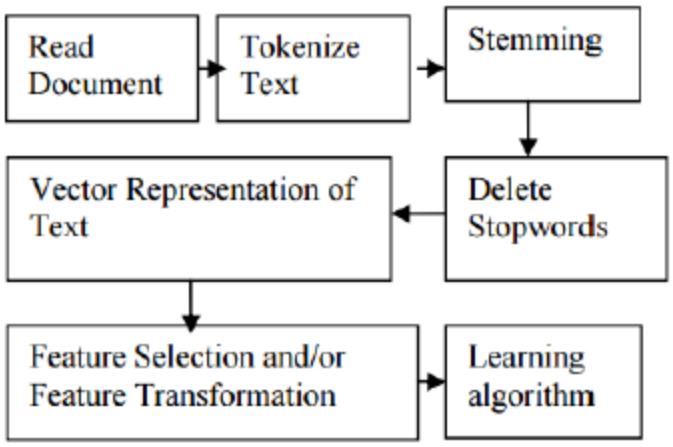

Figure 1.1: Text Classification

The neural systems are arranged to hide the Artificial Neural Networks (ANNs), A neural network is a popular type of ML algorithm that consists of multiple layers. the input and the output of them. A series of synaptic weights combine the neurons. An ANN is a powerful tool in a number of problems for the determination of patterns, predictions and regressions. During the learning process, the ANN constantly changes its synaptic values until sufficient knowledge is acquired (unless a number of iterations are achieved or the error value of the target is met). The ability of the ANN to generalize the problem in samples other than those used during the training stage must be evaluated following a completion of the learning or training. Finally, the ANN is expected to correctly classify the patterns of a specific problem during training and testing. Several classic ANN algorithms were proposed and developed in recent years.

1.2 DEEP LEARNING:

Deep learning is a branch of machine learning, where algorithms learn independently from excessive amounts of information. Similarly, to people, these algorithms get smarter with experience by gathering and processing more and more data.

1.3 HOW DOES DEEP LEARNING WORK?

The magic in deep learning networks is discovering the pattern and structure behind vast amounts of data. The computational model consists of multiple layers, called neural networks, where data is processed.

1.4 WHAT HAPPENS IN NEURAL NETWORKS?

We have 3 elements in the neural network: the input layer, which is the data we want to analyze. At least 2 hidden layers, or nodes, which complete the computation with the deep learning algorithm. In the output layer, we have the calculated result.

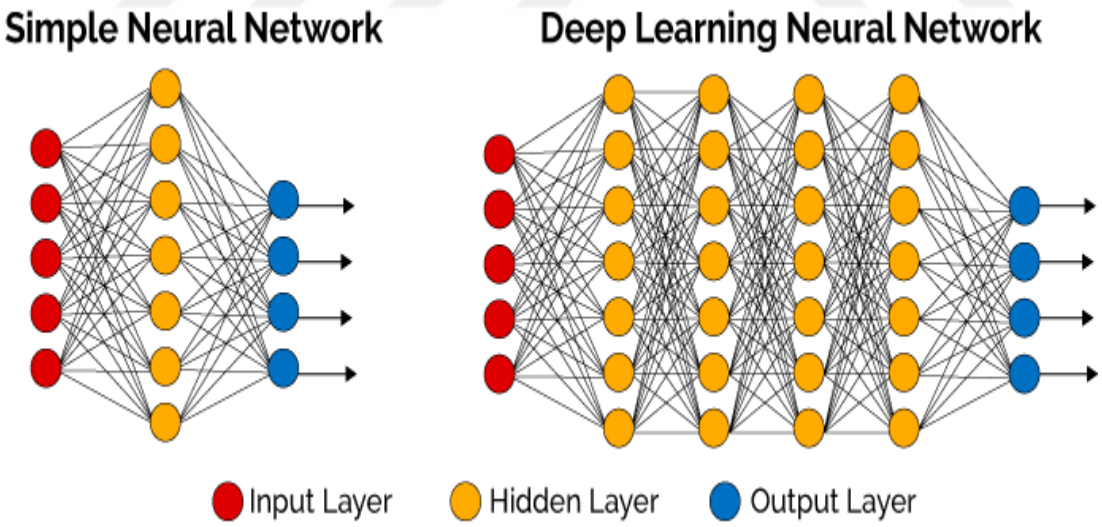

Figure1.2: Deep learning Neural Network

The real actions happen in the hidden layers. The calculation is performed via connections, which contain the input data, a pre-assigned weight, and a path defined by the Activation Function.

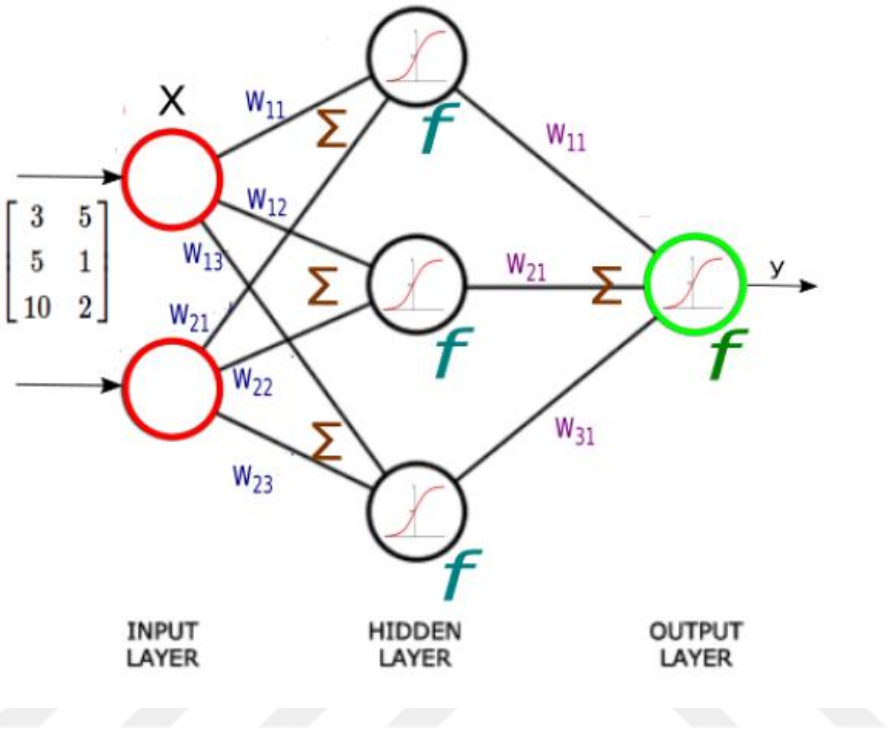

Figure 1.3: ANNS

In the real world, several algorithms exist for classification such as Convolutional Neural Network (CNN)**,** Recurrent Neural Network (RNN)**,** Hierarchical Attention Network (HAN).

1.5 HIERARCHICAL ATTENTION NETWORKS

The overall architecture of the Hierarchical Attention Network (HAN) is shown in Fig. 3. It consists of several parts: a word sequence encoder, a word-level attention layer, a sentence encoder and a sentence-level attention layer. We describe the details of different components in the following sections.

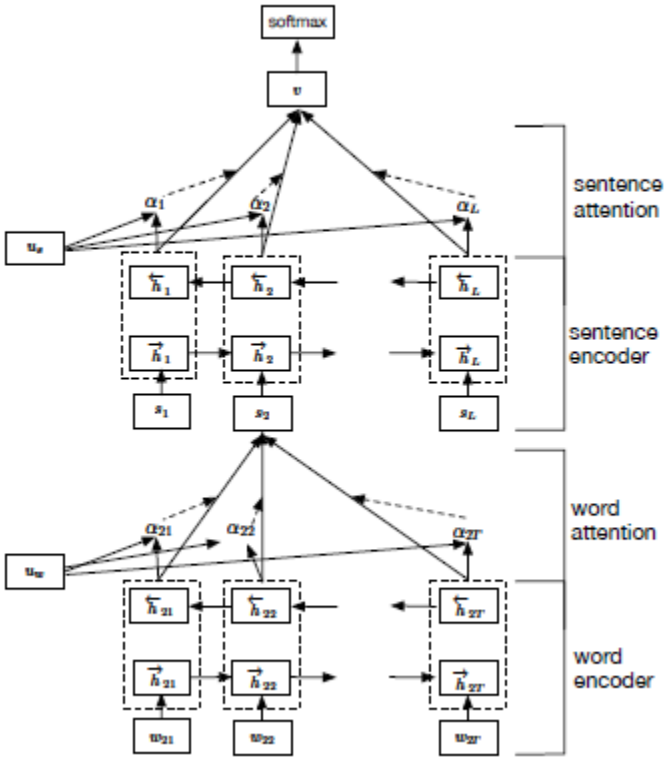

Figure 1.4: HAN

1.6 RNN

Neural networks are models that are motivated by the activities of a human cerebrum, offering a substitute strategy for registering by displaying non-straight connections among data sources and outputs—their use for language demonstrating is known as neural language displaying.

A RNN is a sort of neural system that can abuse the successive idea of the info. It passes everything of the arrangement through a feedforward system and gives the yield of the model as a contribution to the following thing in the grouping, taking into consideration

the capacity of data from the past advances. The "memory" controlled by RNNs makes them incredible for language age, as they can recollect the setting of the discussion after some time. RNNs vary from Markov chains, in that they additionally see words recently observed (not at all like Markov chains, which simply take a gander at the past word) to make expectations.

In each emphasis of the RNN, the model stores in its memory the past words experienced and computes the likelihood of the following word. For instance, if the model created the content "We have to lease a ___ ", it currently needs to make sense of the following word in the sentence. For each word in the lexicon, the model doles out the likelihood dependent on the past words it has seen. In our precedent, the words "house" or "vehicle" will have a higher likelihood than words like "waterway" or "supper". The word with the most astounding likelihood is chosen and put away in the memory, and the model at that point continues with the following emphasis.

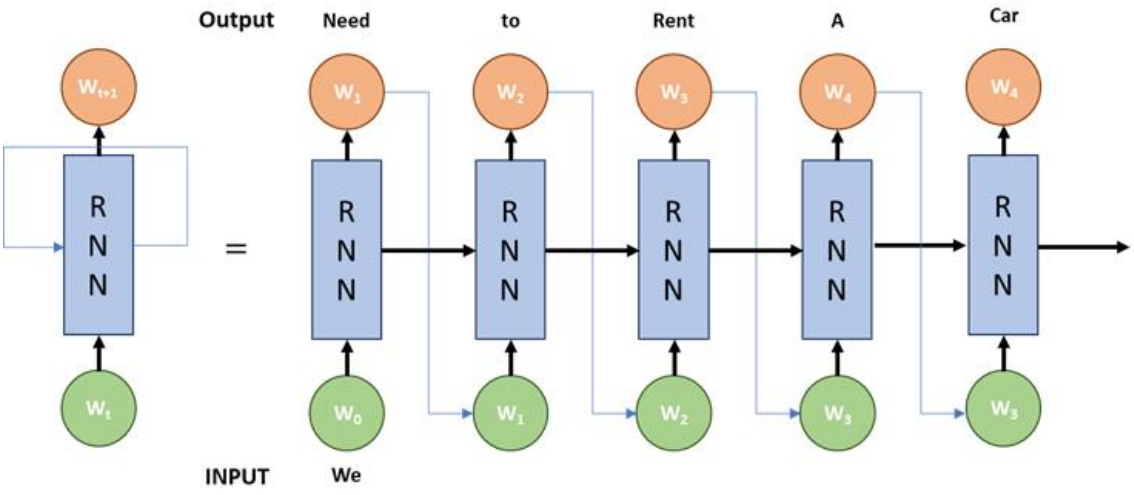

Figure 1.5: Sentence Generation through unrolling of an RNN

1.7 METHODS OF TEXT CLASSIFICATION:

1. **Training text:** It is the input text through which our supervised learning model is able to learn and predict the required class.
2. **Feature Vector:** A feature vector is a vector that contains information describing the characteristics of the input data.
3. **Labels:** These are the predefined categories/classes that our model will predict
4. **ML Algo:** It is the algorithm through which our model is able to deal with text classification (In our case: CNN, RNN, HAN)
5. **Predictive Model:** A model which is trained on the historical dataset which can perform label predictions.

1.8 THESIS OBJECTIVES

In this research we are going to compare a three algorithms to classify the text.
These algorithms are CNN, RNN, HAN. we will compare these algorithms in terms of loss and accuracy. in next chapter the literature review for this field of study, the third chapter is the experiments evaluation and in fourth chapter the conclusion and future works.

2. LITERATURE REVIEW

2.1 LITERATURE REVIEW RELATED ARTIFICIAL NEURAL NETWORKS

The neural systems are arranged to hide the Artificial Neural Networks (ANNs), the input and the output of them. A series of synaptic weights combine the neurons. An ANN is a powerful tool in a number of problems for the determination of patterns, predictions and regressions. During the learning process, the ANN constantly changes its synaptic values until sufficient knowledge is acquired (unless a number of iterations are achieved or the error value of the target is met). The ability of the ANN to generalize the problem in samples other than those used during the training stage must be evaluated following a completion of the learning or training. Finally, the ANN is expected to correctly classify the patterns of a specific problem during training and testing.

Several classic ANN algorithms were proposed and developed in recent years. Many of them can, however, remain caught up in unsolicited solutions; they are far from the ideal or the best solution. In addition, most of these algorithms cannot investigate multimodal or non-continuous surfaces. Other types of techniques are therefore required to train an ANN, such as bio-inspired algorithms (BIAs). The Artificial Intelligence Community is well accepted because BIAs are strong optimization instruments and able to solve very complex problem optimization Mohammed Khalaf Abdullah et al, Reserved 217 problems. BIAs can scan large multimodal and continuous search areas for a certain problem and find the optimal value for the best solution. BIAs are based on the behaviour of nature called swarm intelligence. This concept is defined by [1] as owned by unintelligent agents of limited individual capacity, but intelligent collective behaviour. Several trials use evolutionary and organically inspired algorithms as a fundamental method of ANN training [2]. In neural networks metaheuristic methods are based on local searches, population and other methods, such as cooperative models [3]. The authors present an excellent review of evolving ANN algorithms [2]. An excellent work. The majority of research reports focus, however, on the development and development of synaptic weight, parameters [4] or the evolution of neuronal numbers for hidden layers. Moreover, researchers do not involve the development of transmission functions, an

important element of an ANN that determines each neuron's output. In [5], for example, the authors proposed the combination of ANN and PSO for weight-adjustment with Ant-Colony-Optimization (ACO) methodology. Further studies such as [6] amend the Simulated Annealing PSO (SA) to acquire a set of ANNs with synaptic weights and thresholds. In [7], authors use Evolutionary Programming to get the architecture and weight to solve classification problem and prediction problem. Another example is Genetic Programming [8] where graphs representing different topologies have been obtained. In [9] an ANN was designed to solve a weather forecasting problem with the differential evolution (DE) algorithm. In [10], the authors use a synaptic weight algorithm to change the relationship between daytime rainfall and runoff in Malaya. In [11] the authors only adjust synaptic weights of an ANN to solve classification issues by compared the back-propagation method to the base PSO. In [12] the weighing set is developed by the differential evolution and fundamental PSO. In other works, such as the architecture, transfers and synaptic weights, the three principal features of the ANN were also developed. With the Evolution (DE) differential algorithm [13], the authors resolved the same problem and suggested a new pattern with the authors ' NMPSO (PSO) algorithm. In addition, [14] the author has used an algorithm of the Artificial bee colony (ABC) to develop an ANN with two different fitness functions.

2.2 CNN

Convolutional neural networks with end-to-end training have been used in NLP for the first time in [15]. The authors introduce a new *global* max-pooling operation, which is shown to be effective for text, as an alternative to the conventional *local* max-pooling of the original LeNet architecture [16]. Moreover, they proposed to transfer task-specific information by co-training multiple deep models on many tasks. Inspired by this seminal work, [17] proposed a simpler architecture with slight modifications of [18] consisting of fine-tuned or fixed pretraining word2vec embeddings [19] and its combination as multi-channel. The author showed that this simple model can already achieve state-of-the-art performances on many small datasets. [20] proposed a dynamic *k*-max pooling to handle variable-length input sentences. This dynamic *k*-max pooling is a generalisation of the max pooling operator where *k* can be dynamically set as a part of the network. All of these works are based on word input tokens, following [21] , which introduced for the

first time a solution to fight the curse of dimensionality thanks to distributed representations, also known as *word embeddings*. A limit of this approach is that typical sentences and paragraphs contain a small number of words, which prevents the previous convolutional models to be very deep: most of them indeed only have two layers. Other works [22] further noted that word-based input representations may not be very well adapted to social media inputs like Twitter, where tokens usage may be extremely creative: slang, elongated words, contiguous sequences of exclamation marks, abbreviations, hash tags,... Therefore, they introduced a convolutional operator on characters to automatically learn the notions of words and sentences. This enables neural networks to be trained end-to-end on texts without any pre-processing, not even tokenization. Later, [23] enhanced this approach and proposed a deep CNN for text: the number of characters in a sentence or paragraph being much longer, they can train for the first time up to 6 convolutional layers. However, the structure of this model is designed by hand by experts and it is thus difficult to extend or generalize the model with arbitrarily different kernels and pool sizes. Hence, presented a much simpler but very deep model with 29 convolutional layers. Besides convolutional networks, [24] introduced a character aware neural language model by combining a CNN on character embeddings with an highway LSTM on subsequent layers. [25] also explored a multiplicative LSTM (mLSTM) on character embeddings and found that a basic logistic regression learned on this representation can achieve state-of-theart result on the Sentiment Tree Bank dataset [26] with only a few hundred labeled examples. Capitalizing on the effectiveness of character embedding, [27] proposed a hybrid word character model to leverage the advantages of both worlds. However, their initial experiments show that this simple hybridation does not bring very good results: the learned representations of frequent and rare tokens of words and characters is different and co-training them may be harmful. To alleviate this issue, [28] proposed a scalar gate to control the ratio of both representations, but empiricial studies showed that this fixed gate may lead to suboptimal results. [29] then introduced a finegrained gating mechanism to combine both representations.

They showed improved performance on reading comprehension datasets, including Children's Book Test and SQuAD.

Alexis Conneau et al : They have presented a new architecture for NLP which follows two design principles: "1) operate at the lowest atomic representation of text, i.e. characters, and 2) use a deep stack of local operations, i.e. convolutions and max-pooling of size 3, to learn a high-level hierarchical representation of a sentence." This design has been assessed on eight uninhibitedly accessible vast scale informational collections and they had the capacity to demonstrate that expanding the profundity up to 29 convolutional layers consistently improves execution.

Number of conv. layers per depth In this work:

Table 2.1: Number of conv. layers per depth

Depth:	9	17	29	49
conv block 512	2	4	4	6
conv block 256	2	4	4	10
conv block 128	2	4	10	16
conv block 64	2	4	10	16
First conv. layer	1	1	1	1
#params [in M]	2.2	4.3	4.6	7.8

.

Their models are a lot further than beforehand distributed convolutional neural systems what's more, they beat those methodologies on all dataset sets:

Table 2.2: Large-scale text classification data sets used in our experiments. See (Zhang et al., 2015) for adetailed description.

Data set	#Train	#Test	#Classes	Classification Task
AG's news	120k	7.6k	4	English news categorization
Sogou news	450k	60k	5	Chinese news categorization
DBPedia	560k	70k	14	Ontology classification
Yelp Review Polarity	560k	38k	2	Sentiment analysis
Yelp Review Full	650k	50k	5	Sentiment analysis
Yahoo! Answers	1 400k	60k	10	Topic classification
Amazon Review Full	3 000k	650k	5	Sentiment analysis
Amazon Review Polarity	3 600k	400k	2	Sentiment analysis

To the best of their insight, this is the first time that the "advantage of profundities" was appeared convolutional neural systems in NLP.

Eventhough content pursues human-characterized rules also, pictures can be viewed as crude signs of our condition, pictures and little messages have comparable properties. Writings are additionally compositional for some dialects. Characters consolidate to frame n-grams, stems, words, state, sentences and so on. These comparative properties make the correlation between PC vision and regular language handling very beneficial and they trust future research ought to put into making content preparing models further.

Their work is a first endeavor towards this objective.

In this paper, they center around the utilization of profound convolutional neural systems for sentence grouping errands. Applying comparative plans to other grouping preparing errands, specifically neural machine interpretation is left for future research. It should be researched whether these additionally advantage from having further convolutional encoders. [39]

Tong He et al . They have presented a new system for scene text detection , by introducing a novel Text-CNN classifier and a newly developed CE-MSERs detector. The Text-CNN is designed to compute discriminative text features from an image component.

It leverages highly-supervised text information, including text region mask, character class, and binary text/non-text information. They formulate the training of Text-CNN as a multitask learning problem that effectively incorporates interactions of multi-level supervision.

Structure of the Text-Attentional Convolutional Neural Network:

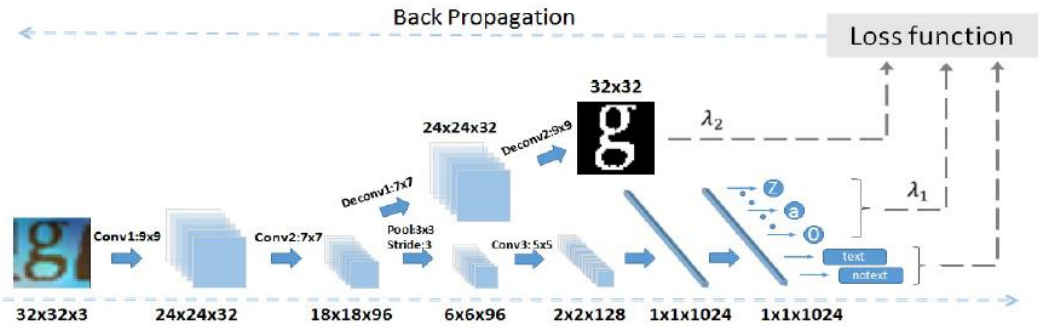

Figure 2.1: Text-Attentional Convolutional Neural Network

They show that the informative multilevel supervision are of particularly importance for learning a powerful Text-CNN which is able to robustly discriminate ambiguous text from complicated background. In addition, they improve current MSERs by developing a contrast enhancement mechanism that enhances region stability of text patterns. Extensive experimental results show that our system has achieved the state-of-the-art performance on a number of benchmarks[40] .

Their research aims at the problem that SVM is easily affected by datasets in short text classification, and proposes to combine CNN with SVM to improve the classification effect. Their datasets as shown :

According to testing with Twitter users remarks, the results show that the SVMCNN with pre-trained initialization performs better than other algorithms. SVMCNN can play a role in the public opinion analysis and sensitive information identification on the online social platform, which helps to guide and maintain a safe and pure network environment

2.3 HAN

Jantima Polpinij et al. they made a research that aims to propose a method for handling requirement specification documents which have a similar content to each other through a hierarchical text classification. Their method has a two main processes , one is a heavy to classify the requirements specification documents having the same content . and the second method is the light method that elaborate specification requirement documents by using the Euclidean Distance. In their motivation , They said that “text classification has been applied to handle the requirements specifications. This is because the initially collected requirements should be classified into various clusters prior to the analysis phase in order to be usable as input in several requirements analysis methods. It is believed that if the requirements collection is small, it can help to understand the specification requirements better. Therefore, many studies have led to major interest in applying text classification techniques to handling the problem of requirements. An early study can be found in [30,31]. Ormandjieva et al [30] presented the problem of automatic quality assessment of textual requirements from an innovative viewpoint, namely the use of a text classification technique. They also proposed a quality model for the

requirements text and a text classification system to automate the quality assessment process. Finally, this study provided a benchmark for such an evaluation and an upper bound on what automatic requirements quality assessment tools can be expected to achieve. Meanwhile, Ko et al [31] applied text classification to reduce the number of specification documents because they believed that slimming down the number of requirements contributes to a better understanding of the impact of the requirements." The heavy method was the Naive Bayes (NB) is a common machine learning for text classification because it executes perfectly in a lot of fields, and it's easy to use . this method estimate as following.Eq (2.1) , (2.2)

$$P(c_j \mid d_i) = \arg\max P(c_j) \prod_{t=1}^{|V|} P(w_t \mid c_j)^{n(w_t, d_i)} \tag{2.1}$$

Where the $p(c_j|d_i)$:

$$P(w_t \mid c_j) = \frac{1 + \sum_{d_i \in c_j} n(w_t, d_i)}{|V| + \sum_{t=1}^{|v|} \sum_{d_i \in c_j} n(w_t, d_i)}$$

$$P(c_j) = \frac{|c_j|}{\sum_{j'}^{|C|} |c_j^{/}|} \tag{2.2}$$

Then they choose the best p(c|d) as a best class for the documents.

The light method as shown like this : Eq (2.3)

$$d(x_1, x_2) = \sqrt{(x_1 - x_2)(x_1 - x_2)^2}$$

$$= \sqrt{\sum_{i=1}^{m} (x_{1i} - x_{2i})^2} \tag{2.3}$$

In this process, they grouped the documents with smaller d(x1, x2) into the same cluster, otherwise, assign them into different groups. In their experiments they choose a 100 document , 70% of them for training and 30% for testing based on 5 folds cross validation .their experiments was into 5 class and their result was precision which is 0.91 and recall which is 0.92 and f-measure which is 0.915 . They said that "The experimental results show an accuracy of 80% for the Naïve Bayes classifier. The results of the Naïve Bayes classifier then are quite good in terms of accuracy. However, the NB

text classifier model is actually built based on a word-probability or word count approach and it classifies a text to a class according to the words appearing in the text content, together with the probabilities of the words learned from a training set. A strong assumption in the approach is that the quantity of interest is governed by the distribution of word probabilities, and the words in the text are independent from one another. For this reason, it may lead to the accuracy of the NB text model being decreased. These may affect for the next process that is to re-classify by using the Euclidean Distance algorithm."

Their result from the hierarchical text classifier shown that precision is 0.93 and recall is 0.92 and f-measure is 0.925.

Based on their results, it can demonstrate that their method can provide more effectiveness for handling the requirement specification documents [30] .

Park et al . They introduce a hierarchical attention network to classify a documents . their model has two features , one is that has a hierarchical structure that mirrors the hierarchical structure of documents, and second the model has two stage of attention mechanisms that apply at word and sentence stage . their experiments was done on six large texts . Experimental results demonstrate that their model performs significantly better than previous methods. Visualization of these attention layer illustrates that our model is effective in picking out important words and sentences. [31].

Kamran Kowsari et al . In their research they presents another way to deal with hierarchical document classification, HDLTex, that joins different profound learning ways to deal with produce progressive characterizations. Testing on an informational index of reports acquired from the Web of Science demonstrates that blends of RNN at the more elevated amount and DNN or CNN at the lower level delivered exactness reliably higher than those possible by ordinary methodologies utilizing na¨ıve Bayes or SVM. These results demonstrate that profound learning strategies can give upgrades for record characterization and that they give adaptability to arrange reports inside a chain of command. Thus, they give expansions over current techniques to report arrangement that just consider the multi-class issue. The strategies depicted here can improved in various

ways. Extra preparing and testing with other progressively organized record informational collections will keep on distinguishing models that work best for these issues. Additionally, it is conceivable to broaden the pecking order to multiple dimensions to catch a greater amount of the unpredictability in the various leveled order. For instance, if catchphrases are treated as requested then the progressive system proceeds down numerous dimensions. HDLTex can likewise be connected to unlabeled reports, for example, those found in news or other news sources.[35]

Julia Ive et al . In their paper, they have connected a various leveled Recurrent Neural Network (RNN) design to the order of presents related on psychological well-being, which is, as far as anyone is concerned, is the primary endeavor of the sort. The capacity to order posts as such is the initial move towards focused mediations, for example by diverting posts requiring mediator consideration. Our model logically manufactures an archive portrayal: it totals vital words into sentence vectors and after that totals vital sentence portrayals to report portrayals, straightforwardly utilized for surmising. We have demonstrated that the characteristic capacity of RNNs to think about contribution to its succession by and large, also, the various leveled structure of this design explicitly can be advantageous for the examination of wellbeing related online content. We watched an act improvement of 4 F-measure (FM) as thought about to Convolutional Neural Network (CNN) arrangements. This improvement is for the most part due to the execution improvement for increasingly uncommon classes (8 FM all things considered). We have likewise demonstrated that the consideration component is skilled to effectively recognize words what's more, sentences of a record important for arrangement choices. We gave a point by point investigation of consideration circulation designs at the sentence level 4Depending on the sort of word and sentence vector estimate, HIERRNN takes around from 30 minutes up to 1 hour to prepare on a 12G GeForce TITAN X NVIDIA GPU. Their results shown as following :

Table 2.3: Multiclass classification evaluation results (we indicate the percentage of posts belonging to a class in the.

				PR/RC/FM		
Theme	%	$\bar{l}_{doc}$	$\bar{l}_{sent}$	CNN	RNN-max	RNN-att
BPD	2%	14	19	0.88 / 0.46 / 0.60	0.84 / 0.52 / 0.64	**0.87 / 0.53 / 0.66**
bipolar	8%	13	18	0.77 / 0.60 / 0.67	0.73 / 0.67 / 0.70	**0.79 / 0.68 / 0.73**
schizophrenia	1%	11	19	0.75 / 0.48 / 0.58	0.78 / 0.60 / 0.67	**0.82 / 0.59 / 0.69**
Anxiety	11%	13	19	0.83 / 0.75 / 0.79	0.79 / 0.81 / 0.80	0.89 / 0.76 / 0.82
depression	37%	16	18	0.70 / 0.77 / 0.73	0.72 / 0.76 / 0.74	0.73 / 0.81 / 0.76
selfharm	3%	11	17	0.70 / 0.58 / 0.64	0.72 / 0.67 / 0.70	**0.76 / 0.67 / 0.71**
suicidewatch	17%	17	17	0.62 / 0.59 / 0.61	0.62 / 0.60 / 0.61	0.65 / 0.61 / 0.63
addiction	0.8%	6	17	0.72 / 0.41 / 0.52	0.76 / 0.41 / 0.53	**0.75 / 0.51 / 0.60**
cripplingalcoholism	8%	7	15	0.68 / 0.76 / 0.72	0.83 / 0.77 / 0.80	**0.73 / 0.86 / 0.79**
Opiates	12%	9	17	0.76 / 0.86 / 0.80	0.82 / 0.89 / 0.85	**0.88 / 0.88 / 0.88**
autism	0.2%	5	18	0.84 / 0.71 / 0.77	0.90 / 0.80 / 0.85	**0.86 / 0.85 / 0.86**
all	100%	11	18	0.72 / 0.71 / 0.72	0.74 / 0.74 / 0.74	**0.76 / 0.76 / 0.76**

furthermore, demonstrated that the start of a report, as well as the last sentence are the most vital. In the meantime, consideration will in general be similarly conveyed between those positions. Later on, we intend to recreate our investigation for different kinds of wellbeing related content, including Electronic Health Records (EHRs), where the succession of occasions can be significantly progressively critical for characterization choices. They additionally plan to examine consideration loads at the word level and contrast those outcomes with the outcomes created utilizing best in class weighting methods, e.g., TFIDF. They similarly plan to deliberately look at execution of various consideration components with the motivation behind finding a hearty arrangement ready to supplant the computationally costly recursion step.

XUEJUN ZHANG et al. They present a deep recurrent convolution neural network model for sentence-level subjectivity classification of microblog short text which is based on Twitter data. They said that "With widespread use of social media, microblog (e.g., Twitter, Facebook) has become one of the important platforms for people's access to information, social communication, interest appealing, self-display, and learning exchanges. Recently, the analysis and mining of the microblog opinion, sentiment, and subjectivity has attracted wide attention from governments, enterprises and researchers, in part because of its potential applications. For example, sentiment analysis system can help organizations understand public emotions towards the events or products associated

with them, obtain public opinions and attitudes in a timely manner. Thus, the microblog sentiment analysis is of great significant for public opinions control, products marketing, public opinion polls, and so on.

Their model concatenates the contextual emotional features, prior polarity features and sentence-level semantic features of microblog text, and uses these feature as input to different layer of the deep recurrent convolution neural network. Their model as shown

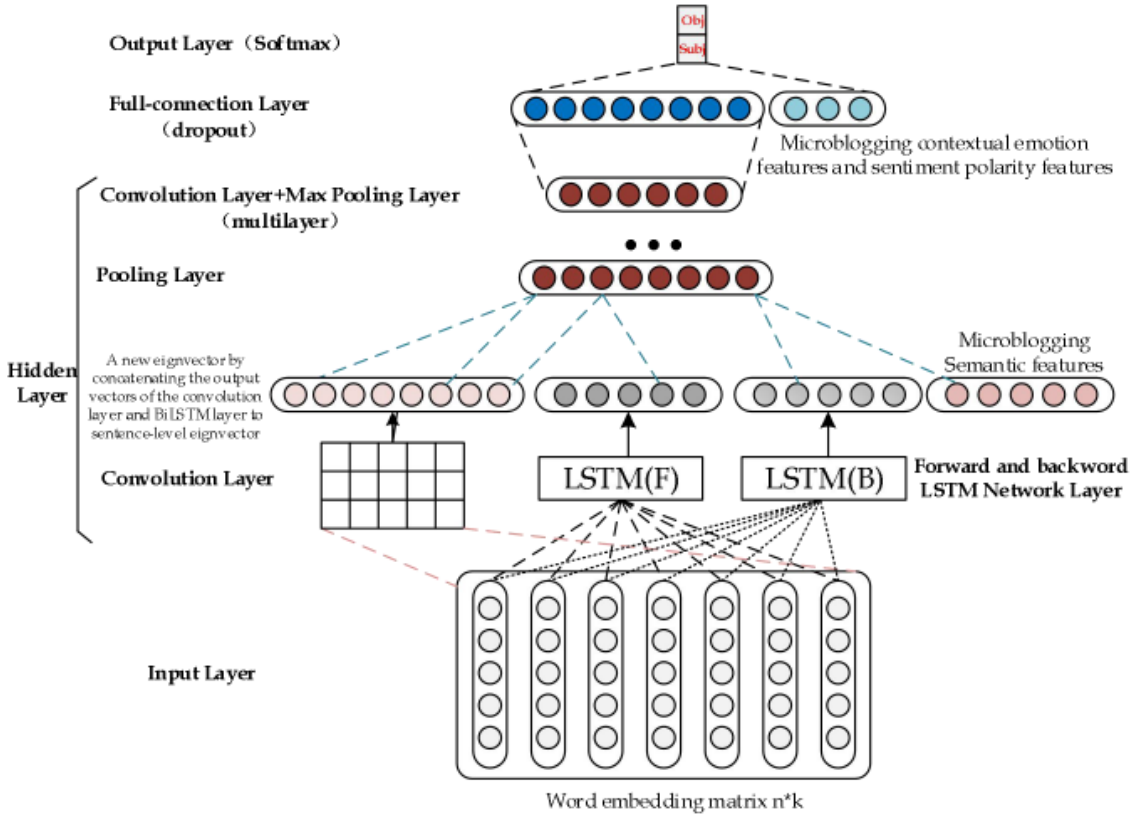

Figure 2.2: Convolution neural network

By combining the bi-direction recurrent structure and the convolution networks, their model captures the features of word contextual semantic relations in text sequences and constructs presentation of the word meaning and relationship of the text simultaneously. They exam the empirical performance of their model on six commonly used" Twitter datasets :

Table 2.4: The Detailed Statistics About Six Datasets

Datasets	# of Tweets	# of Objective	# of Subjective
SE-Twitter	6745	4097	2648
SS-Twitter	4242	1953	2289
SemEval2014	11042	5150	5892
STS-test	498	139	359
SemEval2015-Test	2392	987	1405
SemEval2016	26632	11205	15427

and compare it to other state-of-the-art methods and baseline model. The experimental results demonstrate that their model outperforms the baseline models in both the accuracies of subjectivity classification and F1-Measure values. [36]

Duyu Tang et al. they present neural network models (Conv- GRNN and LSTM-GRNN) for archive level conclusion arrangement. The methodology encodes semantics of sentences and their relations in record portrayal, and is adequately prepared start to finish with administered feeling order goals. They lead broad investigations on four survey datasets with two assessment measurements. Exact outcomes demonstrate that their methodologies achieve state-of-the-art performances on all these datasets. They additionally find that (1) traditional recurrent neural network is amazingly frail in displaying record structure, while including neural doors drastically helps the execution, (2) LSTM performs superior to a multi-sifted CNN in displaying sentence portrayal.

They briefly discuss some future plans. Instructions to successfully form sentence implications to archive which means is a focal issue in regular language preparing. In this work, they create neural models in a successive manner, and encode sentence semantics and their relations naturally without utilizing outside talk examination results. From one point of view, one could cautiously characterize a lot of estimation delicate talk relations ,, for example, "differentiate", "condition", "cause", and so on. Thereafter, connection explicit gated RNN can be created to expressly display semantic organization rules for every connection. In any case, characterizing such a connection conspire is semantic driven and tedious, which they left as future work .[38]

2.4 RNN

Bolan Su et al They recognize texts in images using recurrent neural network . in their proposed method, they convert the word image into sequential column vectors based on Histogram of Oriented Gradient, then they adapted the RNN to classify the sequential feature vectors into the matching word.

Their method shown as following:

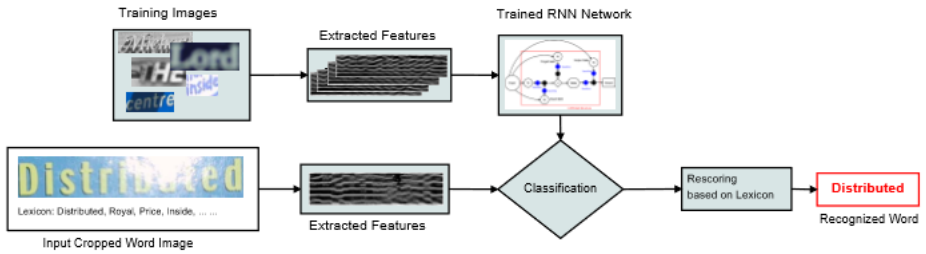

Figure 2.3: RNN to classify the sequential feature vectors

In their system their training data can be viewed as a pair of input column feature and a target word string (C,W) . The objective function of CTC is then defined as follows: Eq (2.3)

$$\mathcal{O} = -\sum_{(\mathbf{C},\mathcal{W})\in\mathcal{S}} \ln p(\mathcal{W}|\mathbf{C}) \tag{2.4}$$

In their experiments on a number of dataset they reported that their method outperforms the state of art techniques significantly and their result provide a good benchmark for the future in this area . as shown as following [32]

Table 2.5: Word Recognition Accuracy on the ICDAR 03 testing dataset, the ICDAR 11 testing dataset and the SVT dataset.

Datsets	ICDAR03 (Full)	ICDAR03 (50)	ICDAR11 (Full)	ICDAR11 (50)	SVT
MRF [5]	0.67	0.69	-	-	-
IR [7]	0.75	0.77	-	-	-
NESP [6]	0.66	-	0.73	-	
PLEX [16]	0.62	0.76	-	-	0.57
HOG + CRF [10]	-	0.82	-	-	0.73
PBS [9]	0.79	0.87	0.83	0.87	0.74
WFST [11]	0.83	-	0.56	-	0.73
CNN [14]	0.84	0.90	-	-	0.70
Proposed	**0.82**	**0.92**	**0.83**	**0.91**	**0.83**

Tang et al. They present neural network models (Conv-GRNN and LSTM-GRNN) for archive level conclusion classification. The methodology encodes semantics of sentences and their relations in archive portrayal, and is successfully prepared start to finish with administered assumption classification targets. They lead broad analyses on four survey datasets with two assessment measurements. Experimental outcomes demonstrate that our methodologies accomplish best in class exhibitions on all these datasets. We likewise find that(1)traditional repetitive neural system is very frail in displaying record piece, while including neural doors drastically supports the execution, (2) LSTM performs superior to a multi-filtered CNN in demonstrating sentence portrayal. They briefly examine some tentative arrangements. Step by step instructions to adequately create sentence implications to archive significance is a focal issue in normal language handling. In this work, we create neural models in a successive manner, and encode sentence and semantics viewpoint their relations consequently without utilizing outside talk investigation results. From one, one could cautiously define a lot of conclusion touchy talk relations, for example, "differentiate", "condition", "cause", and so on. Subsequently, connection specific gated RNN can be created to expressly display

semantic piece rules for every connection .However, defining such a connection plot is etymological driven and tedious, which we leave as future work. From another point of view, one could make record Representation over talk tree structures as opposed to in a consecutive manner. In like manner, Recursive Neural Network and Structured LSTM can be utilized as synthesis calculations. Be that as it may, existing talk structure learning calculations are difficult to scale to enormous survey messages on the web. The most effective method to all the while learn report structure and synthesis work is an intriguing future work.[33]

JuliaIvIn et al. in their exploration they connected a various hierarchical Recurrent Neural Network (RNN) design to the classification of presents related on psychological wellness, which is, as far as anyone is concerned, is the first endeavor of the sort. The capacity to arrange posts thusly is the first venture towards focused intercessions, for example by diverting posts requiring mediator consideration. Their model logically fabricates a record portrayal: it totals imperative words into sentence vectors and afterward totals critical sentence portrayals to archive portrayals, legitimately utilized for surmising. We have demonstrated that the characteristic capacity of RNNs to think about contribution to its succession by and large, and the various leveled structure of this engineering specifically can be beneficial for the examination of wellbeing related online content.

They watched execution improvement of 4F-measure(FM)as contrasted with Convolutional Neural Network (CNN) arrangements. This improvement is predominantly because of the execution improvement previous are classes(8 FM by and large). They have likewise demonstrated that the consideration instrument is competent to efficiently recognize words and sentences of a record significant for classification choices. They gave a point by point investigation of consideration conveyance designs at the sentence level 4 . Contingent upon the sort of word and sentence vector guess, HIERRNN takes around from 30 minutes as long as 1 hour to prepare on a 12G GeForce TITAN X NVIDIA GPU. furthermore, demonstrated that the start of a report, just as the last sentence are the most essential. In the meantime, consideration will in general be similarly appropriated between those positions. Later on, they intend to recreate our

investigation for different sorts of wellbeing related content, including Electronic Health Records (EHRs), where the succession of occasions can be much increasingly essential for classification choices. They likewise plan to explore consideration loads at the word level and contrast those outcomes with the outcomes created utilizing best in class weighting strategies ,e.g., TFIDF. They likewise plan to efficiently contrast execution of various consideration systems and the reason insulting strong arrangement ready to supplant the computationally costly recursion step.[34]

They have compared discriminative and generative LSTM-based content arrangement models in wording of test multifaceted nature and asymptotic mistake rates.

Their data set :

Table 2. 6: Descriptive statistics of datasets used in our experiments. Dani Yogatama et al They

Name	#Train	#Dev	# Test	# Classes
AGNews	115,000	5,000	7,600	4
Sogou	445,000	5,000	60,000	5
Yelp	645,000	5,000	50,000	5
Yelp Binary	555,000	5,000	7,600	2
DBPedia	555,000	5,000	70,000	14
Yahoo	1,395,000	5,000	60,000	10

demonstrated that generative models are better than their discriminative partners in little information routine. Formal portrayal of the speculation conduct of complex neural systems is troublesome, with discoveries from raised issues neglecting. Accordingly, this outcome is amazing for being one space in which speculation conduct of less complex models exchanges to progressively unpredictable models. We additionally examined their properties in the nonstop and zero-shot settings. Our gathering of results demonstrated that generative models are increasingly appropriate in these settings and they had the capacity to acquire equivalent execution to generative models prepared on the full datasets in the standard setting.as shown in their results:

Table 2.7: Summary of results on the full datasets.

Models	AGNews	Sogou	Yelp Bin	Yelp Full	DBPed	Yahoo
Naïve Bayes	90.0	86.3	86.0	51.4	96.0	68.7
Kneser–Ney Bayes	89.3	94.6	81.8	41.7	95.4	69.3
MLP Naïve Bayes	89.9	76.1	73.6	40.4	87.2	60.6
Discriminative LSTM	92.1	94.9	92.6	59.6	98.7	73.7
Generative LSTM–independent comp.	90.7	93.5	90.0	51.9	94.8	70.5
Generative LSTM–shared comp.	90.6	90.3	88.2	52.7	95.4	69.3
bag of words (Zhang et al., 2015)	88.8	92.9	92.2	58.0	96.6	68.9
fastText (Joulin et al., 2016)	92.5	96.8	95.7	63.9	98.6	72.3
char-CNN (Zhang et al., 2015)	87.2	95.1	94.7	62.0	98.3	71.2
char-CRNN (Xiao & Cho, 2016)	91.4	95.2	94.5	61.8	98.6	71.7
very deep CNN (Conneau et al., 2016)	91.3	96.8	95.7	64.7	98.7	73.4

3.EXPERIMENTS EVALUATION

3.1 TEXT CLASSIFICATION

Text classification is a widely studied subject in the information science sector. It has been relevant ever since the origin of digital text documents. Text classification generally involves a multi-step process. Ikonomakis et al**.** [41]. The main goal of the whole classification process is to assign a label to an event by evaluating its textual details, this label corresponds with an Uganda calendar genre. However, it is hard to compare a large bulk of textual details from one event to that of another event without breaking both bulks down into smaller pieces. A text can be separated into a set of words, which we will use as features for classification. The performance of classification generally improves if the features are transformed and filtered, which reduces the size and sparsity of the feature set.

3.1.1 Text Classification Using Convolutional Neural Network (CNN)

CNN is a class of deep, feed-forward artificial neural networks (where connections between nodes do not form a cycle) & use a variation of multilayer perceptron's designed to require minimal preprocessing. These are inspired by animal visual cortex.in figure 3.1 the Architecture of the CNN Model.

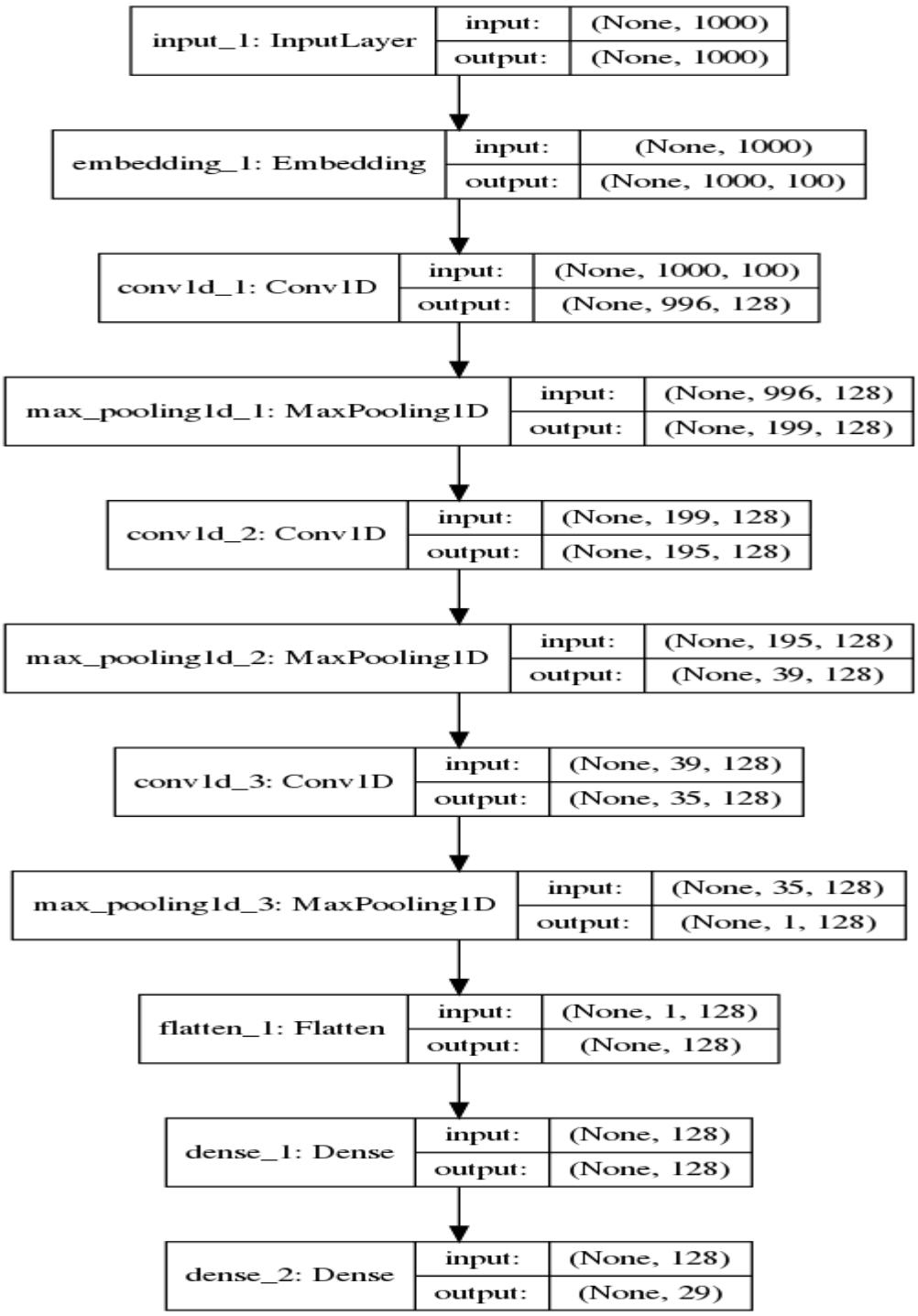

Figure 3.1. Architecture of the CNN Model.

3.1.1.1 Programming

In [0]:
```
import numpy as np
import pandas as pd
import pickle
from collections import defaultdict
import re
from bs4 import BeautifulSoup
import sys
import os
os.environ['KERAS_BACKEND']='theano' # Why theano why not
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import Embedding
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout
from keras.models import Model
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
plt.switch_backend('agg')
%matplotlib inline
```

In [0]:
```
def clean_str(string):
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()
```

In [0]:
```
MAX_SEQUENCE_LENGTH = 1000
MAX_NB_WORDS = 20000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.2
```

In [74]:
```
# reading data
df = pd.read_excel('dataset.xlsx')
df = df.dropna()
df = df.reset_index(drop=True)
print('Shape of dataset ',df.shape)
print(df.columns)
print('No. of unique classes',len(set(df['class'])))
```

```
Shape of dataset  (100, 2)
Index(['message', 'class'], dtype='object')
No. of unique classes 17
```

In [0]:
```
macronum=sorted(set(df['class']))
macro_to_id = dict((note, number) for number, note in enumerate(macronum))

def fun(i):
    return macro_to_id[i]

df['class']=df['class'].apply(fun)
```

In [76]:
```
texts = []
labels = []


for idx in range(data_train.message.shape[0]):
    text = BeautifulSoup(data_train.message[idx])
    texts.append(clean_str(str(text.get_text().encode())))

for idx in data_train['class']:
    labels.append(idx)
```

In [77]:
```
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index
print('Number of Unique Tokens',len(word_index))
```

```
Number of Unique Tokens 932
```

In [78]:
```
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)

labels = to_categorical(np.asarray(labels))
print('Shape of Data Tensor:', data.shape)
print('Shape of Label Tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

```
Shape of Data Tensor: (100, 1000)
Shape of Label Tensor: (100, 17)
```

In [79]:
```
embeddings_index = {}
f = open('glove.6B.100d.txt',encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors in Glove 6B 100d.' % len(embeddings_index))
```

```
Total 400000 word vectors in Glove 6B 100d.
```

In [0]:
```
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,trainable=True)
```

In [80]:
```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_cov1= Conv1D(128, 5, activation='relu')(embedded_sequences)
l_pool1 = MaxPooling1D(5)(l_cov1)
l_cov2 = Conv1D(128, 5, activation='relu')(l_pool1)
l_pool2 = MaxPooling1D(5)(l_cov2)
l_cov3 = Conv1D(128, 5, activation='relu')(l_pool2)
l_pool3 = MaxPooling1D(35)(l_cov3)  # global max pooling
l_flat = Flatten()(l_pool3)
l_dense = Dense(128, activation='relu')(l_flat)
preds = Dense(len(macronum), activation='softmax')(l_dense)

model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

print("Simplified convolutional neural network")
model.summary()
cp=ModelCheckpoint('model_cnn.hdf5',monitor='val_acc',verbose=1,save_best_only=True)
```

```
Simplified convolutional neural network
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 1000)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 100)         93300
_________________________________________________________________
conv1d_10 (Conv1D)           (None, 996, 128)          64128
_________________________________________________________________
max_pooling1d_10 (MaxPooling (None, 199, 128)          0
_________________________________________________________________
conv1d_11 (Conv1D)           (None, 195, 128)          82048
_________________________________________________________________
max_pooling1d_11 (MaxPooling (None, 39, 128)           0
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 35, 128)           82048
_________________________________________________________________
max_pooling1d_12 (MaxPooling (None, 1, 128)            0
_________________________________________________________________
flatten_4 (Flatten)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_8 (Dense)              (None, 17)                2193
=================================================================
Total params: 340,229
Trainable params: 340,229
Non-trainable params: 0
```

```
_________________________________________________________________
```

In [62]:
```
history=model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=15, batch_size=2,callbacks=[cp])
```

```
Train on 80 samples, validate on 20 samples
Epoch 1/15
80/80 [==============================] - 4s 49ms/step - loss: 2.6623 - acc: 0.2500 - val_loss: 2.3584 - val_acc: 0.2500

Epoch 00001: val_acc improved from -inf to 0.25000, saving model to model_cnn.hdf5
Epoch 2/15
80/80 [==============================] - 4s 50ms/step - loss: 1.9873 - acc: 0.4375 - val_loss: 1.8733 - val_acc: 0.4500

Epoch 00002: val_acc improved from 0.25000 to 0.45000, saving model to model_cnn.hdf5
Epoch 3/15
80/80 [==============================] - 4s 50ms/step - loss: 1.2715 - acc: 0.6625 - val_loss: 1.4019 - val_acc: 0.6500

Epoch 00003: val_acc improved from 0.45000 to 0.65000, saving model to model_cnn.hdf5
Epoch 4/15
80/80 [==============================] - 4s 49ms/step - loss: 0.9800 - acc: 0.7250 - val_loss: 1.5273 - val_acc: 0.6500

Epoch 00004: val_acc did not improve from 0.65000
Epoch 5/15
54/80 [===================>.........] - ETA: 1s - loss: 0.7271 - acc: 0.833380/80 [==============================] - 4s 50ms/s
tep - loss: 0.7497 - acc: 0.8250 - val_loss: 1.1414 - val_acc: 0.6500

Epoch 00005: val_acc did not improve from 0.65000
Epoch 6/15
80/80 [==============================] - 4s 49ms/step - loss: 0.6599 - acc: 0.8500 - val_loss: 1.2248 - val_acc: 0.6500

Epoch 00006: val_acc did not improve from 0.65000
Epoch 7/15
80/80 [==============================] - 4s 50ms/step - loss: 0.5452 - acc: 0.8875 - val_loss: 1.1178 - val_acc: 0.7000

Epoch 00007: val_acc improved from 0.65000 to 0.70000, saving model to model_cnn.hdf5
Epoch 8/15
80/80 [==============================] - 4s 49ms/step - loss: 0.4755 - acc: 0.8625 - val_loss: 1.0355 - val_acc: 0.7500

Epoch 00008: val_acc improved from 0.70000 to 0.75000, saving model to model_cnn.hdf5
Epoch 9/15
80/80 [==============================] - 4s 49ms/step - loss: 0.4063 - acc: 0.8750 - val_loss: 1.3590 - val_acc: 0.7000

Epoch 00009: val_acc did not improve from 0.75000
Epoch 10/15
16/80 [=====>........................] - ETA: 2s - loss: 1.1223 - acc: 0.875080/80 [==============================] - 4s 49ms/s
tep - loss: 0.4028 - acc: 0.9250 - val_loss: 1.2219 - val_acc: 0.8000

Epoch 00010: val_acc improved from 0.75000 to 0.80000, saving model to model_cnn.hdf5
Epoch 11/15
80/80 [==============================] - 4s 49ms/step - loss: 0.4737 - acc: 0.8875 - val_loss: 1.1734 - val_acc: 0.8000

Epoch 00011: val_acc did not improve from 0.80000
Epoch 12/15
80/80 [==============================] - 4s 49ms/step - loss: 0.2344 - acc: 0.9250 - val_loss: 1.3924 - val_acc: 0.8000

Epoch 00012: val_acc did not improve from 0.80000
Epoch 13/15
80/80 [==============================] - 4s 50ms/step - loss: 0.1823 - acc: 0.9750 - val_loss: 1.7238 - val_acc: 0.8000

Epoch 00013: val_acc did not improve from 0.80000
Epoch 14/15
80/80 [==============================] - 4s 49ms/step - loss: 0.4431 - acc: 0.9375 - val_loss: 1.4792 - val_acc: 0.8000

Epoch 00014: val_acc did not improve from 0.80000
Epoch 15/15
10/80 [==>...........................] - ETA: 3s - loss: 0.1110 - acc: 0.900080/80 [==============================] - 4s 49ms/s
tep - loss: 0.1814 - acc: 0.9250 - val_loss: 1.5112 - val_acc: 0.8000

Epoch 00015: val_acc did not improve from 0.80000
```

In [63]:
```
fig1 = plt.figure()
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves :CNN',fontsize=16)
fig1.savefig('loss_cnn.png')
plt.show()
```

```
In [63]: fig1 = plt.figure()
         plt.plot(history.history['loss'],'r',linewidth=3.0)
         plt.plot(history.history['val_loss'],'b',linewidth=3.0)
         plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
         plt.xlabel('Epochs ',fontsize=16)
         plt.ylabel('Loss',fontsize=16)
         plt.title('Loss Curves :CNN',fontsize=16)
         fig1.savefig('loss_cnn.png')
         plt.show()
```

```
In [64]: fig2=plt.figure()
         plt.plot(history.history['acc'],'r',linewidth=3.0)
         plt.plot(history.history['val_acc'],'b',linewidth=3.0)
         plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
         plt.xlabel('Epochs ',fontsize=16)
         plt.ylabel('Accuracy',fontsize=16)
         plt.title('Accuracy Curves : CNN',fontsize=16)
         fig2.savefig('accuracy_cnn.png')
         plt.show()
```

```
In [0]: #from keras.utils.vis_utils import plot_model
        #plot_model(model, to_file='cnn_model.png', show_shapes=True, show_layer_names=True)
```

```
In [105]: from PIL import Image
          display(Image.open('cnn_model.png'))
```

Figure 3.2: coding of the CNN

3.1.2 Text Classification Recurrent Neural Network(RAN)

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. This allows it to exhibit dynamic temporal behavior for a time sequence.

Build a tf.keras . Sequential model and start with an embedding layer. An embedding layer stores one vector to per word. When called, it converts the sequences of word indices sequences of vectors. These vectors are trainable. After training (on enough data), words with similar meanings often have similar vectors.

This index-lookup is much more efficient than the equivalent operation of passing a one-hot encoded vector through a tf.keras.layers. Dense layer. A recurrent neural network (RNN) processes sequence input by iterating through the elements. RNNs pass the

outputs from one timestep to their input—and then to the next.The tf.keras.layers.Bidirectional wrapper can also be used with an RNN layer. This propagates the input forward and backwards through the RNN layer and then concatenates the output. This helps the RNN to learn long range dependencies.

In figure 3.4 the model of RNN .

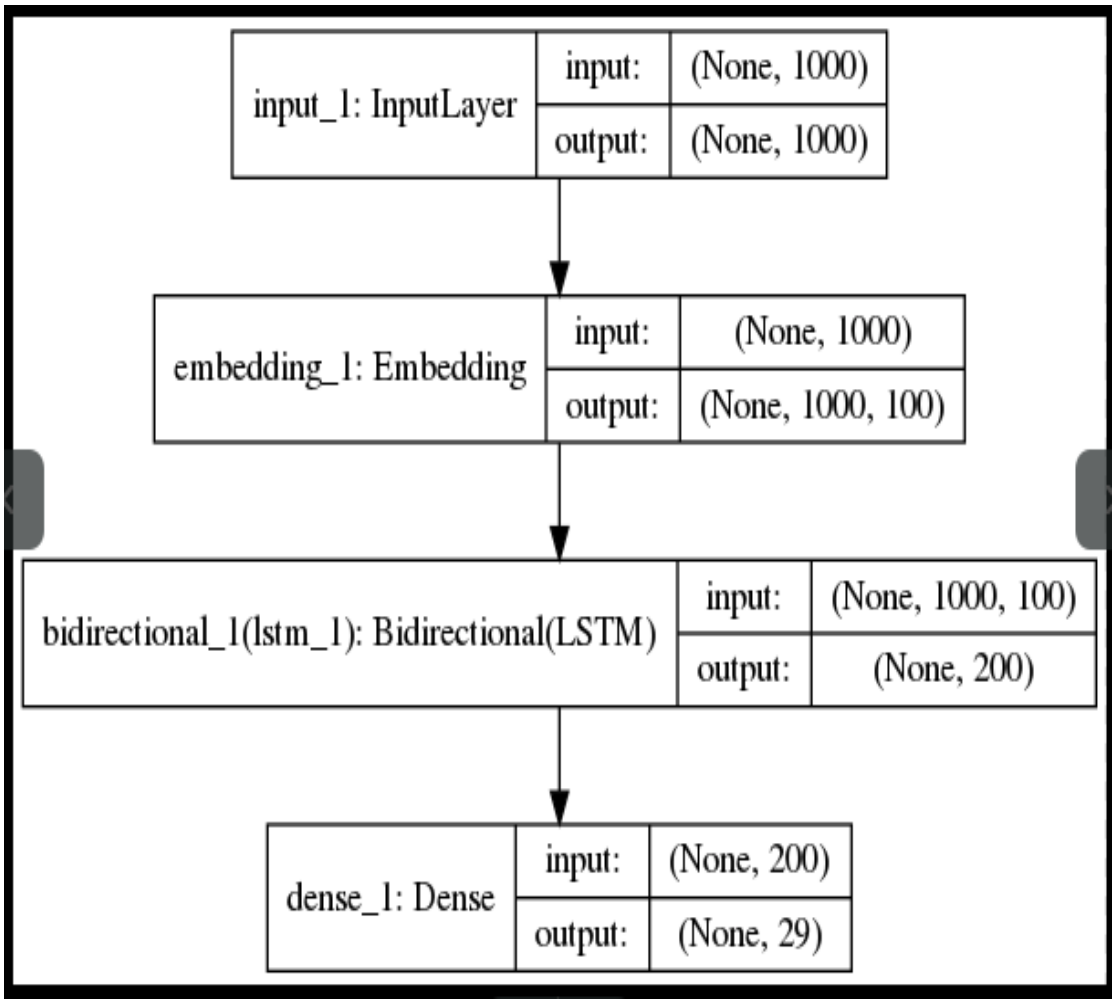

Figure 3.3 Architecture of the RNN Model.

3.1.3 Programming of RNN

In [0]:
```
import numpy as np
import pandas as pd
import pickle
from collections import defaultdict
import re
from bs4 import BeautifulSoup
import sys
import os
os.environ['KERAS_BACKEND']='theano'
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import Embedding
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout, LSTM, GRU, Bidirectional
from keras.models import Model
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
plt.switch_backend('agg')
from keras import backend as K
from keras.engine.topology import Layer, InputSpec
from keras import initializers
%matplotlib inline
```

In [0]:
```
def clean_str(string):
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()
```

In [0]:
```
MAX_SEQUENCE_LENGTH = 1000
MAX_NB_WORDS = 20000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.2
```

In [19]:
```
# reading data
df = pd.read_excel('dataset.xlsx')
df = df.dropna()
df = df.reset_index(drop=True)
print('Shape of dataset ',df.shape)
print(df.columns)
print('No. of unique classes',len(set(df['class'])))
```

```
Shape of dataset  (100, 2)
Index(['message', 'class'], dtype='object')
No. of unique classes 17
```

In [0]:
```
macronum=sorted(set(df['class']))
macro_to_id = dict((note, number) for number, note in enumerate(macronum))

def fun(i):
    return macro_to_id[i]

df['class']=df['class'].apply(fun)
```

In [21]:
```
texts = []
labels = []


for idx in range(df.message.shape[0]):
    text = BeautifulSoup(df.message[idx])
    texts.append(clean_str(str(text.get_text().encode())))

for idx in df['class']:
    labels.append(idx)
```

In [22]:
```
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)

word_index = tokenizer.word_index

print('Number of Unique Tokens',len(word_index))
```

```
Number of Unique Tokens 932
```

In [23]:
```
data = pad_sequences(sequences, maxlen=MAX_SEQUENCE_LENGTH)


labels = to_categorical(np.asarray(labels))
print('Shape of Data Tensor:', data.shape)
print('Shape of Label Tensor:', labels.shape)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])

x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

```
Shape of Data Tensor: (100, 1000)
Shape of Label Tensor: (100, 17)
```

In [24]:
```
embeddings_index = {}
f = open('glove.6B.100d.txt',encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors in Glove 6B 100d.' % len(embeddings_index))
```

```
Total 400000 word vectors in Glove 6B 100d.
```

In [0]:
```
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

In [0]:
```
embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=True)
```

In [27]:
```
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sequence_input)
l_lstm = Bidirectional(LSTM(100))(embedded_sequences)
preds = Dense(len(macronum), activation='softmax')(l_lstm)
model = Model(sequence_input, preds)
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

print("Bidirectional LSTM")
model.summary()
```

```
Bidirectional LSTM
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1000)              0
_________________________________________________________________
embedding_1 (Embedding)      (None, 1000, 100)         93300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200)               160800
_________________________________________________________________
dense_1 (Dense)              (None, 17)                3417
=================================================================
Total params: 257,517
Trainable params: 257,517
Non-trainable params: 0
_________________________________________________________________
```

In [18]:
```
cp=ModelCheckpoint('model_rnn.hdf5',monitor='val_acc',verbose=1,save_best_only=True)
history=model.fit(x_train, y_train, validation_data=(x_val, y_val),epochs=15, batch_size=2,callbacks=[cp])
```

```
Train on 80 samples, validate on 20 samples
Epoch 1/15
80/80 [==============================] - 168s 2s/step - loss: 2.4410 - acc: 0.2125 - val_loss: 1.9150 - val_acc: 0.4500

Epoch 00001: val_acc improved from -inf to 0.45000, saving model to model_rnn.hdf5
Epoch 2/15
80/80 [==============================] - 147s 2s/step - loss: 2.0052 - acc: 0.4750 - val_loss: 1.5308 - val_acc: 0.5500

Epoch 00002: val_acc improved from 0.45000 to 0.55000, saving model to model_rnn.hdf5
Epoch 3/15
80/80 [==============================] - 140s 2s/step - loss: 1.5564 - acc: 0.5750 - val_loss: 1.3702 - val_acc: 0.6500

Epoch 00003: val_acc improved from 0.55000 to 0.65000, saving model to model_rnn.hdf5
Epoch 4/15
80/80 [==============================] - 99s 1s/step - loss: 1.3451 - acc: 0.6125 - val_loss: 1.1095 - val_acc: 0.6500
```

```
Epoch 00005: val_acc did not improve from 0.65000
Epoch 6/15
80/80 [==============================] - 80s 1s/step - loss: 0.9999 - acc: 0.7250 - val_loss: 0.7506 - val_acc: 0.9000

Epoch 00006: val_acc improved from 0.65000 to 0.90000, saving model to model_rnn.hdf5
Epoch 7/15
80/80 [==============================] - 100s 1s/step - loss: 0.7787 - acc: 0.7875 - val_loss: 0.7046 - val_acc: 0.9500

Epoch 00007: val_acc improved from 0.90000 to 0.95000, saving model to model_rnn.hdf5
Epoch 8/15
80/80 [==============================] - 96s 1s/step - loss: 0.7224 - acc: 0.8000 - val_loss: 0.5418 - val_acc: 0.9500

Epoch 00008: val_acc did not improve from 0.95000
Epoch 9/15
80/80 [==============================] - 136s 2s/step - loss: 0.5805 - acc: 0.8375 - val_loss: 0.5848 - val_acc: 0.9500

Epoch 00009: val_acc did not improve from 0.95000
Epoch 10/15
 4/80 [>.............................] - ETA: 2:09 - loss: 0.9108 - acc: 0.750080/80 [==============================] - 127s 2
s/step - loss: 0.6635 - acc: 0.8250 - val_loss: 0.4901 - val_acc: 0.9500

Epoch 00010: val_acc did not improve from 0.95000
Epoch 11/15
80/80 [==============================] - 103s 1s/step - loss: 0.5786 - acc: 0.8500 - val_loss: 0.4940 - val_acc: 0.9500

Epoch 00011: val_acc did not improve from 0.95000
Epoch 12/15
80/80 [==============================] - 102s 1s/step - loss: 0.4891 - acc: 0.9000 - val_loss: 0.6753 - val_acc: 0.9000

Epoch 00012: val_acc did not improve from 0.95000
Epoch 13/15
80/80 [==============================] - 113s 1s/step - loss: 0.4893 - acc: 0.8875 - val_loss: 0.4899 - val_acc: 0.9500

Epoch 00013: val_acc did not improve from 0.95000
Epoch 14/15
80/80 [==============================] - 104s 1s/step - loss: 0.4824 - acc: 0.8750 - val_loss: 1.1208 - val_acc: 0.6500


Epoch 00014: val_acc did not improve from 0.95000
Epoch 15/15
16/80 [=====>........................] - ETA: 1:26 - loss: 0.6188 - acc: 0.875080/80 [==============================] - 111s 1
s/step - loss: 0.4562 - acc: 0.8875 - val_loss: 0.4913 - val_acc: 0.9500

Epoch 00015: val_acc did not improve from 0.95000
```

In [20]:
```
fig1 = plt.figure()
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves :RNN',fontsize=16)
fig1.savefig('loss_rnn.png')
plt.show()
```

In [21]:
```
fig2=plt.figure()
plt.plot(history.history['acc'],'r',linewidth=3.0)
plt.plot(history.history['val_acc'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves : RNN',fontsize=16)
fig2.savefig('accuracy_rnn.png')
plt.show()
```

```
In [0]: #from keras.utils.vis_utils import plot_model
        #plot_model(model, to_file='rnn_model.png', show_shapes=True, show_layer_names=True)

In [28]: from PIL import Image
         display(Image.open('rnn_model.png'))
```

Figure 3.4: Coding of the RNN

3.1.4 Text Classification Using Hierarchical Attention Network(HAN)

Contrary to most text classification implementations, a Hierarchical Attention Network (HAN) also considers the hierarchical structure of documents (document - sentences - words) and includes an attention mechanism that is able to find the most important words and sentences in a document while taking the context into consideration. Other methods only return importance weights resulting from previous words. Summarizing, HAN tries to find a solution for these problems that previous works did not consider:

- Not every word in a sentence and every sentence in a document are equally important to understand the main message of a document.
- The changing meaning of a word depending on the context needs to be taken into consideration. For example, the meaning of the word “pretty” can change depending on the way it is used: “The bouquet of flowers is pretty” vs. “The food is pretty bad”.

In this way, HAN performs better in predicting the class of a given document.

To start from scratch, have a look at this example:

pork belly = delicious . || scallops? || I don't even like scallops, and these were a-m-a-z-i-n-g . || fun and tasty cocktails. || next time I in Phoenix, I will go back here. || Highly recommend.

Here, we have a review from yelp that consists of five sentences. The highlighted sentences in red deliver stronger meaning compared to the others, and inside, the words *delicious* and *amazing* contribute the most in attributing the positive attitude contained in this review. This example reproduces our aforementioned statement about HAN, which we also intuitively know: not all parts of a document are equally relevant to gain the essential meaning from it. The model of HAN in figure 3.7 .

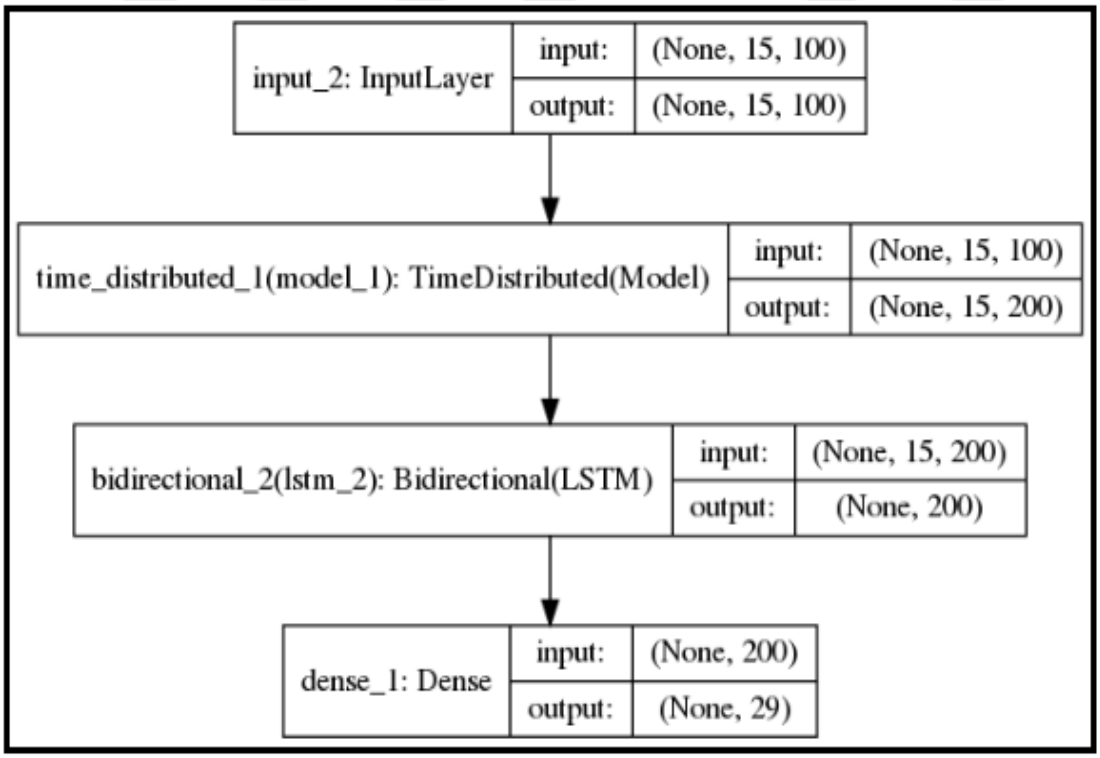

Figure 3.5 Architecture of the HAN Model.

3.1.4.1 Programming of the HAN

In [0]:
```
import numpy as np
import pandas as pd
import pickle
from collections import defaultdict
import re
from bs4 import BeautifulSoup
import sys
import os
os.environ['KERAS_BACKEND']='theano'
from keras.preprocessing.text import Tokenizer,text_to_word_sequence
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import Embedding
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout, LSTM, GRU, Bidirectional, TimeDistributed
from keras.models import Model
from keras.callbacks import ModelCheckpoint
import matplotlib.pyplot as plt
plt.switch_backend('agg')
from keras import backend as K
from keras.engine.topology import Layer, InputSpec
from keras import initializers
%matplotlib inline
```

In [0]:
```
def clean_str(string):
    string = re.sub(r"\\", "", string)
    string = re.sub(r"\'", "", string)
    string = re.sub(r"\"", "", string)
    return string.strip().lower()
```

In [0]:
```
MAX_SENT_LENGTH = 100
MAX_SENTS = 15
MAX_NB_WORDS = 20000
EMBEDDING_DIM = 100
VALIDATION_SPLIT = 0.2
```

In [42]:
```
# reading data
df = pd.read_excel('dataset.xlsx')
df = df.dropna()
df = df.reset_index(drop=True)
print('Shape of dataset ',df.shape)
print(df.columns)
print('No. of unique classes',len(set(df['class'])))
```

```
Shape of dataset  (100, 2)
Index(['message', 'class'], dtype='object')
No. of unique classes 17
```

In [0]:
```
import nltk
from nltk import tokenize

reviews = []
labels = []
texts = []
```

In [0]:
```
macronum=sorted(set(df['class']))
macro_to_id = dict((note, number) for number, note in enumerate(macronum))
```

In [0]:
```
def fun(i):
    return macro_to_id[i]

df['class']=df['class'].apply(fun)
```

In [46]:
```
for i in range(df.message.shape[0]):
    text = BeautifulSoup(df.message[i])
    text=clean_str(str(text.get_text().encode()).lower())
    texts.append(text)
    sentences = tokenize.sent_tokenize(text)
    reviews.append(sentences)


for i in df['class']:
    labels.append(i)
```

In [0]:
```
tokenizer = Tokenizer(num_words=MAX_NB_WORDS)
tokenizer.fit_on_texts(texts)

data = np.zeros((len(texts), MAX_SENTS, MAX_SENT_LENGTH), dtype='int32')

for i, sentences in enumerate(reviews):
    for j, sent in enumerate(sentences):
        if j< MAX_SENTS:
            wordTokens = text_to_word_sequence(sent)
            k=0
            for _, word in enumerate(wordTokens):
                if k<MAX_SENT_LENGTH and tokenizer.word_index[word]<MAX_NB_WORDS:
                    data[i,j,k] = tokenizer.word_index[word]
                    k=k+1
```

In [48]:
```
word_index = tokenizer.word_index
print('No. of %s unique tokens.' % len(word_index))
```

```
No. of 932 unique tokens.
```

In [49]:
```
labels = to_categorical(np.asarray(labels))
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)


indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
labels = labels[indices]
nb_validation_samples = int(VALIDATION_SPLIT * data.shape[0])
```

```
Shape of data tensor: (100, 15, 100)
Shape of label tensor: (100, 17)
```

In [0]:
```
x_train = data[:-nb_validation_samples]
y_train = labels[:-nb_validation_samples]
x_val = data[-nb_validation_samples:]
y_val = labels[-nb_validation_samples:]
```

In [51]:
```
embeddings_index = {}
f = open('glove.6B.100d.txt',encoding='utf8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()

print('Total %s word vectors.' % len(embeddings_index))
```

```
Total 400000 word vectors.
```

In [0]:
```
embedding_matrix = np.random.random((len(word_index) + 1, EMBEDDING_DIM))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

embedding_layer = Embedding(len(word_index) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SENT_LENGTH,
                            trainable=True)
```

In [53]:
```
sentence_input = Input(shape=(MAX_SENT_LENGTH,), dtype='int32')
embedded_sequences = embedding_layer(sentence_input)
l_lstm = Bidirectional(LSTM(100))(embedded_sequences)
sentEncoder = Model(sentence_input, l_lstm)

review_input = Input(shape=(MAX_SENTS,MAX_SENT_LENGTH), dtype='int32')
review_encoder = TimeDistributed(sentEncoder)(review_input)
l_lstm_sent = Bidirectional(LSTM(100))(review_encoder)
preds = Dense(len(macronum), activation='softmax')(l_lstm_sent)
model = Model(review_input, preds)

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['acc'])

print("Hierachical LSTM")
model.summary()
```

```
Hierachical LSTM
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 15, 100)           0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 15, 200)           254100
_________________________________________________________________
bidirectional_2 (Bidirection (None, 200)               240800
_________________________________________________________________
dense_1 (Dense)              (None, 17)                3417
=================================================================
Total params: 498,317
Trainable params: 498,317
Non-trainable params: 0
_________________________________________________________________
```

In [54]:
```
cp=ModelCheckpoint('model_han_.hdf5',monitor='val_acc',verbose=1,save_best_only=True)
history=model.fit(x_train, y_train, validation_data=(x_val, y_val),
          epochs=15, batch_size=2,callbacks=[cp])
```

```
Train on 80 samples, validate on 20 samples
Epoch 1/15
80/80 [==============================] - 28s 351ms/step - loss: 2.5511 - acc: 0.1875 - val_loss: 2.9483 - val_acc: 0.2500

Epoch 00001: val_acc improved from -inf to 0.25000, saving model to model_han_.hdf5
Epoch 2/15
80/80 [==============================] - 28s 349ms/step - loss: 2.1806 - acc: 0.2750 - val_loss: 2.8415 - val_acc: 0.3500

Epoch 00002: val_acc improved from 0.25000 to 0.35000, saving model to model_han_.hdf5
Epoch 3/15
80/80 [==============================] - 28s 350ms/step - loss: 1.6876 - acc: 0.5875 - val_loss: 3.9033 - val_acc: 0.2000

Epoch 00003: val_acc did not improve from 0.35000
Epoch 4/15
80/80 [==============================] - 28s 351ms/step - loss: 1.3917 - acc: 0.6500 - val_loss: 2.5706 - val_acc: 0.4500

Epoch 00004: val_acc improved from 0.35000 to 0.45000, saving model to model_han_.hdf5
Epoch 5/15
50/80 [=================>............] - ETA: 10s - loss: 0.9921 - acc: 0.700080/80 [==============================] - 28s 350m
s/step - loss: 1.1223 - acc: 0.6625 - val_loss: 2.4962 - val_acc: 0.5000

Epoch 00005: val_acc improved from 0.45000 to 0.50000, saving model to model_han_.hdf5
Epoch 6/15
80/80 [==============================] - 28s 351ms/step - loss: 0.9457 - acc: 0.7125 - val_loss: 2.4805 - val_acc: 0.4500

Epoch 00006: val_acc did not improve from 0.50000
Epoch 7/15
80/80 [==============================] - 34s 420ms/step - loss: 0.8393 - acc: 0.7625 - val_loss: 1.9323 - val_acc: 0.7000

Epoch 00007: val_acc improved from 0.50000 to 0.70000, saving model to model_han_.hdf5
Epoch 8/15
80/80 [==============================] - 34s 431ms/step - loss: 0.6905 - acc: 0.8375 - val_loss: 2.0766 - val_acc: 0.7000

Epoch 00008: val_acc did not improve from 0.70000
Epoch 9/15
80/80 [==============================] - 33s 413ms/step - loss: 0.6308 - acc: 0.8250 - val_loss: 2.1420 - val_acc: 0.6500
```

```
Epoch 00009: val_acc did not improve from 0.70000
Epoch 10/15
80/80 [==============================] - 38s 474ms/step - loss: 0.5324 - acc: 0.8500 - val_loss: 2.0284 - val_acc: 0.7000

Epoch 00010: val_acc did not improve from 0.70000
Epoch 11/15
80/80 [==============================] - 41s 507ms/step - loss: 0.4603 - acc: 0.8750 - val_loss: 2.2320 - val_acc: 0.6500

Epoch 00011: val_acc did not improve from 0.70000
Epoch 12/15
80/80 [==============================] - 46s 573ms/step - loss: 0.4084 - acc: 0.8875 - val_loss: 2.2356 - val_acc: 0.6500

Epoch 00012: val_acc did not improve from 0.70000
Epoch 13/15
80/80 [==============================] - 53s 668ms/step - loss: 0.3975 - acc: 0.8875 - val_loss: 2.4935 - val_acc: 0.6500

Epoch 00013: val_acc did not improve from 0.70000
Epoch 14/15
80/80 [==============================] - 60s 745ms/step - loss: 0.3789 - acc: 0.8750 - val_loss: 2.1979 - val_acc: 0.7000

Epoch 00014: val_acc did not improve from 0.70000
Epoch 15/15
14/80 [====>.........................] - ETA: 57s - loss: 0.1149 - acc: 1.000080/80 [==============================] - 68s 855m
s/step - loss: 0.2603 - acc: 0.9250 - val_loss: 2.3647 - val_acc: 0.7000

Epoch 00015: val_acc did not improve from 0.70000
```

In [55]:
```
fig1 = plt.figure()
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves :HAN',fontsize=16)
fig1.savefig('loss_han.png')
plt.show()
```

In [56]:
```
fig2=plt.figure()
plt.plot(history.history['acc'],'r',linewidth=3.0)
plt.plot(history.history['val_acc'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves : HAN',fontsize=16)
fig2.savefig('accuracy_han.png')
plt.show()
```

In [0]:
```
#from keras.utils.vis_utils import plot_model
#plot_model(model, to_file='han_model.png', show_shapes=True, show_layer_names=True)
```

In [65]:
```
from PIL import Image
display(Image.open('han_model.png'))
```

Figure 3.6: Coding of the HAN

4. THE RESULTS

4.1 THE CNN

results shown in figure 4.2 and 4.3. Train on 80 samples, validate on 20 samples

Figure 4.1: The Loss curve

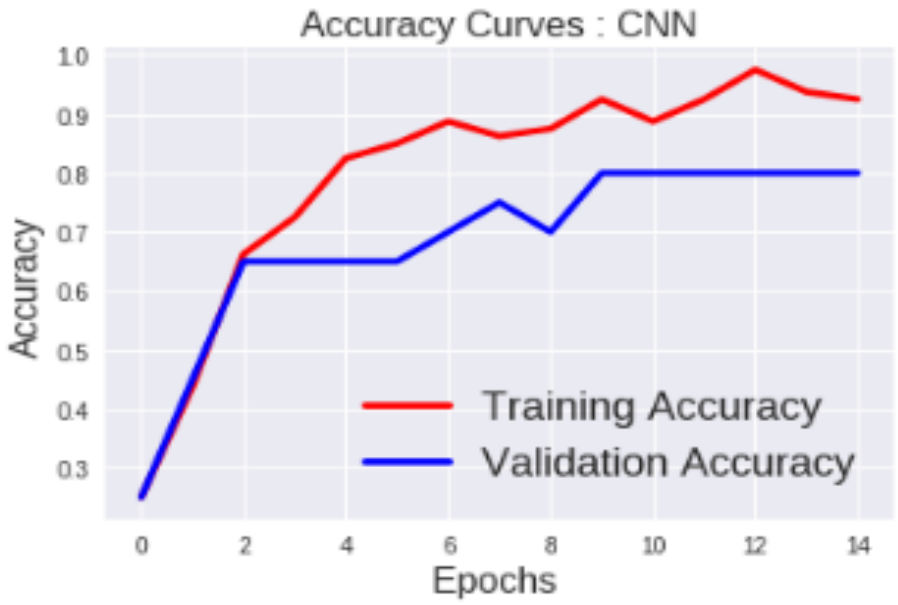

Figure 4.2 Accuracy Curve

4.2 THE RESULTS FROM (RNN)

The results from the programming of RNN shown in figure Train on 80 samples, validate on 20 samples

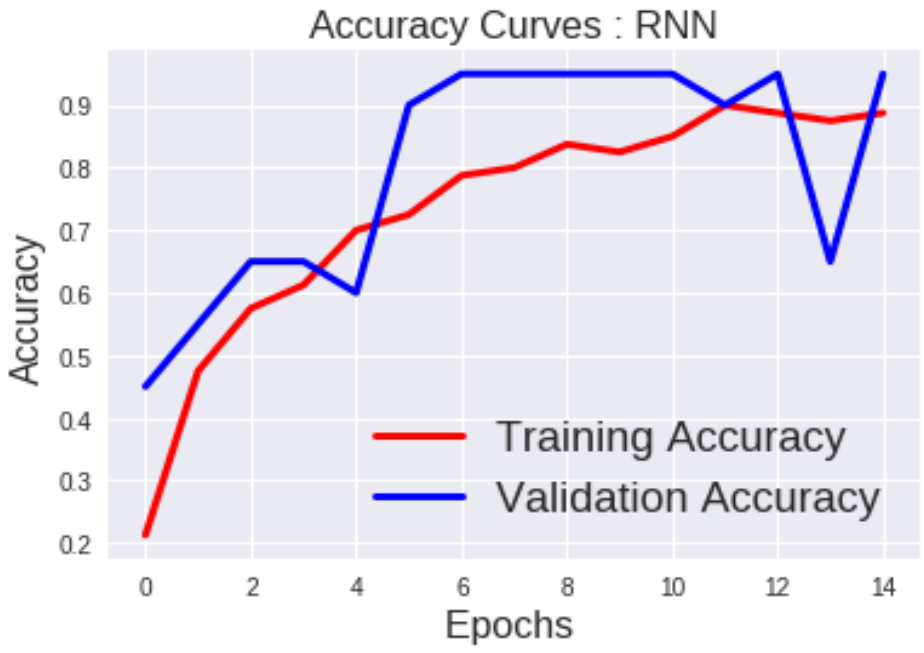

Figure 4.3 The Results from RNN Accuracy Curves

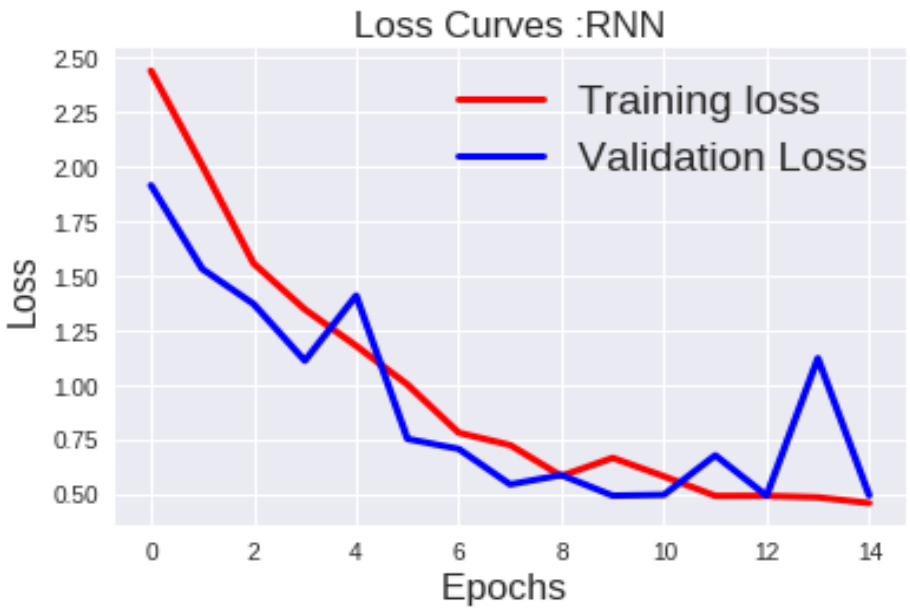

Figure 4.5 The Results of RNN for Loss Curves

4.3 THE RESULTS FROM (HAN)

The results shown in figure 4.6and 4.7 Train on 80 samples, validate on 20 samples

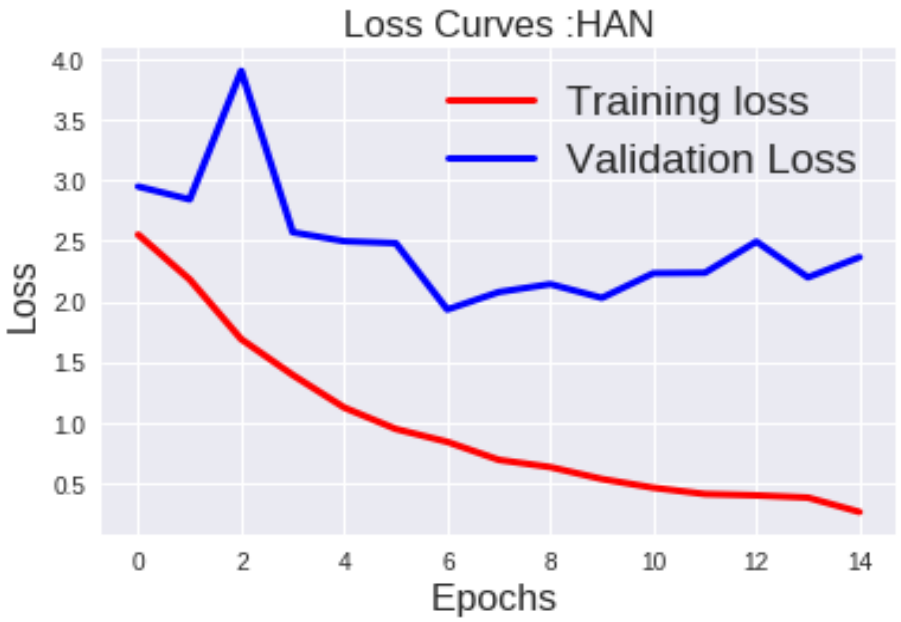

Figure 4.6: Loss curves for HAN

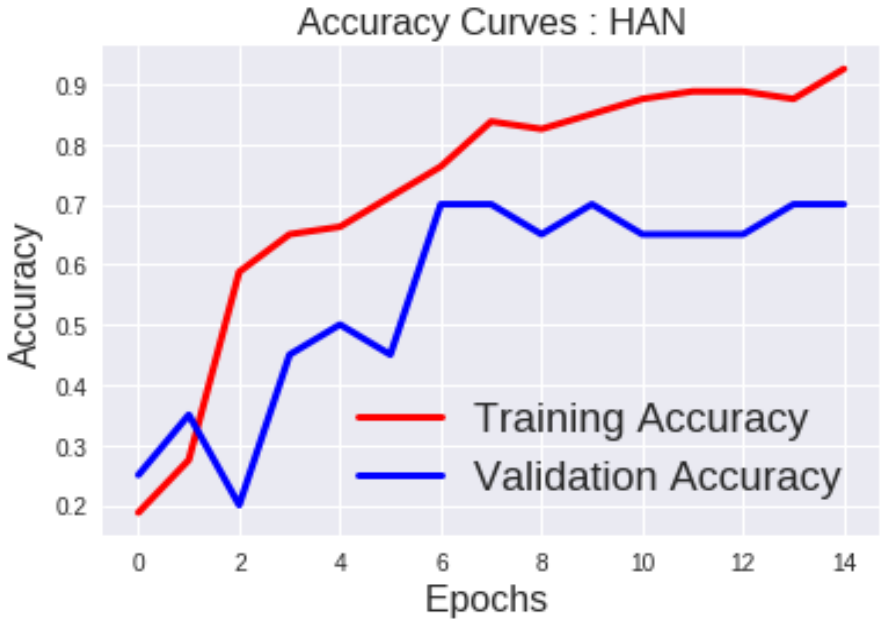

Figure 4.7: Results For Accuracy For HAN

4.4 FINAL RESULTS

Our results based on Training 80 samples, validate on 20 samples the table 4.1 show the average of accuracy and loss for each example.

Table 4.1: Final Results

Algorithms	**Accuracy**	**Valid accuracy**	**Loss**	**Valid loss**
CNN	**0.79**	**1.26**	**0.8234**	**1.43**
HAN	**0.72**	**0.55**	**0.9215**	**2.4628**
RNN	**0.72**	**0.80**	**1.0129**	**0.9118**

5. THE CONCLUSION

5.1 CONCLUSION

In this thesis we have presented the NLP concepts and their approaches in the field of text classification. we have shown the important of NLP's algorithms and their contributions in the field of text classification through the related works .in our experiments we have used python programming language because it's more efficient in this field and easy to use. we have made a comparison for three text classification algorithms. form the result that we got from experiments field, we can figure out that the CNN algorithm is the best one in term of accuracy and valid accuracy and it has the minimum loss among others, HAN algorithm has the highest value in term of Valid loss. RNN has the highest value in term of loss.

REFERENCE

[1] G. Benin and J. Wang, “Swarm intelligence in cellular robotic systems,” in Robots and Biological Systems: Towards a New Bionics? vol. 102 of NATO ASI Series, pp. 703–712, Springer, Berlin, Germany, 1993.

[2] X. Yao, “Evolving artificial neural networks,” Proceedings of the IEEE, vol. 87, no. 9, pp. 1423–1447, 1999.

[3] E. Alba and R. Martí, Metaheuristic Procedures for Training Neural Networks, Operations Research/Computer Science Interfaces Series, Springer, New York, NY, USA, 2006.

[4] J. Yu, L. Xi, and S. Wang, “An improved particle swarm optimization for evolving feedforward artificial neural networks,” Neural Processing Letters, vol. 26, no. 3, pp. 217–231, 2007.

[5] M. Conforth and Y. Meng, “Toward evolving neural networks using bio-inspired algorithms,” in IC-AI, H. R. Arabnia and Y. Mun, Eds., pp. 413–419, CSREA Press, 2008.

[6] Y. Da and G. Xiurun, “An improved PSO-based ANN with simulated annealing technique,” Neurocomputing, vol. 63, pp. 527–533, 2005.

[7] X. Yao and Y. Liu, “A new evolutionary system for evolving artificial neural networks,” IEEE Transactions on Neural Networks, vol. 8, no. 3, pp. 694–713, 1997.

[8] D. Rivero and D. Periscal, “Evolving graphs for ann development and simplification,” in Encyclopedia of Artificial Intelligence, J. R. Rabuñal, J. Dorado, and A. Pazos, Eds., pp. 618–624, IGI Global, 2009.

[9] H. M. Abdul-Kader, “Neural networks training based on differential evolution algorithm compared with other architectures for weather forecasting34,” International Journal of Computer Science and Network Security, vol. 9, no. 3, pp. 92–99, 2009.

[10] K. K. Kuok, S. Harun, and S. M. Shamsuddin, “Particle swarm optimization feedforward neural network for modeling runoff,” International Journal of Environmental Science and Technology, vol. 7, no. 1, pp. 67–78, 2010.

[11] B. A. Garro, H. Sossa, and R. A. Vázquez, “Back-propagation vs particle swarm optimization algorithm: which algorithm is better to adjust the synaptic weights of a feed-forward ANN?” International Journal of Artificial Intelligence, vol. 7, no. 11, pp. 208–218, 2011.

[12] B. Garro, H. Sossa, and R. Vazquez, “Evolving neural networks: a comparison between differential evolution and particle swarm optimization,” in Advances in Swarm Intelligence, Y. Tan, Y. Shi, Y. Chai, and G. Wang, Eds., vol. 6728 of Lecture Notes in Computer Science, pp. 447–454, Springer, Berlin, Germany, 2011.

[13] B. Garro, H. Sossa, and R. Vazquez, “Design of artificial neural networks using differential evolution algorithm,” in Neural Information Processing. Models and Applications, K. Wong, B. Mendis, and A. Bouzerdoum, Eds., vol. 6444 of Lecture Notes in Computer Science, pp. 201–208, Springer, Berlin, Germany, 2010.

[14] B. A. Garro, H. Sossa, and R. A. Vazquez, “Artificial neural network synthesis by means of artificial bee colony (abc) algorithm,” in Proceedings of the IEEE Congress on Evolutionary Computation (CEC '11), pp. 331–338, IEEE, New Orleans, La, USA, June 2011.

[15] Conneau, A.; Schwenk, H.; Barrault, L.; and LeCun, Y. 2016. Very deep convolutional networks for natural language processing. CoRR abs/1606.01781.

[16] Conneau, A.; Schwenk, H.; Barrault, L.; and LeCun, Y. 2016. Very deep convolutional networks for natural language processing. CoRR abs/1606.01781.

[17] Kim, Y. 2014. Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL, 1746–1751.

[18] Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural language processing (almost) from scratch. J. Mach. Learn. Res. 12:2493–2537.

[19] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In Burges, C. J. C.;

[20] Kalchbrenner, N.; Grefenstette, E.; and Blunsom, P. 2014. A convolutional neural network for modelling sentences. In Proceedings of the 52nd Annual Meeting of the Association

[21] for Computational Linguistics, 655665. Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks.

[22] Bengio, Y.; Ducharme, R.; Vincent, P.; and Janvin, C. 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3:1137–1155.

[23] Severyn, A., and Moschitti, A. 2015. Twitter sentiment analysis with deep convolutional neural networks. In Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR ’15, 959–962. New York, NY, USA: ACM.

[24] Zhang, Y., andWallace, B. C. 2015. A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification. CoRR abs/1510.03820.

[25] Kim, Y.; Jernite, Y.; Sontag, D.; and Rush, A. M. 2016. Character-aware neural language models. In AAAI, 2741– 2749. AAAI Press.

[26] Radford, A.; J´ozefowicz, R.; and Sutskever, I. 2017. Learning to generate reviews and discovering sentiment. CoRR abs/1704.01444.

[27] 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. In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 1631–1642. Seattle, Washington, USA: Association for Computational Linguistics.

[28] Dhingra, B.; Liu, H.; Cohen, W. W.; and Salakhutdinov, R. 2016. Gated-attention readers for text comprehension. CoRR abs/1606.01549.

[29] Miyamoto, Y., and Cho, K. 2016. Gated word-character recurrent language model. In EMNLP.

[30] Yang, Z.; Dhingra, B.; Yuan, Y.; Hu, J.; Cohen, W. W.; and Salakhutdinov, R. 2017. Words or characters? fine-grained gating for reading comprehension. In ICLR.

[31] O Ormandjieva, I. Hussain, and L. Kosseim, “Toward a Text classification System for the Quality Assessment of Software Requirements Written in Natural Language”, In: Fourth international workshop on Software quality assurance: in conjunction with the 6th ESEC/FSE joint meeting SOQUA '07, ACM press, 2007.

[32] Y. Ko, S. Park, S., J. Seo, and S. Choi, “Using classification techniques for informal requirements in the requirements analysis-supporting system”, Information and Software Technology, 2007, 49(11-12), pp 1128-1140.

[33] Su, Bolan, and Shijian Lu. "Accurate scene text recognition based on recurrent neural network." Asian Conference on Computer Vision. Springer, Cham, 2014.

[34] Tang, Duyu, Bing Qin, and Ting Liu. "Document modeling with gated recurrent neural network for sentiment classification." Proceedings of the 2015 conference on empirical methods in natural language processing. 2015.

[35] Ive, Julia, et al. "Hierarchical neural model with attention mechanisms for the classification of social media text related to mental health." Proceedings of the Fifth Workshop on Computational Linguistics and Clinical Psychology: From Keyboard to Clinic. 2018.

[36] Kowsari, Kamran, et al. "Hdltex: Hierarchical deep learning for text classification." 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, 2017.

[37] Zhang, Xuejun, et al. "Exploring Deep Recurrent Convolution Neural Networks for Subjectivity Classification." IEEE Access7 (2019): 347-357.

[38] Tang, Duyu, Bing Qin, and Ting Liu. "Document modeling with gated recurrent neural network for sentiment classification." Proceedings of the 2015 conference on empirical methods in natural language processing. 2015.

[39] Conneau, Alexis, et al. "Very deep convolutional networks for text classification." arXiv preprint arXiv:1606.01781 (2016).

[40] He, Tong, et al. "Text-attentional convolutional neural network for scene text detection." IEEE transactions on image processing 25.6 (2016): 2529-2541.

[41] M. Ikonomakis, S. Kotsiantis, and V. Tampakas. “Text classification using machine learning techniques.” In: WSEAS Transactions on Computers 4.8 (2005), pp. 966–974.