Comparison analysis of CNNs and RNN-LSTMs for Social Reviews

K.S.N.S Gopala Krishna[1], B.L.S Suraj[2], M. Trupthi[3]
[1,2]Student, [3]Assistant Professor, Department of Information Technology, Chaitanya Bharathi Institute of Technology, Hyderabad (India)

Abstract-**In this paper we present simple and efficient deep learning models for sentiment analysis and text classification. Sentiment analysis is an important task in natural language processing and has many applications in the real world. A typical sentiment analysis task can be described as a process of classifying opinions expressed in a text as positive, negative or neutral. This paper employs two models; one model is built using Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs) and the other with Convolution Neural Networks. In our first model we used RNN-LSTMs model to capture the semantic and syntactic relationships between words of a sentence with help of word2vec. In the second model One-dimensional Convolution Neural networks were used to learn structure in paragraphs of words and the techniques invariance to the specific position of features. The IMDB movie reviews dataset set is being used for sentiment analysis in both the models and the results are compared. Both the models yielded excellent results.**

Index Terms— **Deep Learning, Artificial Neural Networks, Convolutional Neural networks, Recurrent Neural Networks, Long Short Term Memory (LSTMs), Word2vec.**

I. Introduction

The power of Deep learning can be demonstrated by recurrent neural networks (RNNs) and Convolution Neural Networks (CNNs) which can be viewed as a top class of dynamic models that is used to generate sequences in multiple domains such as machine translations, speech recognition, music, generating captions for input images and determining emotional tone behind a piece of text (Sentiment classification of text).

The Long Short Term Memory networks (LSTM) is a part of RNN architecture which are capable of storing and accessing information more efficiently than regular RNNs. Sentiment Classification of texts is a classic topic of Deep Learning. Text classification plays a major role in multiple real-time applications for example spam filtering, document search, web search and predicting the polarity of a sentence.

CNNs are majorly responsible for breakthroughs in Image Classification related tasks and are the center of many computer vision systems in the present day. CNNs can also be applied to Natural language processing (NLP) tasks to obtain interesting results. For sentence classification tasks, instead of image pixels, the input sentences are represented as a matrix. These vectors are word embedding like word2vec.

There are different methods to achieve the task for text classification, Machine learning algorithms like Logistical Regression, SVM etc., are at the center of these applications. These algorithms need the text input to be represented as vector.

Another traditional method for fixed-length vector representations is bag-of-words; a text in from of a sentence is represented as multiset of its words. The frequency of each word can be used as a feature for training of the classifier. The multiple downsides for this method are ignoring the grammar and order of words that results in different sentences having the same vector representations.

Another widespread method is n-grams. The limitation of the n-grams model is that it takes into account the word order in short sentence, but it suffers from data sparsity and high dimensionality. These traditional and simple methods have limitations for many NLP tasks. In this paper we propose new methodologies such as CNNs and RNN-LSTMs to overcome the drawbacks of traditional methods and algorithms, improve the accuracy of sentiment analysis and finally compare the results of the two best deep learning methods.

II. CONCEPTS

A. *Artificial Neural Networks(ANN)*

Artificial neural networks try to simulate the human brain. ANN devices are loosely modeled on the neural structure of the human brain but on a much smaller scale. Where a large ANN has hundreds or thousands of processor units, a human brain has billions of neurons with a corresponding increase in magnitude of their overall interaction and emergent behavior. A neural network consists of neurons which are simple processing units and there are directed, weighted connections between these neurons. A neural network has a number of neurons processing in parallel manner and arranged in layers. Layers consist of a number of interconnected nodes which contain an activation function. The patterns from the input layer are processed using the weighted connections which help with the communication with the hidden layer. The output from one layer is fed to the next and so on. The output is given by the final layer. For a neuron j, the propagation function receives the outputs of other neurons and transforms them in consideration of the weights into the network input that can be further processed by the activation function. Every neuron operates in two modes: training and using mode.

B. *Convolution Neural Networks(CNN)*

The term convolution can be defined as a sliding window function applied to a Matrix. Convolution Neural Networks (CNN) are basically multiple layers of convolution with non-linear activation functions like ReLU and tanh applied to the outputs. CNNs make use of convolutions over input layer to compute the output. This results in local connections; where in individual region of the input is connected to a neuron present in the output. Every single layer applies different filters and combines the results[4]. CNNs are designed to honor the spatial structures present in the image data at the same time to be robust to orientation and positioning of learned objects in the given scene. The different types of layers present in the CNN architecture: Convolutional Layer**,** Pooling Layer**,** and Fully-Connected Layer**.** These layers are stacked on each other to form a CNN model. Figure 1 shows the structure of Convolution Neural Networks. [2]

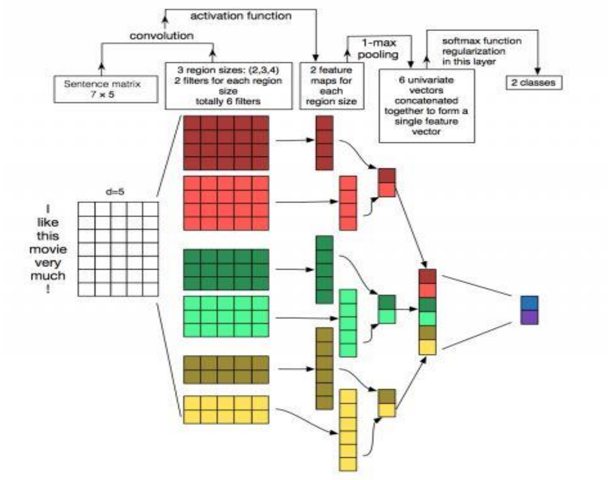

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Fig 1: CNN Structure

C. *Recurrent Neural Networks(RNN)*

The Major difference between a traditional Neural network and an RNN is that, the traditional neural networks cannot use its reasoning about previous events in the process to inform the later neurons or events. Traditional neural networks start thinking from scratch again. RNNs address this issue, by using loops in their network; thus making information persist.

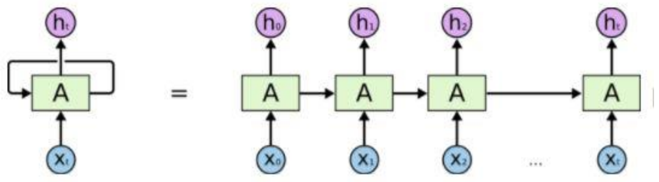


Fig.2: RNN Structure

In the above figure, the RNN A takes some input x_t and outputs some value h_t[4]. A loop allows information to be passed from one step of the network to the next. A RNN can be considered as multiple copies of same network, each network passing some information to its successor. In RNNs, each word in an input sequence will be associated with a specific time step. As a result, the number of time steps will be equal to the max sequence length. Associated with each time step is also a new component called a hidden state vector h_t which constitutes the output at each iteration . From a high level, h_t seeks to encapsulate and summarize all of the information that was seen in the previous time steps. Similarly x_t is a vector that encapsulates all the information of a specific input word; the hidden state is a function of both the current word vector and the hidden state vector at the previous time step. The sigma indicates that the sum of the two terms will be put through an activation function. The activation function is given by:

$$h_t=\sigma(W^{H*}h_{t-1}+W*^{X}x_t) \qquad (1)$$

Where, Wx is weight matrix to be multiplied with input x_t and is variable. W^H is recurrent weight matrix which is multiplied with hidden state vector of previous step. W^H is a constant weight matrix[3]. These weight matrixes are adjusted and updated through an optimization process called Backpropagation through time. Sigma indicates that the sum of two terms would be passed through an activation function,usually sigmoid or Tanh.

D. *Long Short Term Memory Networks(LSTMs)*

Long Short Term Memory networks(LSTMs) are a special kind of RNN that are capable of learning the long term dependencies. LSTMs are modules that you can place inside a RNN which are explicitly be used to avoid the long term dependency problem. The long term dependency problem can be described as a situation where the gap between the relevant information and the point where the network is needed becomes very large and as a result the RNNs become incapable to learn to connect the previous information[4]. The compuatation in LSTMs can be broken down into 4 components, an input gate which detemines the amount of emphasis to be put on each gate, a forget gate determines the unnecessary information, an output gate which will determine the final hidden vector state based on Intermediate states and a new memory container.

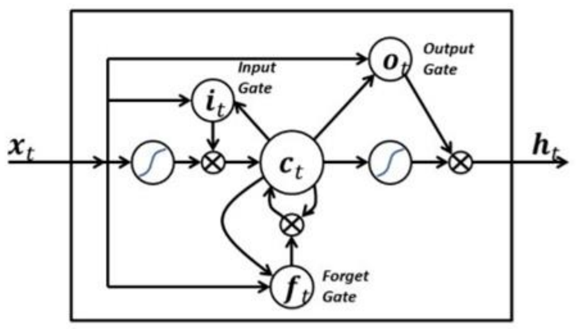


Fig. 3: LSTM Structure

Each gate as described above takes two inputs, x_t and h_{t-1} . These inputs perform computation to obtain intermediate states. Later the information gates are fed into different pipelines and the output h_t (as shown in the figure) is obtained.

E. Word2vec

There are different forms of data which can be taken as input; for example logistic regression model takes in quantifiable features, reinforced learning models take reward signals as input[3]. Similarly for the models, we will be making use of Word Vectors instead of strings which enable us to perform common operations such as backpropagation and dot products. These word vectors are created in such a way that they represent the word and its context, meaning and semantics. Word Embedding is the term used for vector representation of a word. In order to perform the task of creating word embedding, we use Word2vec model. This model picks the words in the sentence with same context semantics and same connotations and places then in the same vector space[3]. This model takes a large dataset of sentences and outputs vectors for each word exclusively in the corpus. The output of the Word2vec model is called as an embedded matrix.

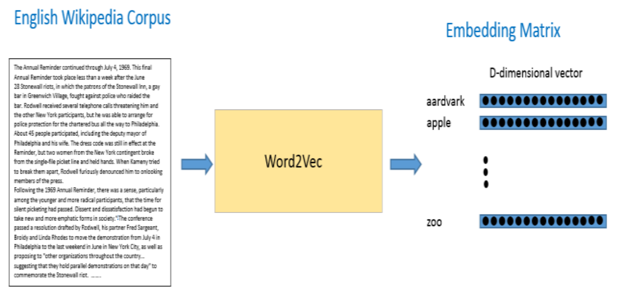


Fig. 4: Word2vec for English Wikipedia Corpus

From figure 4 by taking the entire Wikipedia Corpus, and applying Word2Vec, we converted it into an Embedding matrix. We train the Word2vec model by taking each single sentence in our dataset and slide a window of fixed size over it, trying to calculate and predict the center word of the window. This model uses loss function and optimization procedures to generate vectors for each word.

III. Methodology Using RNN with LSTM

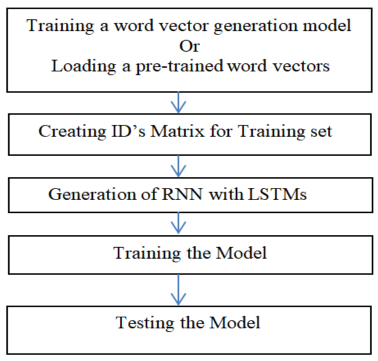

Training a word vector generation model
Or
Loading a pre-trained word vectors
Creating ID's Matrix for Training set
Generation of RNN with LSTMs
Training the Model
Testing the Model

Fig 5: Flowchart representation of RNN and LSTM Model.

In the first step, we load a pre-trained model Word2vec which was trained by Google over Google-news dataset. Word2vec contains 3 million word vectors, with a dimensionality of 300.

We import two data structures, a python list and an embedded matrix which holds all the word vector values. After we have our vectors we take an input sentence and construct its vector representation. We make use of Tensorflow's embedded lookup function to generate word vectors.

The next step of this model is creating the ids matrix for our dataset. The data used for training the RNN-LSTM model is the IMDB movie review dataset. It contains 25,000 movie reviews with 12,500 negative and positive reviews respectively. Matplot library is used to visualize the data in a Histogram format. From figure 6, it can be inferred that average number of words in a file is 250. In order to convert our data into ids matrix we load the movie training set and integer the dataset to obtain 25000*250 matrix (25000 number of files and average number of words per file 250).

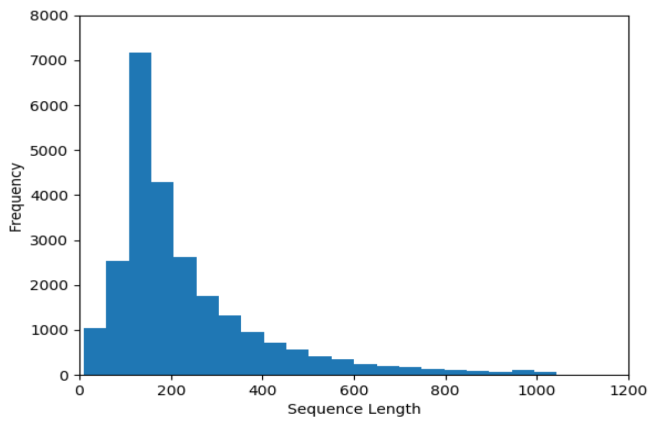


Fig 6: Visualizing data with a histogram.

On creation of the ids Matrix we begin working with RNNs. Some hyper parameters, such as number of LSTM units, number of output classes, batch size, and number of training iterations are defined. In this model, two placeholders are specified, one for inputs into the network and other for labels. The Label placeholders represent values set [1, 0] or [0, 1] based on the positive or negative training example respectively. Every row present in the integer input placeholder represents the integer representation of each training example that we have included in our batch.

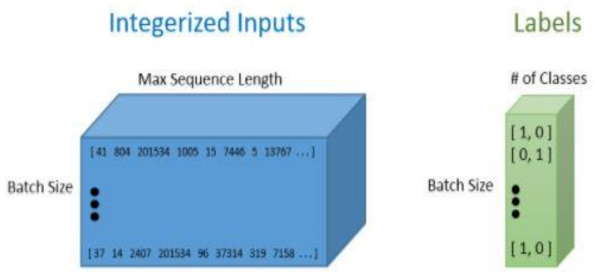


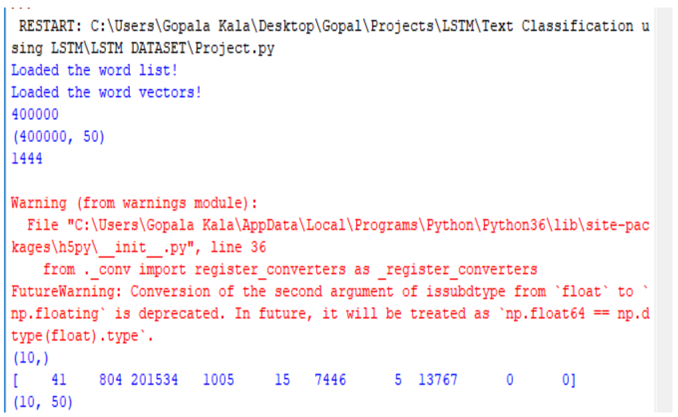
Fig 7: Integreizied inputs and their labels.

Word vectors are obtained using various functions present in tensor flow which return a 3 Dimensional tensor of dimensionality batch size by max sequence length by word vector dimensions. On visualizing this 3-D tensor by considering each data point in the integer input tensor as a result of a D dimensional vector that it refers to(As shown in the figure 6).

On obtaining in our required format we feed it as an input into the LSTM network. We make use of tf.nn.rnn_cell.Basic LSTM Cell function to specify number of LSTM units we require to build our model. The number of LSTMs in our model is a hyper parameter which we tune to obtain the optimum number. The LSTM cell in dropout layer is covered to prevent our network from overfitting.[3]

The next step is to feed the 3-D tensor completely with input data and the LSTM cell to a function known as tf.nn.dynamic_rnn. We use the function to unroll the entire network and generate a pathway for the data to flow through the RNN model. Multiple LSTM cells are being stacked on top of each other such that the final hidden state vector of the first LSTM feeds to the second LSTM in the model and so on. This helps the model to improve information long term dependency but also leads more parameters into our model and hence probably increases the training time need for additional training examples and a chance of overfitting.

The first output of the dynamic RNN is the last hidden state vector. This vector is reshaped and multiplied with final weight matrix and a bias term to obtain our final output values. Next, we define correct accuracy and prediction metrics to track our network model performance by defining appropriate number hyperparameters. The correct prediction formulation task is done by looking at the index of the maximum value of the two output values, and then we check if it matches with our training labels. We then defined a standard cross entropy loss with a softmax layer put on top of the final prediction values. Adam was used as the optimizer with the default learning rate of .001.



```
...
 RESTART: C:\Users\Gopala Kala\Desktop\Gopal\Projects\LSTM\Text Classification u
sing LSTM\LSTM DATASET\Project.py
Loaded the word list!
Loaded the word vectors!
400000
(400000, 50)
1444

Warning (from warnings module):
  File "C:\Users\Gopala Kala\AppData\Local\Programs\Python\Python36\lib\site-pac
kages\h5py\__init__.py", line 36
    from ._conv import register_converters as _register_converters
FutureWarning: Conversion of the second argument of issubdtype from `float` to `
np.floating` is deprecated. In future, it will be treated as `np.float64 == np.d
type(float).type`.
(10,)
[    41    804 201534   1005     15   7446      5  13767      0      0]
(10, 50)
```

Fig 8: Output when the word list and ID's Matrix has been loaded.

Hyperparameter Tuning:

Choosing the right values for hyperparameters(like number of LSTM units, wordvector size etc.) is a crucial part of training deep neural networks effectively[3]. With RNNs and LSTMs in particular, some other important factors include the number of LSTM units and the size of the word vectors.

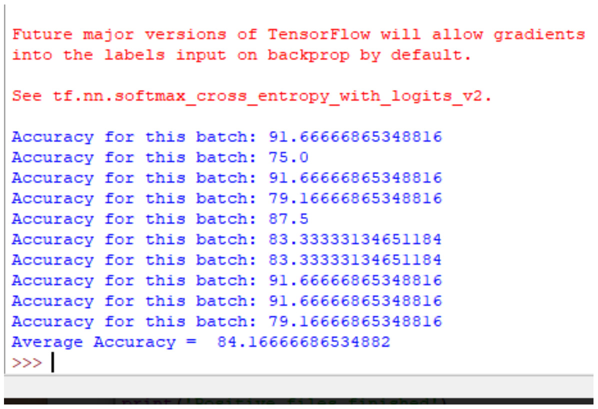
Number of LSTM units: This parameter is dependent on the average length of the input texts. Greater number of LSTM units allows the model to retain more information and improves the ability of more to express. On the other hand, the model takes longer to train and becomes computationally expensive. The number of LSTM units used for our project is 64.

Word Vector Size: The typical dimensions for a word vector range from 50 to 300. The larger size vector can encapsulate more information about the word but it also increases the computation expense of the model.

Optimizer: The optimizer we use in our model is Adam, which is popular due to its adaptive learning property. The optimal learning rates differ with the choice of the optimizer.

Training and testing the model:

The first step of training involves loading in batch of reviews and the labels associated with them. We implement the *run* function which has two arguments; first argument is fetches which defines the value we're interested in computing and the optimizer to be computed since it is the component that minimizes the loss function. The second is the data structure where we provide all of our placeholders. This loop is repeated for a certain number of iterations. The movie reviews from the test set are loaded into the model. These are the movie reviews that the model has not been trained on. The model computes the reviews by predicting the sentiment and generates results for each test batch.



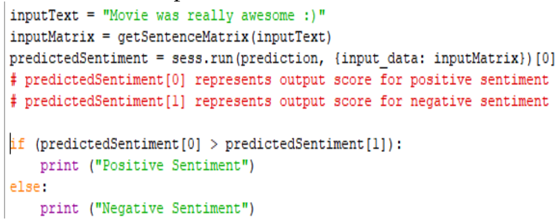
```
Future major versions of TensorFlow will allow gradients
into the labels input on backprop by default.

See tf.nn.softmax_cross_entropy_with_logits_v2.

Accuracy for this batch: 91.66666865348816
Accuracy for this batch: 75.0
Accuracy for this batch: 91.66666865348816
Accuracy for this batch: 79.16666865348816
Accuracy for this batch: 87.5
Accuracy for this batch: 83.33333134651184
Accuracy for this batch: 83.33333134651184
Accuracy for this batch: 91.66666865348816
Accuracy for this batch: 91.66666865348816
Accuracy for this batch: 79.16666865348816
Average Accuracy =  84.16666686534882
>>> 
```

Fig 9: Output showing the accuracy of the model

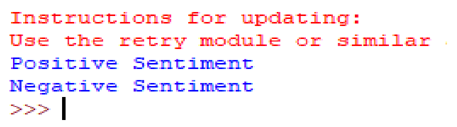
We also tested our model by taking user input, and printing what the model predicts.



```
inputText = "Movie was really awesome :)"
inputMatrix = getSentenceMatrix(inputText)
predictedSentiment = sess.run(prediction, {input_data: inputMatrix})[0]
# predictedSentiment[0] represents output score for positive sentiment
# predictedSentiment[1] represents output score for negative sentiment

if (predictedSentiment[0] > predictedSentiment[1]):
    print ("Positive Sentiment")
else:
    print ("Negative Sentiment")
```

Fig 10: User input.



```
Instructions for updating:
Use the retry module or similar
Positive Sentiment
Negative Sentiment
>>> 
```

Fig 11: Output of fig 10.

IV. Methodology using One Dimensional Convolution Neural Networks

In this model, we make use of Keras module and various methods present in it to build our Convolutional Neural Network model [2]. Keras supports one-dimensional convolutions and pooling by the Conv1D and MaxPooling1D classes respectively. Figure 11 shows an overview of entire process using CNN. In the first step, we import the packages and classes necessary to build the model and initialize the random number generator to constant.

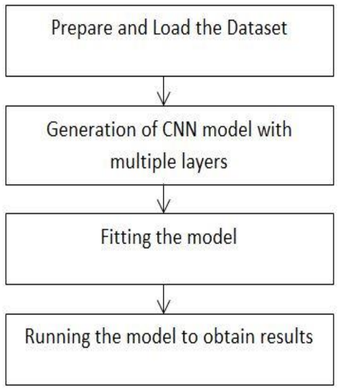
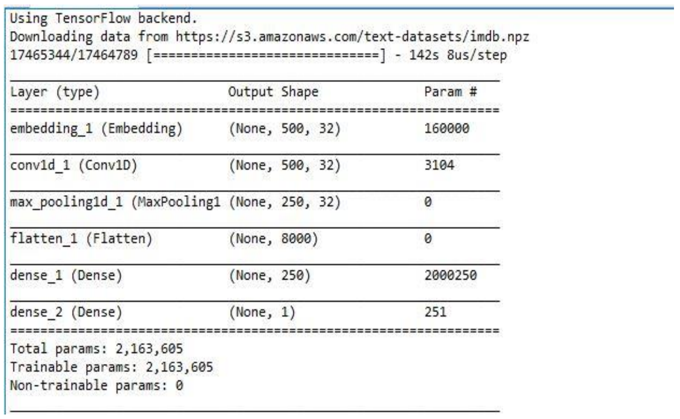

Prepare and Load the Dataset
Generation of CNN model with multiple layers
Fitting the model
Running the model to obtain results

Fig 12: Flowchart for the CNN model

The next step is to load and prepare our IMDB dataset. After this we define the Convolutional Neural Network model. After embedding the input layer, a Conv1D layer is inserted. The convolutional layer in our model has 32 feature maps and takes input of embedded word representations with 3 vector elements of the word embedding at a single time [4]. The convolutional layer present in this paper is followed by a 1D max pooling layer with a stride and length of 2 which halves the size of the feature maps from the convolutional layer.

We fit the model using the model.fit() function present in Keras. On running the model, we obtained a summary of the network structure. The convolutional layer preserves the dimensionality of embedding input layer present in the CNN model of 32-dimensional input with a maximum of 500 words. The pooling layer is used to compress this representation by halving it. The accuracy with which the model classifies the polarities present in the sentence is obtained.

By running the steps as described in the flowchart, a network structure is given as output as shown below.



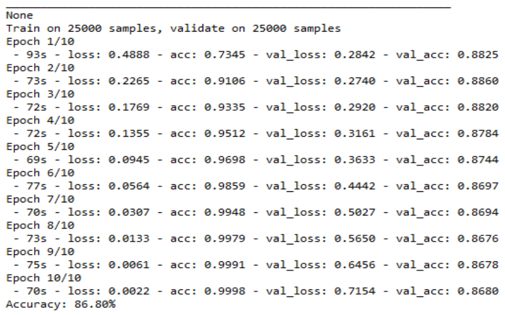
```
Using TensorFlow backend.
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 142s 8us/step

Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 500, 32)           3104
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 250, 32)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 8000)              0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               2000250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 2,163,605
Trainable params: 2,163,605
Non-trainable params: 0
_________________________________________________________________
```

Fig 13: Network Structure of CNN.

From fig 11 it can be inferred that the dimensionality of embedding input layer of 32-dimensional input is preserved with a maximum of 500 words.

The accuracy achieved by running our CNN model for 10 iterations is 86.80%.



```
_________________________________________________________________
None
Train on 25000 samples, validate on 25000 samples
Epoch 1/10
 - 93s - loss: 0.4888 - acc: 0.7345 - val_loss: 0.2842 - val_acc: 0.8825
Epoch 2/10
 - 73s - loss: 0.2265 - acc: 0.9106 - val_loss: 0.2740 - val_acc: 0.8860
Epoch 3/10
 - 72s - loss: 0.1769 - acc: 0.9335 - val_loss: 0.2920 - val_acc: 0.8820
Epoch 4/10
 - 72s - loss: 0.1355 - acc: 0.9512 - val_loss: 0.3161 - val_acc: 0.8784
Epoch 5/10
 - 69s - loss: 0.0945 - acc: 0.9698 - val_loss: 0.3633 - val_acc: 0.8744
Epoch 6/10
 - 77s - loss: 0.0564 - acc: 0.9859 - val_loss: 0.4442 - val_acc: 0.8697
Epoch 7/10
 - 70s - loss: 0.0307 - acc: 0.9948 - val_loss: 0.5027 - val_acc: 0.8694
Epoch 8/10
 - 73s - loss: 0.0133 - acc: 0.9979 - val_loss: 0.5650 - val_acc: 0.8676
Epoch 9/10
 - 75s - loss: 0.0061 - acc: 0.9991 - val_loss: 0.6456 - val_acc: 0.8678
Epoch 10/10
 - 70s - loss: 0.0022 - acc: 0.9998 - val_loss: 0.7154 - val_acc: 0.8680
Accuracy: 86.80%
```

Fig 14: Accuracy of CNN model.

V. COMPARISON BETWEEN THE PERFORMANCE OF CNN AND LSTM-RNN

TABLE 1: Comparing performances of CNN and LSTM-RNN

Attribute	CNN	LSTM-RNN
Accuracy for 10 epochs	86.80%	84.16
Accuracy for 2 epochs	88.56	80.16%
Time taken for training dataset	Within 2mins	Approximately 5Mins
Optimizer used	Adams	Adams
Data Set used	IMDB	IMDB

VI. CONCLUSION

Deep learning is a thriving field that is seeing lots of different advancement. With help of Recurrent Neural networks, long short-term memory networks, convolution networks and backpropagation to name a few this fields is quickly turning explosive and successful. The models presented in the paper are proof of the strength of deep learning in general. We looked at the different components involved in the whole pipeline and the process of creating, training and testing an RNN-LSTM model and a CNN model to classify movie reviews and compare the results outputted by these deep learning models. As we can see from the comparison table that for the same number of epochs RNNs are slow and fickle to train, they take more time for training when compare to one-dimensional CNNs. Also for text classification, feature detection is important like searching for angry words etc. One-dimensional Convolution Neural Networks gave better result as we can also see from the accuracy of the model.

REFERENCES

[1] Kim, Y., Convolutional neural networks for sentence classification. In Proceedings of the 2014 Conference on Empirical Methods in Natual Language Processing (EMNLP). pages 1746-1751, Doha, Qatar, October 2014, Association for Computational linguisitcs.

[2] https://machinelearningmastery.com/

[3] https://www.oreilly.com/learning

[4] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[5] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition," ArXiv e-prints, Feb. 2014.

[6] M. Hermans and B. Schrauwen, "Training and analysing deep recurrent neural networks," in Advances in Neural Information Processing Systems, 2013, pp. 190–198.

[7] R. J. Williams and J. Peng, "An efficient gradient-based algorithm for online training of recurrent network trajectories," Neural Computation, vol. 2, pp. 490–501, 1990.

[8] Mikolov, Tomas, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. "Recurrent neural network based language model." In Interspeech, vol. 2, p. 3. 2010.

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

[10] Zhang, X., J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. in Advances in Neural Information Processing Systems. 2015.