Sentiment Analysis Using RNN

***Abstract*—We introduce the problem of Sentiment Analysis of large movie reviews, consisting of multiple sentences. We leverage a new architecture that uses Recurrent Neural Network with Gated Recurrent Units (GRUs). Sentiment analysis is a well-researched natural language processing field. It is a challenging machine learning task due to the recursive nature of sentences, different length of documents and sarcasm. Traditional approaches to sentiment analysis use count or frequency of words in the text which are assigned sentiment value by some expert. These approaches disregard the order of words and the complex meanings they can convey. Gated Recurrent Units are recent form of recurrent neural network which can store information of long-term dependencies in sequential data. In this work we showed that GRU are suitable for processing long textual data and applied it to the task of sentiment analysis.**

Keywords— *Sentiment Analysis, GRU, Recurrent Neural Network*

I. INTRODUCTION

Sentiment analysis is the process of extracting the attitude of the author towards the entity the text is written. At the simplest level sentiment analysis is a binary classification task which involves deciding whether a text is positive or negative. Sentiment analysis is done at both at phrase level and document or paragraph level.

Both the levels offer unique challenges and hence require different techniques to tackle them. There is an increasing availability of various websites which allows customers to write reviews on various products, movies or hotels. In the reviews the customers share their experience of using that service. Sentiment analysis is applied to such reviews to discover the customers opinion on that service. This is of important value to companies, stocks and politics.

The advent of deep learning approaches has given rise to several new methods for sentiment analysis. The availability of large unlabeled textual data can be used to learn the meanings of words and the structure of sentence formation. This has been attempted by word2vec [1] learns word embeddings from unlabeled text samples. It learns both by predicting the word given its surrounding words (CBOW) and predicting surrounding words from given word (SKIP-GRAM). These word embeddings are used for creating dictionaries and act as dimensionality reducers in traditional method like tf-idf etc. More approaches are found capturing sentence level representations like recursive neural tensor network (RNTN) [2].

Convolution neural network which has primarily been used for image related task has been shown effective in text classification too [3]. The main problem is the variable length of the natural language. Some of it is solved by fixed size context windows but it fails to capture dependencies which extend longer than the context window. Recurrent neural network can take variable length of text sequence, but they are extremely tricky to learn. Hence new types of RNN were employed like LSTM and GRU. LSTM was proposed in 1997 by Hochreiter et al. [4] and is making news in many nlp task like sentiment analysis, translation and sequence generation. GRUs is quite recent development proposed by K. Cho[5] in 2014. GRU are much simpler in structure and probably more practical than LSTM. We attempt to show its advantages over LSTM in sentiment analysis in this work.

We found that GRUs were effective in the task of sentiment analysis because of their ability to remember long time dependencies. GRU are especially useful for large texts. We discussed that LSTM networks were quite slow to train and did not converge to better accuracy even after many epochs. Hence, we feel GRUs as better replacements to LSTM.

To solve this problem, we need several processing steps. First we need to convert the raw text-words into so-called tokens which are integer values. These tokens are just indices into a list of the entire vocabulary. Then we convert these integer-tokens into so-called embeddings which are real-valued vectors, whose mapping will be trained along with the neural network, to map words with similar meanings to similar embedding-vectors. Then we input these embedding-vectors to a Recurrent Neural Network which can take sequences of arbitrary length as input and output a kind of summary of what it has seen in the input. This output is then squashed using a Sigmoid function to give us a value between 0.0 and 1.0, where 0.0 is taken to mean a negative sentiment and 1.0 means a positive sentiment. This whole process allows us to classify input-text as either having a negative or positive sentiment.

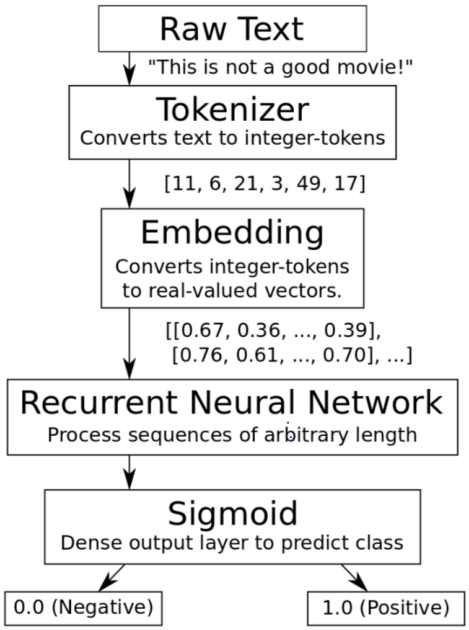

Figure 1: Flowchart of Algorithm

II. LITERATURE REVIEW

A recurrent neural network (RNN) is like conventional feed forward network, with the difference that it has connections to units in the same layer. This provides them an internal memory which can handle a variable length input sequence. It handles the variable length sequences by having a recurrent hidden state whose activation at each time is dependent on that of previous time. The human brain is a recurrent neural network, a network of neurons with feedback connections and hence using them brings us closest to natural design. The basic building block in a Recurrent Neural Network (RNN) is a Recurrent Unit (RU).

The following figure shows the abstract idea of a recurrent unit, which has an internal state that is being updated every time the unit receives a new input. This internal state serves as a kind of memory. However, it is not a traditional kind of computer memory which stores bits that are either on or off. Instead the recurrent unit stores floating-point values in its memory-state, which are read and written using matrix-operations, so the operations are all differentiable. This means the memory-state can store arbitrary floating-point values (although typically limited between -1.0 and 1.0) and the network can be trained like a normal neural network using Gradient Descent.

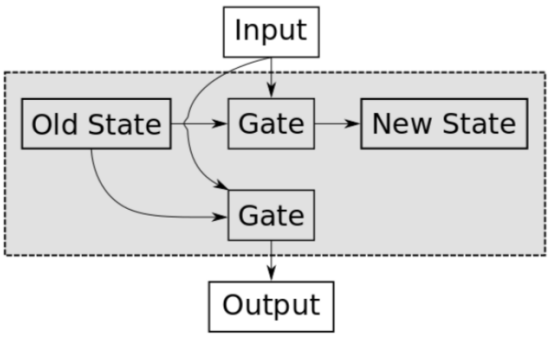

Figure 2: Basic Abstraction of a Recurrent Unit

The new state-value depends on both the old state-value and the current input. For example, if the state-value has memorized that we have recently seen the word "not" and the current input is "good" then we need to store a new state-value that memorizes "not good" which indicates a negative sentiment.

The part of the recurrent unit that is responsible for mapping old state-values and inputs to the new state-value is called a gate, but it is really just a type of matrix-operation. There is another gate for calculating the output-values of the recurrent unit. The implementation of these gates varies for different types of recurrent units. This figure merely shows the abstract idea of a recurrent unit. The LSTM has more gates than the GRU but some of them are apparently redundant so they can be omitted.

In order to train the recurrent unit, we must gradually change the weight-matrices of the gates, so the recurrent unit gives the desired output for an input sequence. This is done automatically in TensorFlow. However, RNNs are difficult to train and suffer from vanishing and exploding gradients problem. Either the gradients become so small that learning stops, or the gradient becomes so large that the weights overflow the max limit. The most effective solution to this problem is adding a gating mechanism to the RNN. Two gated RNN are found in literature

- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)

The Long Short-Term Memory (LSTM) unit was initially proposed by Hochreiter and Schmidhuber[4]. It has undergone many changes over the years like the addition of forget gate. LSTM consists of a memory cell to store information. It computes input gate, forget gate and output gate to manage this memory. LSTM units can propagate an important feature that came early in the input sequence over a long distance, hence, capturing potential long-distance dependencies. LSTM despite being complex are very successful in various tasks like handwriting recognition, machine translation and of course sentiment analysis.

The gated recurrent unit GRU is relatively recent development proposed by Cho et al. [5]. Like the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. Gated Recurrent Unit (GRU) calculates two gates called update and reset gates which control the flow of information through each hidden unit.

The GRU can be used for text classification in the similar manner as it has been used for LSTM by Le P. et al. [6] and Cho K [5,17]. First, an embedding layer of appropriate size is created. Then the words in the phrase are fed into the first layer as 1 of K encoding while maintaining their order. The embedding layer will learn to represent each word by a real valued vector of size equal to the fixed dimension. These values are the weights between the embedding layer and the hidden layer on top of it. The hidden layer consists of gated recurrent units. These are not only connected to the layer below and above them but also connected to units within their own layer. We used only single layered GRUs even though it is possible to stack them. At the end of the hidden layer we get the representation of the entire sequence which can be used as input to linear model or classifier. We used sigmoid function for binary classification in the dense layer (output layer).

III. ARCHITECTURE

In the following, figure there is only a single recurrent unit denoted RU, which will receive a text-word from the input sequence in a series of time-steps.

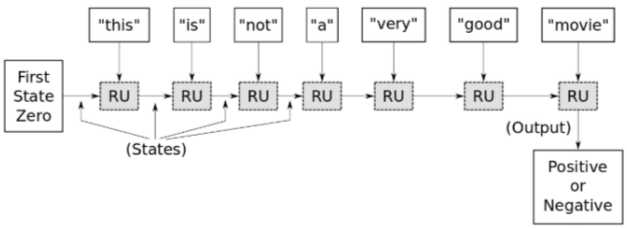

Figure 3: Unrolled layered network

The initial memory-state of the RU is reset to zero internally by Keras / TensorFlow every time a new sequence begins. In the first time-step the word "this" is input to the RU which uses its internal state (initialized to zero) and its gate to calculate the new state. The RU also uses its other gate to calculate the output, but it is ignored here because it is only needed at the end of the sequence to output a kind of summary. In the second time-step the word "is" is input to the RU which now uses the internal state that was just updated from seeing the previous word "this".

There is not much meaning in the words "this is" so the RU probably doesn't save anything important in its internal state from seeing these words. But when it sees the third word "not" the RU has learned that it may be important for determining the overall sentiment of the input-text, so it needs to be stored in the memory-state of the RU, which can be used later when the RU sees the word "good" in time-step 6.

Finally, when the entire sequence has been processed, the RU outputs a vector of values that summarizes what it has seen in the input sequence. We then use a fully connected layer with a Sigmoid activation to get a single value between 0.0 and 1.0 which we interpret as the sentiment either being negative (values close to 0.0) or positive (values close to 1.0).

We will use a Recurrent Neural Network with 3 recurrent units (or layers) denoted RU1, RU2 and RU3 in the "unrolled" figure below.

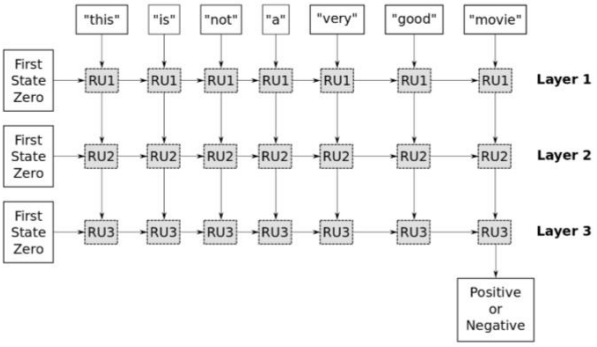

Figure 4: 3 Layer Unrolled Netwoprk

The first layer is much like the unrolled figure above for a single-layer RNN. First the recurrent unit RU1 has its internal state initialized to zero by Keras / TensorFlow. Then the word "this" is input to RU1 and it updates its internal state. Then it processes the next word "is", and so forth. But instead of outputting a single summary value at the end of the sequence, we use the output of RU1 for every time-step.

This creates a new sequence that can then be used as input for the next recurrent unit RU2. The same process is repeated for the second layer and this creates a new output sequence which is then input to the third layer's recurrent unit RU3, whose final output is passed to a fully-connected Sigmoid layer that outputs a value between 0.0 (negative sentiment) and 1.0 (positive sentiment).

In order to train the weights for the gates inside the recurrent unit, we need to minimize some loss-function which measures the difference between the actual output of the network relative to the desired output. From the "unrolled" figures above we see that the recurrent units are applied recursively for each word in the input sequence. This means the recurrent gate is applied once for each time-step. The gradient-signals must flow back from the loss-function all the way to the first time the recurrent gate is used. If the gradient of the recurrent gate is multiplicative, then we essentially have an exponential function.

In our, we will use texts that have more than 500 words. This means the RU's gate for updating its internal memory-state is applied recursively more than 500 times. If a gradient of just 1.01 is multiplied with itself 500 times, then it gives a value of about 145. If a gradient of just 0.99 is multiplied with itself 500 times, then it gives a value of about 0.007. These are called exploding and vanishing gradients. The only gradients that can survive recurrent multiplication are 0 and 1.

To avoid these so-called exploding and vanishing gradients, care must be made when designing the recurrent unit and its gates. That is why the actual implementation of the GRU is more complicated, because it tries to send the gradient back through the gates without this distortion.

IV. RESULTS

The results for this paper were obtained using the IMDB dataset originally collected by Andrew Maas [11]. It consists of the labeled data set of 50,000 IMDB movie reviews, specially selected for sentiment analysis. The sentiment of reviews is binary, meaning the IMDB rating < 5 results in a sentiment score of 0, and rating >=7 have a sentiment score of 1. We will have 23750 training samples and 1250 test samples.

For our experimentation, we used our Google Research Colab on our own desktop and used runtime type as GPU. We imported the required necessary libraries such as Keras. The average learning time for 10 epochs was 5 hours. Our vocabulary size is 10000 words.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_embedding (Embedding)  (None, 544, 8)            80000
_________________________________________________________________
gru (GRU)                    (None, 544, 16)           1200
_________________________________________________________________
gru_1 (GRU)                  (None, 544, 8)            600
_________________________________________________________________
gru_2 (GRU)                  (None, 4)                 156
_________________________________________________________________
dense (Dense)                (None, 1)                 5
=================================================================
Total params: 81,961
Trainable params: 81,961
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Sequential Model RNN

Our model consists of 3 GRU Layers, one embedding layer and one Dense layer. The dense layer acts as the output layer since it consists of the Activation function which in our case is Sigmoid. We fixed our embedding vector dimensions to 8 and then trained our model for 3, 4 and 10 epochs. Naturally, the more we train the data the more accurate it should be. We get the following results after training the data on different epochs.

RNN MODEL EPOCHS	ACCURACY
3	72.55%
4	75.72%
10	83.58%

Table 1: Accuracy for different epochs for our RNN model

text1 = "This movie is fantastic! I really like it because it is so good!"
text2 = "Good movie!"
text3 = "Maybe I like this movie."
text4 = "Meh ..."
text5 = "If I were a drunk teenager then this movie might be good."
text6 = "Bad movie!"
text7 = "Not a good movie!"
text8 = "This movie really sucks! Can I get my money back please?"
texts = [text1, text2, text3, text4, text5, text6, text7, text8]

We passed the array texts through all the models trained above and we get the following results. Every row of the array corresponds the predicted sentiment score of the above-mentioned texts.

```
array([[0.868934  ],
       [0.72526425],
       [0.33099633],
       [0.49190348],
       [0.3054021 ],
       [0.14959489],
       [0.5235635 ],
       [0.21565402]], dtype=float32)
```

Model 1- Epoch 3 Result

```
array([[0.96067977],
       [0.89856577],
       [0.75594366],
       [0.84377855],
       [0.67702633],
       [0.42514688],
       [0.8552542 ],
       [0.25050125]], dtype=float32)
```

Model 2 Epoch 4 Result

```
array([[0.95066727],
       [0.94813561],
       [0.45628396],
       [0.52337514],
       [0.52308733],
       [0.42514688],
       [0.8552542 ],
       [0.15560128]], dtype=float32)
```

Model 3 Epoch 10 Result

Now we can observe that the model 3 with most epochs has the highest accuracy among all the models.

V. CONCLUSION

Sentiment Analysis remains a popular and an important field of natural language processing. We conclude that gated recurrent units are a good enough model for sentiment analysis at a paragraph level. Being a recurrent network, it can effectively capture long sequence data required for natural language understanding. They perform better than traditional bag of features models which disregard the order of the features. They eliminate the problem of exploding and diminishing gradient problem as effectively based on the design of how the gates are designed. The GRU converges to the solution faster than LSTM. The GRU networks hence look promising and may create the state of the art in sentiment analysis and other NLP tasks.

REFERENCES

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013.

[2] Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng and Chris Potts. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013).

[3] Yoon Kim. Convolution Neural Network for Sentence Classification, EMNLP 2014

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

[5] Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. arXiv preprint arXiv:1409.1259.

[6] Compositional Distributional Semantics with Long Short Term Memory, Phong Le and Willem Zuidema,2015

[7] Alec Radford, https://indico.io/blog/passage-text-analysis-withrecurrent-neural-nets-next-ml-cambridge/

[8] Shamim Biswas , Ekamber Chadda and Faiyaz Ahmad (2015), Sentiment Analysis with Gated Recurrent Units, Advances in Computer Science and Information Technology (ACSIT)

[9] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). "Learning Word Vectors for Sentiment Analysis." The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)