Text Classification using Deep Neural Networks

Problem Statement

Text classification is one of the most important Natural Language Processing & Supervised Machine Learning tasks for understanding audience sentiment based on their reaction to a product or event. It helps to understand the tone of a piece of written text automatically by training a machine. This is useful for businesses looking to understand how their product is received by the public on social media or reviews on Amazon, Yelp, etc. We have chosen the Amazon Reviews Sentiment Dataset for analyzing customer reactions on Amazon products.

Dataset Description

This reviews dataset consists of a 4 million Amazon customer reviews (input text) and star ratings (output labels). The idea here is a dataset is more than a toy - real business data on a reasonable scale - and can be trained in a few hours on a modest laptop. The dataset can be downloaded from here.

Data format :__label__<X> : <Text>

where X is class name. In our case, the classes are _label_1 and __label__2, and there is only one class per row. __label__1 corresponds to 1-star and 2-star reviews, and __label__2 corresponds to 4-star and 5-star reviews. The review titles, followed by ':' and a space, are prepended to the text. Most of the reviews are in English, but there are a few in other languages, like Spanish.

Dataset Split:

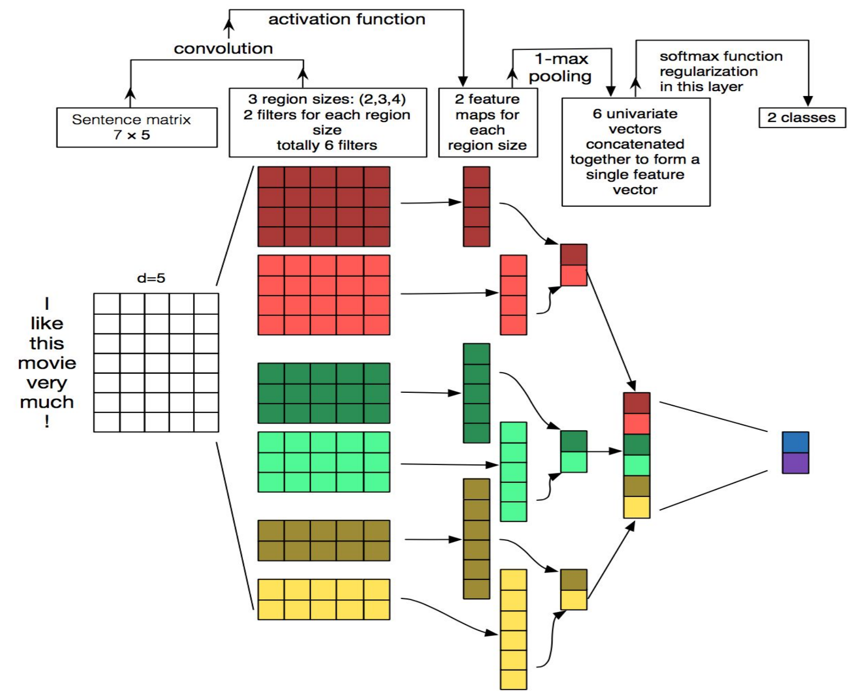
Training Data	**Validation Data**	**Test Data**
0.9 million reviews	**0.1 million reviews**	**0.4 million reviews**

Proposed Solution

We have chosen 3 Deep Learning techniques namely **Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Hierarchical Attention Networks (HAN)** to solve this problem. This will allow us to compare the performance of all 3 Deep Learning models in terms of loss, accuracy, training time and accordingly select the best model suitable for this classification task. We have also used pretrained GLOVE word-embeddings (6B -100D) for obtaining vector representation of words. Each of these models working is explained briefly below. Actual model architectures and choice of hyper parameters is covered in the next section.

1) Convolutional Neural Networks (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 perceptrons designed to require minimal preprocessing. CNNs are generally used in computer vision, however they've recently been applied to various NLP tasks and the results are very promising.


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
!

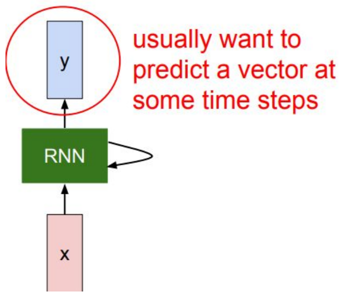
The result of each convolution will fire when a special pattern is detected. By varying the size of the kernels and concatenating their outputs, we can detect patterns of multiples sizes (2, 3, or 5 adjacent words).Patterns could be expressions like "I hate", "very good" and therefore CNNs can identify them in the sentence regardless of their position.

2) Recurrent Neural Networks(RNN)

In CNN, they don't keep track of the previous input in order to understand the context of input data. However, RNN has hidden states which are used to predict output based on given input and previous state..

Basic unit of RNN is as follows.

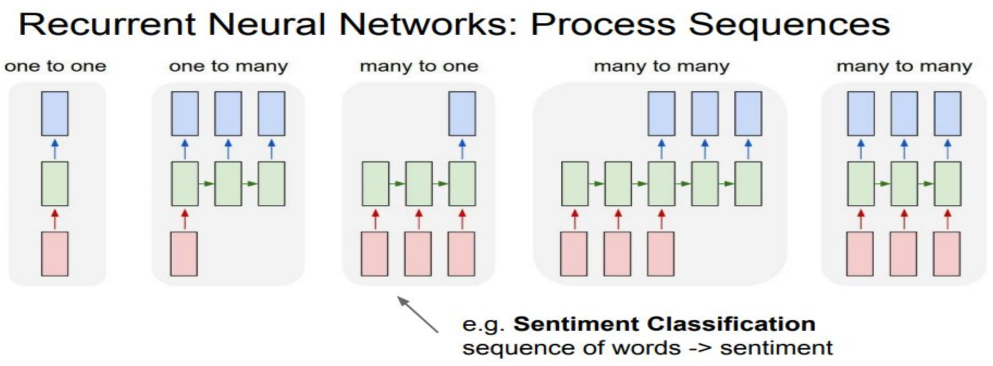
Input not only predict the output but also updates its internal stage.



By packing RNN units one after another we can form a computational graph as shown in the following figure. Internal Weights gets updated based on current inputs and hidden states.

Sentiment Analysis Application:

RNN with layers LSTM - Long Short Term Memory can be used to predict the sentiment. We will use the architecture as shown in figure to build sentiment analysis model. we have we have to predict only 2 types of sentiments positive or negative.


Recurrent Neural Networks: Process Sequences
one to one
one to many
many to one
many to many
many to many
e.g. Sentiment Classification
sequence of words -> sentiment

3) Hierarchical Attention Networks(HAN)

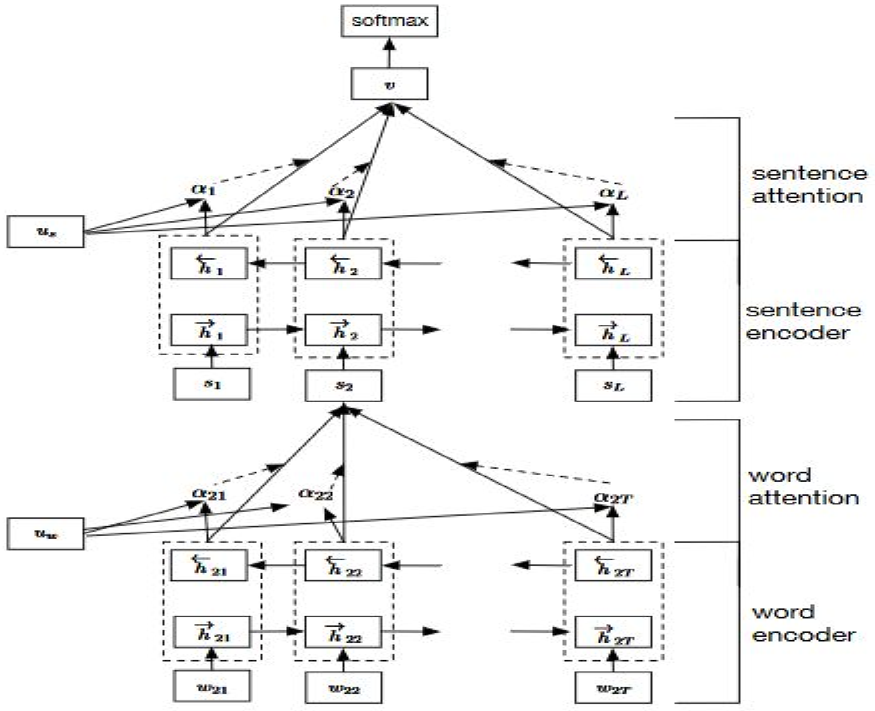
At the core of the Hierarchical Attention Network are the following features:

1. It has a hierarchical structure that mirrors the hierarchical structure of documents. The words form sentences and the sentences form the document.
2. It has two levels of attention mechanisms applied at the word and sentence level, enabling it to attend differentially to more and less important content when constructing the document representation.

GRU-based sequence encoder: It essentially uses the update gate and the reset gate to decide how much of the past information must be kept versus how much of the past information to forget. The current state is an interpolation of the new and old states.

Word Encoder: We first embed the words to vectors using an embedding matrix. A bidirectional GRU is used to get annotations of words by summarizing information from both directions for the words to incorporate contextual information.

Word Attention: The word annotation is fed through a one-layer MLP to get its hidden representation. Then the importance of the word is measured in terms of the similarity of the hidden vector with a word level context vector to get the normalized weight of the word using softmax function. Then the sentence vector is computed as the weighted sum of the word annotations based on the weights.

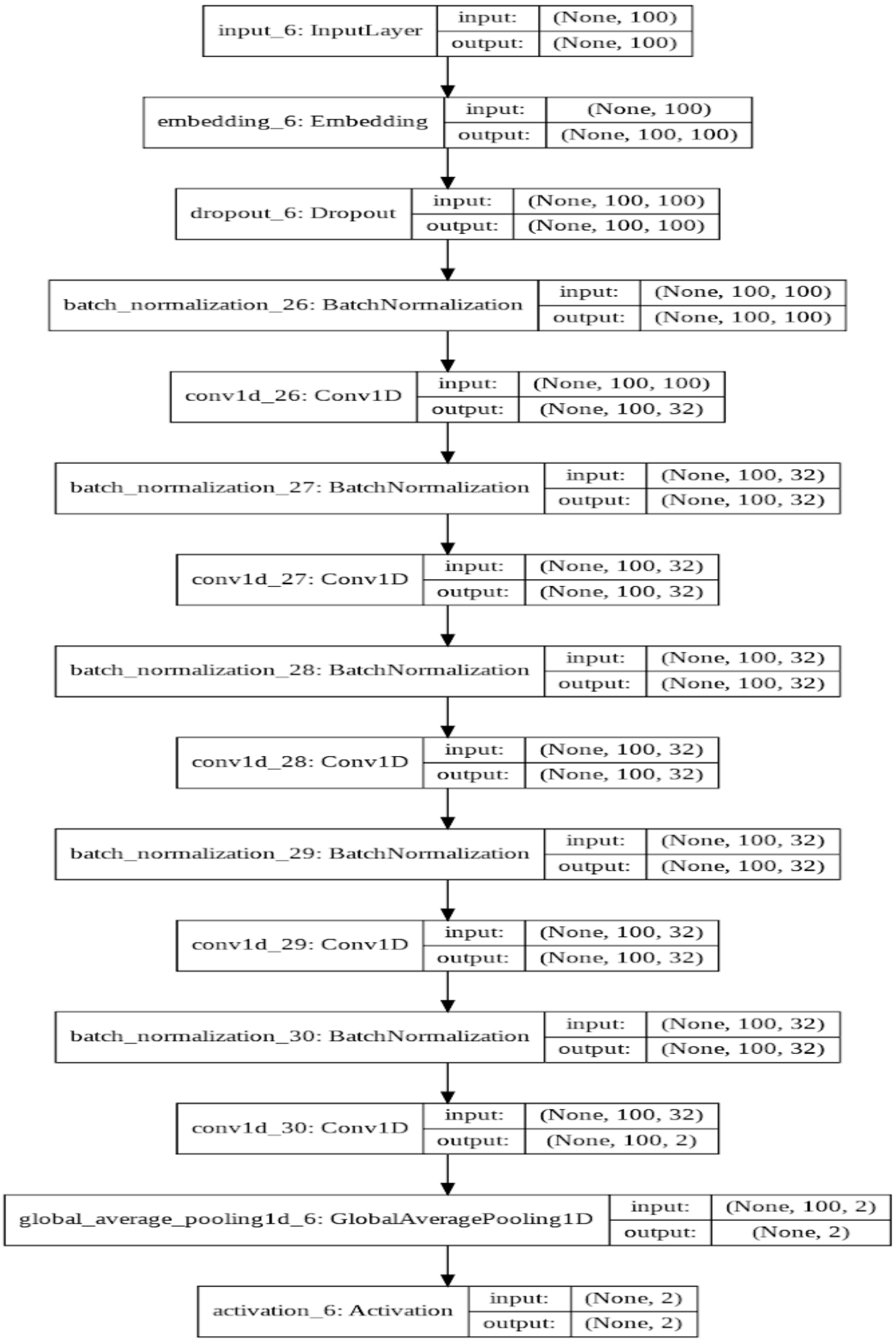


Further the sentence encoder and the sentence attention work in a similar way to get the document vector using bidirectional GRU and to correctly weight all sentences according to their importance to the document.

→ **GLOVE Word Embeddings:**
GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

<u>Model Architecture, Hyperparameter choice and Tuning</u>

a) Convolutional Neural Network


input_6: InputLayer
input: (None, 100)
output: (None, 100)
embedding_6: Embedding
input: (None, 100)
output: (None, 100, 100)
dropout_6: Dropout
input: (None, 100, 100)
output: (None, 100, 100)
batch_normalization_26: BatchNormalization
input: (None, 100, 100)
output: (None, 100, 100)
conv1d_26: Conv1D
input: (None, 100, 100)
output: (None, 100, 32)
batch_normalization_27: BatchNormalization
input: (None, 100, 32)
output: (None, 100, 32)
conv1d_27: Conv1D
input: (None, 100, 32)
output: (None, 100, 32)
batch_normalization_28: BatchNormalization
input: (None, 100, 32)
output: (None, 100, 32)
conv1d_28: Conv1D
input: (None, 100, 32)
output: (None, 100, 32)
batch_normalization_29: BatchNormalization
input: (None, 100, 32)
output: (None, 100, 32)
conv1d_29: Conv1D
input: (None, 100, 32)
output: (None, 100, 32)
batch_normalization_30: BatchNormalization
input: (None, 100, 32)
output: (None, 100, 32)
conv1d_30: Conv1D
input: (None, 100, 32)
output: (None, 100, 2)
global_average_pooling1d_6: GlobalAveragePooling1D
input: (None, 100, 2)
output: (None, 2)
activation_6: Activation
input: (None, 2)
output: (None, 2)

Each sentence(review) is converted to a vector and padded with zero to make it's length uniform that is 100 in this case. The embedding layer defines the input shape and the word vectors (pre-trained GLOVE word-embeddings) that are used to replace each word index with its corresponding vector. A dropout layer with probability (0.2) is used to randomly drop some weights during training. This helps in avoiding overfitting of model on training set. This layer is followed by a sequence of Convolution Layers which help to extract useful text features and Batch Normalization layers which help to normalize input sequence at each stage not just at the start. This plays a major role in avoiding overfitting of models. Global Average pooling helps to get a single value for the final representation of each output mask. A softmax Activation function is used to get the probability of reviews belonging to each class (positive or negative) and the class with higher probability is assigned. (Sigmoid function could have been used but, we have a plan to extend this architecture for more categories in future and so softmax is used.)

CNN Hyperparameters and their effect on model performance

1) **Dataset size:**
   This is not actually a model hyperparameter but varying dataset size has a major impact on model performance**.**

Dataset size	**Number of Epochs**	**Accuracy**
3.6 million	**1**	**92.7%**
0.6 million	**5**	**87%**
1 million	**5**	**92%**

2) **Learning rate**
   Using **Adam** optimizer with learning rate of **0.001** worked best for our model. Adam works best for a huge dataset over most of the choices available. It worked efficiently to learn as quickly as possible and also avoid overfitting over training dataset.
3) **Number of epochs**
   Increasing number of epochs almost always increased the training and validation accuracy for the CNN model. We have used **5** epochs to train the model as per time constraints. It can be increased further to 10 or 15 until there is no overfitting.

4) **Batch size**
   CNN model is sensitive to batch size. We have trained the model using 512, 1024,2048 as batch size. Batch size of 1024 gave the best performance, followed by 2048 and then 512.

5) **Activation function**
   Relu activation function is used in the inner layers of Neural Network as it gives the best performance and works for almost any combination of hyperparameters. It is

computationally very efficient as compared to tanh or sigmoid and hence reduces model training time to a significant extent. **Softmax** activation function is used in the final output layer to assign the reviews to the category with max probability. (Sigmoid can also be used as we have only 2 output classes, but used softmax as we plan to extend this model to more classes and other text classification problems).

6) **Weight initialization**

   Glove word embeddings (**Glove 6B 100d**) serve as initial weights for each word converted to a vector form in 100d vector space. We tried different combinations like not using a word embedding to initialize weights, using only the pre trained version and setting trainable parameter = false and using pre-trained word embeddings along with parameter trainable = true. The third combination worked best.

7) **Dropout for regularization**

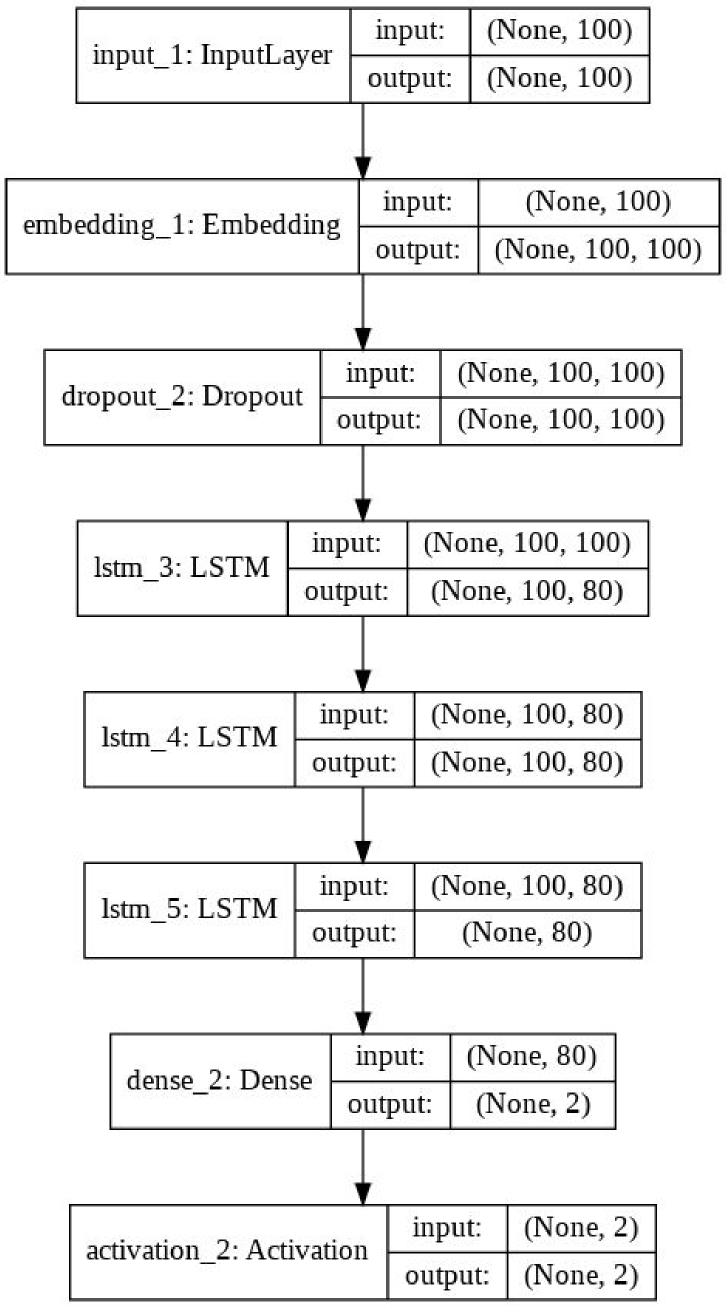
   Dropout layer in model training phase, randomly drops some of the weights (makes some neurons zero). This helps to avoid overfitting model on training set. A dropout probability of 0.4 and 0.2 was tried and 0.2 worked better in this particular case.

CNN Further Improvements:

1) Increasing the training data size or number of epochs will increase model performance.
2) Hyper-parameters tuning (eg : learning rate, batch size, number of epochs). Fine tuning can be done by : Manual Search, Grid Search, Random Search - Cross Validation techniques.
3) Adding more layers of convolution and Batch Normalization may help to improve the performance.

b) Recurrent Neural Networks

Each sentence is limited into vector of 100 words and padded with 0. Embedding layer followed by three layer of 80 LSTM layers. In last but one stage we are taking vector of size 2. Based on the weights, softmax function gives highest probable output. categorical cross entropy loss functions makes sure to avoid biased results towards one sentiments.



RNN Hyperparameters and their effect on model performance

1) **Dataset size:**
   This is not actually a model hyperparameter but varying dataset size has a major impact on model performance**.**

Dataset size	**Number of Epochs**	**Accuracy**
3.6 million	**1**	**91%**

0.6 million	**5**	**85%**
1 million	**5**	**90%**

2) **Learning Rate:**
We have made use of the Adam optimizer with a learning rate of 0.01 because it is suited for large datasets. It helps to learn quickly and avoid overfitting.

3) **Dropout:**
Without applying dropout rate, our model was overfitting. Validation accuracy was less that training accuracy. In order to remove overfitting we tried various dropout rates and found that 0.5 is better one.

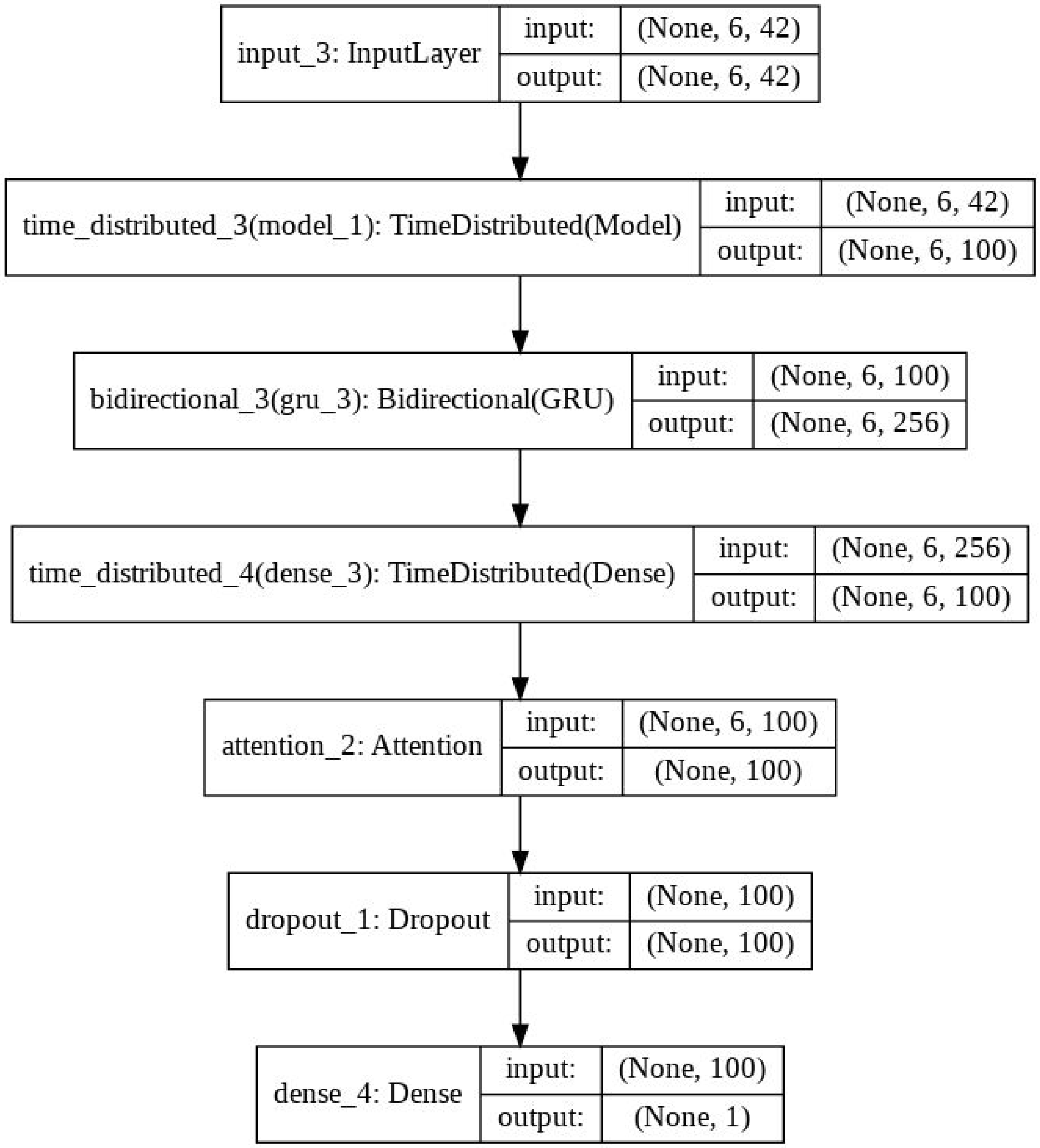
4) **Activation Function:**
We are predicting the sentiments based on given text, which have 2 types of output either 0 or 1. We had two choices for activation function.
1. Softmax with 1-2 encoder vector as output.
2. Sigmoid function with binary as output.
We have chosen softmax however, Sigmoid would have also worked.

5) **Loss Function:**
We have two classes output, in order two avoid skewness towards one the class we have used categorical cross entropy as loss function for model.

6) **Number of Epochs:**
Our models keeps improving as epoch increases, We have restricted ourselves to use 5 epochs considering time and memory limitations. Model will improve if we increase the number of epochs.

c) Hierarchical Attention Networks

- First the input words are converted to vectors using the embedding matrix.
- The annotation of each word is obtained using a bidirectional GRU of 128 neurons.
- This annotation is further used to obtain the weights of each word of the sentence according to its importance. The Attention layer is used to train the weights of each word within the sentence.
- Then the vector representation of the sentence is obtained using a weighted sum of the word representations within that sentence.
- This process is again repeated on a word level to obtain the document vector.
- This document vector is further passed to a dense layer with sigmoid activation to get the class of the review.



Dataset size	**Number of Epochs**	**Accuracy (training)**
3.6 million	**1**	**89.77%**
0.6 million	**5**	**91.9%**
1 million	**5**	**92.32%**

Hyperparameter tuning:

1. **Learning Rate:**
   We have made use of the Adam optimizer with a learning rate of 0.001 because it is suited for large datasets. It helps to learn quickly and avoid overfitting.

2. **Dropout:**
   We experimented with several different dropouts. Using no dropout at all gave us lesser accuracy. A 0.2 dropout improved the accuracy a bit from 0.898 to 0.9 over the test dataset and a 0.5 dropout improved the accuracy a little bit more.

3. **Number of top tokens:**
   This defines the number of top words to be included from the entire training dataset. We experimented with 6000, 8000, and 10000. But there were miniscule changes to the accuracy of the model.

4. **Maximum number of words per sentence:**
   We started out by selecting 10 words per sentence and that gave an accuracy of around 87%. We then computed the average number of words present in all the sentences of our training dataset and that amounted to 42**.** We then designed the model with 42 as the Maximum number of words per sentence and that gave us a model with better accuracy 0.89~

5. **Maximum number of sentences per review:**
   Initialize we started out by setting this field to 3 sentences per review and got an accuracy of around 87. Then we computed the average which came upto 6. The resulting model showed significant improvement. Accuracy: 90%

6. **Number of GRU neurons:**
   We began with 128 neurons and the resulting model had around 403,493 trainable parameters. That model upon training for 5 epochs gave an accuracy of around 90. We then tried a model with 256 number of neurons and the number of trainable parameters rose to around 1 million. This should have meant a better model and the training accuracy became about 91% but the test accuracy stayed the same at 90%.

7. **Activation functions:**
   The attention layer makes use of the tanh activation function. The range of tanh activation function is (+1, -1). The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero. This is useful in the binary classification problem.For the last Dense layer for document classification we had a choice of sigmoid and softmax. We preferred sigmoid over softmax since our output classes are independent of one another and binary in nature.

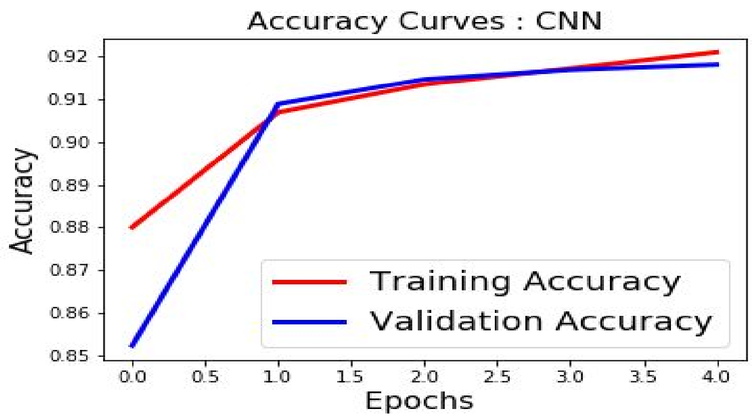
8. **Number of Epochs:**
   Increasing the number of epochs did improve the training accuracy but the test accuracy stayed the same. Thus we stuck to a model with just 5 epochs.

Model Evaluation and Comparison

(Dataset - 1 million reviews, 5 epochs)

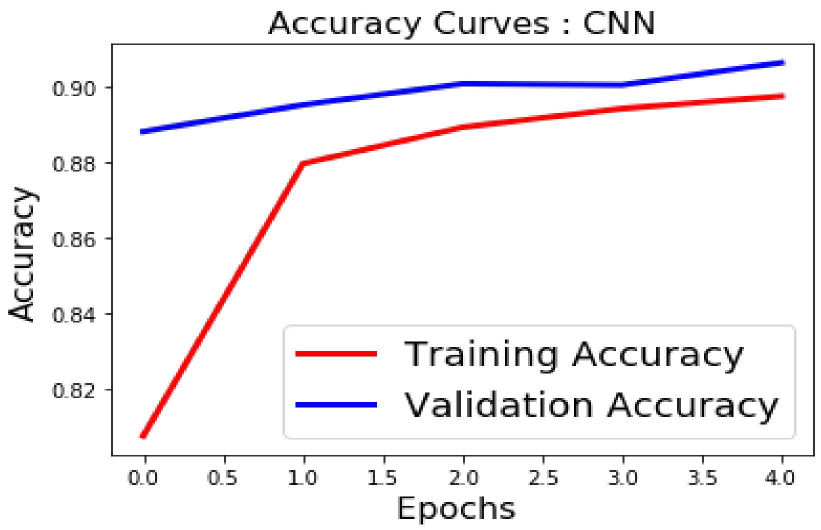
Accuracy Plots:

CNN: Training Accuracy - 91.9%, Validation Accuracy - 91.7%

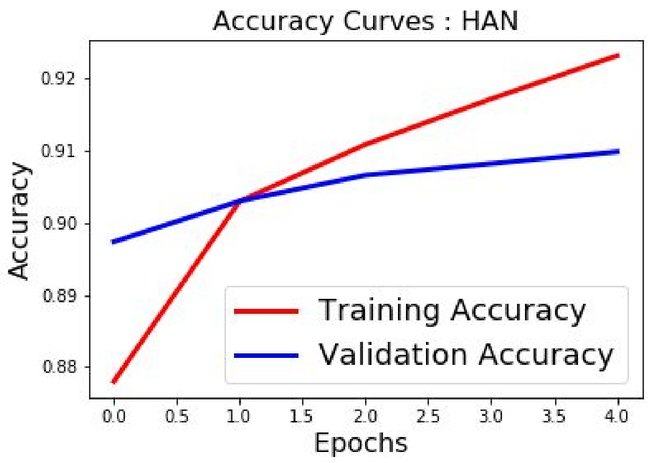


RNN: Training Accuracy - 89%, Validation Accuracy - 90%

- **Ignore name in the plot, its for RNN.. Typo Error**



HAN: Training Accuracy - 92.32%, Validation Accuracy - 90.98%

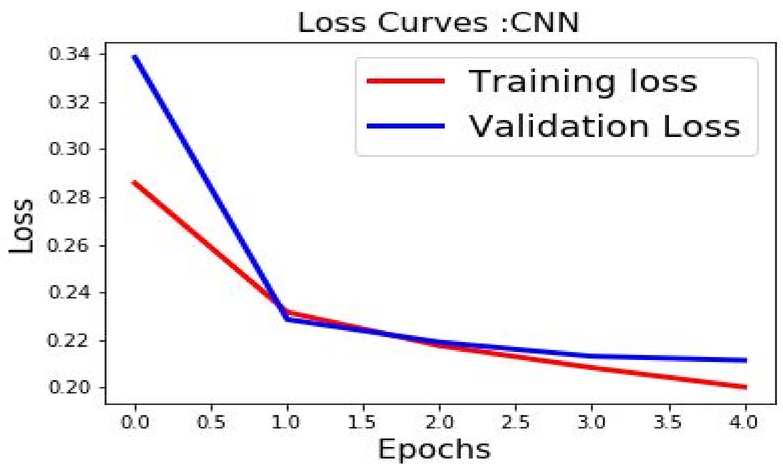


Test Accuracy:

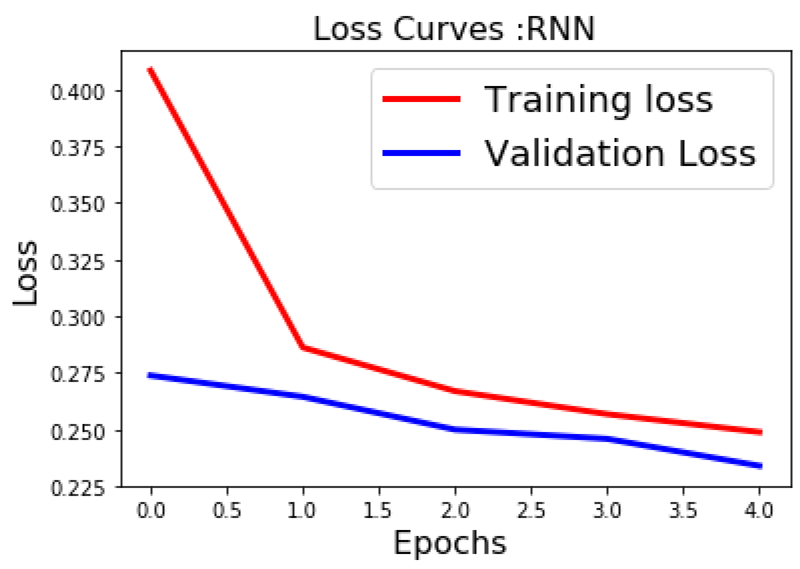
CNN	RNN	HAN
91.7	90.58	91.11

Loss Plots:

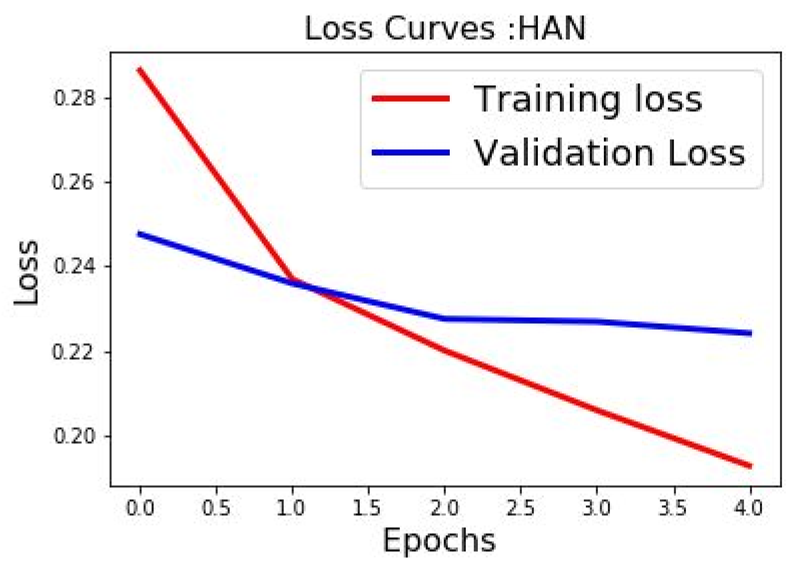
CNN



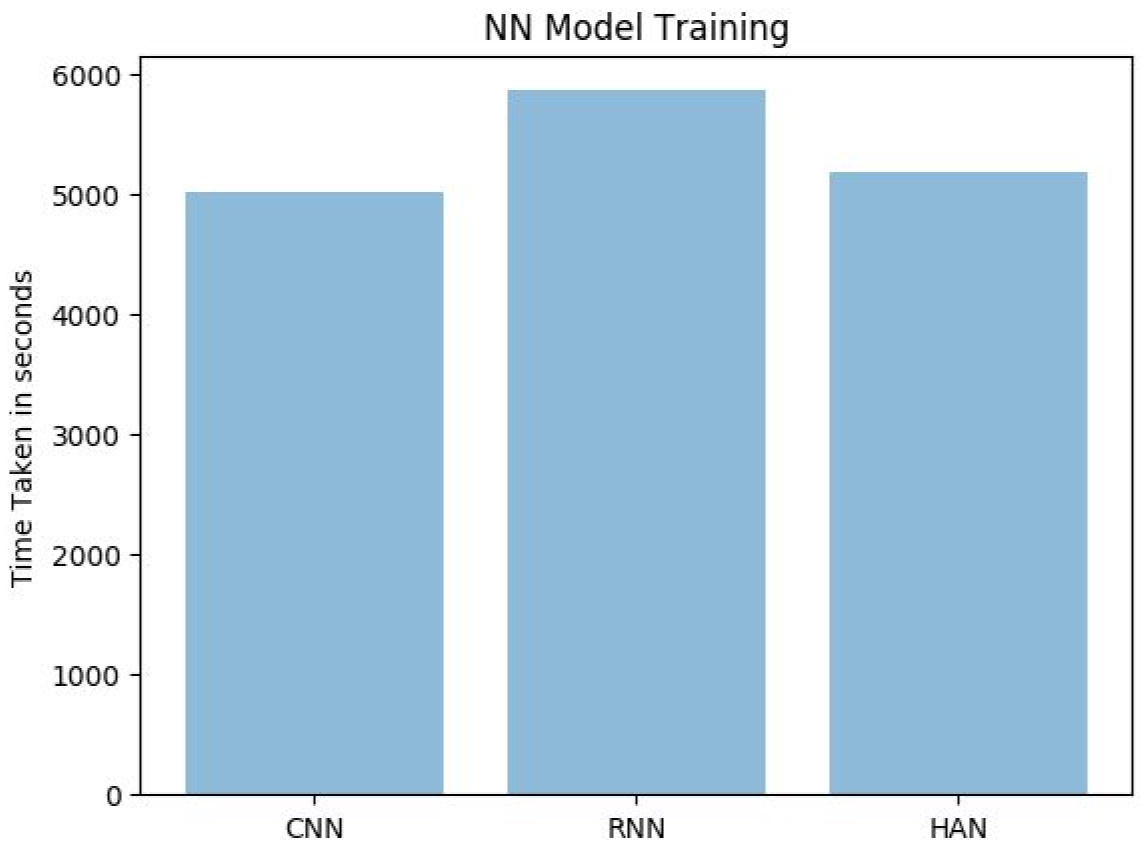
RNN



HAN



Time taken per epoch for training:



General Observations and Summary:

- CNN model has the highest training accuracy followed by HAN and then RNN.
- HAN model seems most promising because it's accuracy is increasing sharply at the end of 5 epochs. CNN and RNN have almost gone flat by the end of 5 epochs. So **HAN would perform the best if trained for more epochs or for larger dataset.**
- CNN outperforms RNN and HAN in terms of validation accuracy.
- Time taken to train is the most for RNN, and almost similar for CNN and HAN.So RNN seems to be the worst choice for production scenario.
- When training samples are less (0.6 million) CNN has achieved the best accuracy too.
- HAN does not perform as well on a smaller dataset as it does for huge datasets.
- So depending on the size of the dataset and the training time available, a choice between CNN and HAN could be made for this classification problem.