IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-5, Issue-9 E-ISSN: 2347-2693

Sentiment Analysis Based on a Deep Stochastic Network and Active Learning

Tulsi Jain[1*], Kushagra Agarwal[2], Ronil Pancholia[3]

[1*]Dept. of CSE, Indian Institute of Technology (IIT), Delhi, India
[2]Dept. of CSE, Indian Institute of Technology (IIT), Delhi, India
[3]Dept. of CSE, Birla Institute of Technology and Science, Pilani, India

[*]*Corresponding Author: jaintulsi43@gmail.com*

Available online at: www.ijcseonline.org

Received: 22/Aug/2017, Revised: 08/Sep/2017, Accepted: 19/Sep/2017, Published: 30/Sep/2017

***Abstract*—** this paper proposes a novel approach for sentiment analysis. The growing importance of sentiment analysis commensurate with the use of social media such as reviews, forum discussions, blogs, micro blogs like Twitter, and other social networks. We require efficient and higher accuracy algorithms in sentiment polarity classification as well as sentiment strength detection. In comparison to pure vocabulary based system, deep learning algorithms show significantly higher performance. The goal of this research is to modify a Recurrent Neural Network (RNN) with Gated Recurrent Unit (GRU) by introducing stochastic depth in a hidden layer and comparing it with baseline Naïve Bayes, vanilla RNN and GRU-RNN models. To improve our results, we also incorporated Active Learning with Uncertainty Sampling approach. Movie review dataset from Rotten Tomatoes was used, the dataset includes 215,154 fine grained labelled phrases in addition to 11,855 full sentences. We performed pre-processing on the data and used an embedding matrix with pre-trained word vectors as features for training our model. These word vectors were generated using character level n-grams with fasttext on Wikipedia data.

***Keywords*—** Fasttext, Recurrent Neural Network, Gated Recurrent Unit, Active Learning

I. INTRODUCTION

Sentiment analysis arrives from the field of study that deals with analysing emotions, attitude, and sentiments attached with the text. The computational analysis of opinion, sentiment, and subjectivity is an active area of research in recent time, because of its wide range of applications.

Traditional approaches for sentiment analysis have focused on calculating sentiment at the document level. However, accurate evaluation of sentiment for complex documents containing a mixture of positives and negatives movie reviews require sentence-level or even phrase-level sentiment analysis.

We believe that the lack of suitable labelled data that could be used in machine learning techniques to train sentiment classifiers is one of the major reasons the field of sentiment analysis is not advancing more rapidly. As we will compare the results of various machine learning models and recommend the best performing model. We observed that few predictions were corresponding to a low probability distribution. So, we also applied active learning on top of it to further improve the accuracy of the individual model.

This paper is organised in the following manner. Section I starts with the need for sentiment analysis and introduction of traditional approaches for sentiment analysis like Naïve Bayes, Section II contains the related work on sentiment analysis and describes our baseline approaches. Proceeding ahead, Section III provides details to the baseline approaches and continues developing methodology of revised versions of Recurrent Neural Network by introducing Stochastic Depth in GRU. Section IV describes the Experiments performed and discusses results achieved by the proposed model. Finally, Section V concludes research work with future directions.

II. RELATED WORK

Bag of words models is the first fundamental approach to sentiment analysis. As the name suggests, in these type of model problem is addressed by assuming that a bunch of non-dependent words formed a complex sentence, assumed independence of word ordering and context[5]. Although limitation of above-mentioned assumptions can be overcome up to a certain extent, especially when numerous superlatives are present, can do not consider very basic interactions because they do not encode any structure of the consecutive word because of the robust assumption of normal distribution. Another big limitation of Naïve Bayes[5] is the assumption of word independence, which is not true. For example, if the word ‘good' is encountered in a review, it expresses a strong positive sentiment. However, if that word is preceded by ‘not', as in ‘not good', then it likely expresses a negative sentiment. To solve this problem, many took the

alternative of trying to create hand engineered features, however as we know language is complicated, heuristically designed rules become too complex and highly dependent on the dataset. Hence, the chances of overfitting increase. In the past, many researchers worked to find a robust solution to create techniques for designing features. One of the most prominent ideas is developing word vectors - mapping the text of words to a high dimensional embedded vector space. A primary motivation for such a transformation is that, in nonlinear learning techniques like neural networks, convolutional neural networks etc. a word vector representation helps in providing a semantically better context.

Deep learning with neural networks is extremely powerful. However, applying them is not straightforward. Some of the issues we face when dealing with deep neural networks are: 1) elongated training times 2) vanishing gradient problems and 3) overfitting of training data. The idea introduced by Huang [2] of training an ensemble of models with stochastic depth is explored in this paper. Using this technique, we can train shorted models while at prediction use the deep model. Thus, ensuring we don't lose on accuracy but also have reduced training time [3]. The epitome of this idea is that for a given network, for each layer of our network i, the Bernoulli random variable is $d_i \sim Bernoulli(p_s)$, where p_s is the survival probability. If $d_i = 0$, then the activation function of layer i comes out to be equal to the identity function and we directly feed forward the hidden state from the previous input layer as our output hidden state. So, in the previous model the output of hidden layer i can be expressed as:

$$h_i = ReLU\big(f_i h_{i-1} + id(h_{i-1})\big) \dots (1)$$

Training a model with stochastic depth, the hidden state now has an element of randomness depending on the Bernoulli variable. It then becomes:

$$h_i = ReLU\big(d_i f_i h_{i-1} + id(h_{i-1})\big) \dots (2)$$

which becomes identity function of $h_{i-1}$when $d_i = 0$ and the original network when $d_i = 1$. This technique is described in the original paper by Huang, as convolution neural networks in the application of image classification. The biggest advantage of this technique is that it's simple at the surface and hence doesn't comes across issues which are possible in potentially deep networks. Effectively in this approach, the network is shortened to reduce the risk of vanishing gradients. For a given input, which in this case is a sequence of M different word vectors, there are 2M different possible networks. Out of these 2M networks, one sample is used and updated. This architecture training is effectively an ensemble training of different models. Training with stochastic depth adds its own element of regularization along with L2 penalty on your model weights helps improve the model's ability to generalize well in an out of sample context.

Image and audio data are already represented in the form of numbers but while working with text data, we need to convert it into a numerical format, to be able to run our deep learning model. One simple approach is to use separate vectors for each of the words, but this does not address even the very basic problems. Firstly, we will have high dimensional vectors with all 'zeroes' and exactly one 'one' corresponding to the word's position. This type of vector creation is also known as one-hot encoding. Secondly, this simple approach does not take the natural notion of similarity into consideration. By using word embedding, the above-mentioned problem can be addressed up to a great extent. Syntactic and semantic word similarities play a very important role in a majority of NLP tasks. It is quite possible that the same word may represent different meanings depending on its surrounding words (which is known as context). In layman terms, let's take a word 'bark'. One of the meanings of this word is a tree's outer layer and another meaning is the sound a dog makes. If in a sentence, bark occurs with neighbouring words as dog, bite, hurt, etc. we can understand it's the latter meaning. Contrarily, if neighbouring words are tree, wood, etc. the reference is for former. We will exploit this concept to deal with polysemy and synonyms and make our model learn.

Word embeddings are a continuous vector representation of words in n-dimensional space which basically is an efficient learning of word representations. These embedding can be generated using fasttext [6], which trains character level n-grams of a huge dataset in a shallow neural network. These vectors help us find words which are syntactically and semantically similar. For our experiments, we are using pre-trained 300-dimensional vectors generated from Wikipedia data using skip-gram model described in Bojanowski et al. (2016) [1].

III. METHODOLOGY

Baseline Approach: Naïve Bayes

Naïve Bayes[5] was used as a baseline classifier. In Naïve Bayes classification takes advantage of the assumption of conditional independence among attributes. This assumption of conditional independence is not always true. This can be clearly seen in examples where a negative keyword placed before a word makes a big difference in the sentiment for that sentence. Another issue with Naïve Bayes comes when you have no occurrences of a particular class label with a particular attribute value together.

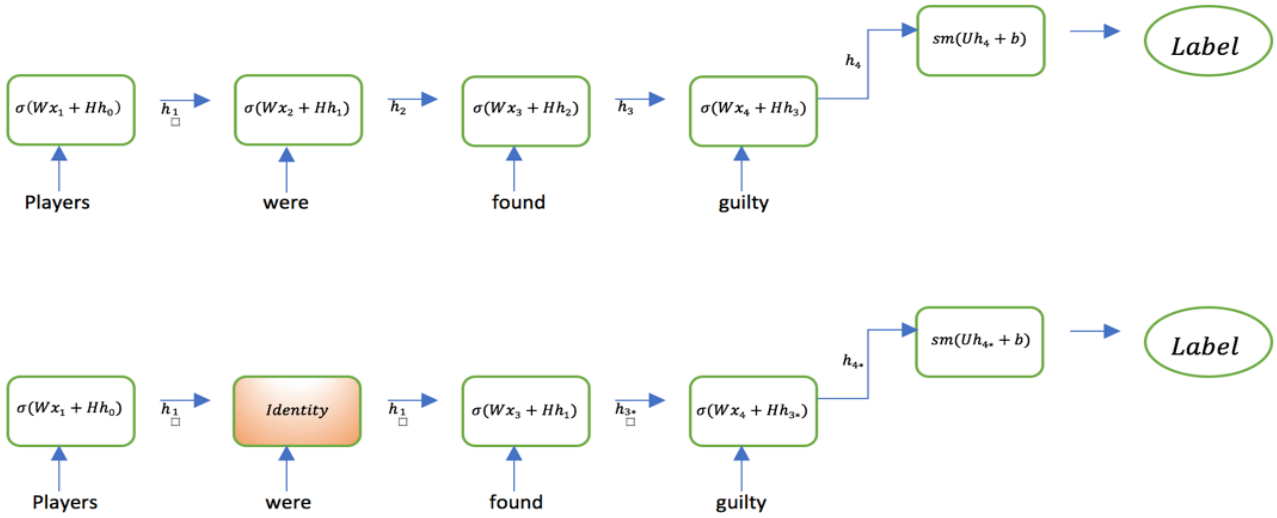

Figure 1. In the above figure, we are describing difference between a recurrent neural network and its variant by adding stochastic depth. Identity function is used in second recurrent cell hence as it passes directly into the next layer without modification.

This causes the probability estimates for this attribute to be zero, affecting the posterior probability.

Being a very basic model only dependent on bag of words, this classifier still achieves good results, hence is used as baseline metric. We are primarily concerned with estimating the class for sentence s, $p(C^{(i)}|a_1^{(s)}, \dots, a_n^{(s)})$. Here, $C^{(s)}$ denotes the class of sentence and $a_i^{(s)}$ represents the words in the sentence. Using Bayes rules, the conditional probability is estimated as

$$p\left(C^{(l)}\middle|a_1^{(l)}, \dots, a_n^{(l)}\right) = \frac{p(C^{(l)})p\left(a_1^{(l)}, \dots, a_n^{(l)}\middle|C^{(l)}\right)}{p\left(a_1^{(l)}, \dots, a_n^{(l)}\right)} = \frac{p(C^{(l)})\prod_{i=1}^{n} p\left(a_i^{(l)}\middle|C^{(l)}\right)}{p\left(a_1^{(l)}, \dots, a_n^{(l)}\right)} \dots (3)$$

Generally, the order of the search space is too huge and the conditional independence assumption reduces it drastically. But conditional independence isn't always a correct assumption.

Recurrent Neural Network

A recurrent neural network[7] architecture is a class of artificial neural network where connections between units form a directed cycle. This allows it to exhibit dynamic temporal behaviour. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. A standard recurrent neural network for each step i, takes as input not just the current input x_i example but also the example perceived in the previous step h_i to create a hidden state based upon and both x_i and h_i. Generally, the composition function is passed through some non-linear transformation like hyperbolic tangent, sigmoid and ReLU leading to an affine transformation. Each time step in terms of an input state and a hidden state can be defined as

$$h_i = f\left(W_{x_i} + Hh_{i-1} + b_1\right) \dots (4)$$

with f being the activation function. The parameters for the model are h, W and b. Here, the hidden layer used is affine transformation which then passes through an activation function to calculate the probability distribution $\hat{y_j}$ of the j^{th} label. Each sentence is converted into a sequence to normalise the sentence before passing into the model, after which a single vector is generated as output. Thus, the output, $\hat{y_j}$ is stated as

$$\hat{y_j} = softmax(Uh_L + b_2) \dots (5)$$

where M is the length of the sentence sequence. Figure 1 describes the network used in the model. On applying the softmax function, labels with highest probability were obtained for that sequence. Although, this structure becomes problematic when longer term word relationships are encountered and also managing vanishing gradients is a concern. At every time step in GRU (Gated Recurrent Unit), an update gate and a reset gate are added, that helps in controlling the information flow within a sequence without using a memory unit. The update gate z_i is added as

$$z_i = \sigma\left(W^{(z)}x_i + H^{(z)}h(i-1)\right) \dots (6)$$

The reset gate is added as

$$r_i = \sigma\left(W^{(r)}x_i + H^{(r)}h_{(i-1)}\right) \dots (7)$$

With the combination of these gates, the hidden cell state, $\tilde{h}_i$ can be computed as

$$\tilde{h}_i = \tanh\left(Wx_i + r_i \circ Hh_{(i-1)} + b_i\right) \dots (8)$$

where $\circ$ is the Hadamard product which is the entry-wise product of two matrices. This hidden state information is encoded. The state information that gets transferred to the subsequent cell in the network, is qualified by the update gate. This state is computed as

$$h_i = z_i \circ h_{(i-1)} + (1 - z_i) \circ \tilde{h}_i \dots (9)$$

Although a new cell is defined in the network, the information flow from one cell to another is unchanged.

Adding to this we introduce stochastic dominance to the model. While training, every node is weighed with a random Bernoulli variable $d_i \sim Bernoulli(p_s)$, where p_s is the probability of survival for that layer. As proposed by Huang [2], we add this Bernoulli variable to nodes of individual hidden layers. The layer stays unchanged where the value of d_i is *one*. However, when d_i is *zero*, hidden state of the precursor node is passed forward. The output h_i for GRU with this added Bernoulli variable can be computed as

$$h_i = d_i(z_i \circ h_{(i-1)} + (1 - z_i) \circ \tilde{h}_i) + (1 - d_i)h_{(i-1)} \dots (10)$$

As shown in figure 1, for each node random variable is drawn in the second network. We drop out the second node in the network to feed forward the hidden state of the second node. This in effect transforms the training sentence from "Players were found guilty" to "Players found guilty". This greatly reduces the network size which further means reduced training time. Although during prediction, the complete input is fed into the network, each input is amended based on the survival probability at the time of training.

Thus, the hidden state output h_i^{test} during prediction for the vanilla recurrent network is

$$h_i^{test} = f(p_s W x_i + H h_{i-1} + b_1) \dots (11)$$

For the GRU network

$$\tilde{h}_i^{test} = tanh(p_s W x_i + r_i \circ H h_{i-1} + b_1) \dots (12)$$

IV. Experiments and Results

Throughout the experimentation, Stanford Sentiment Treebank Rotten Tomatoes data set is used. All of the techniques have been implemented with python library, Tensorflow by Google along with a combination of scikit-learn and pandas libraries. The code was executed on GPU Amazon GPU web server. Dataset used was in the form of binary classification (with positive and negative class), as opposed to the fine-grained analysis.

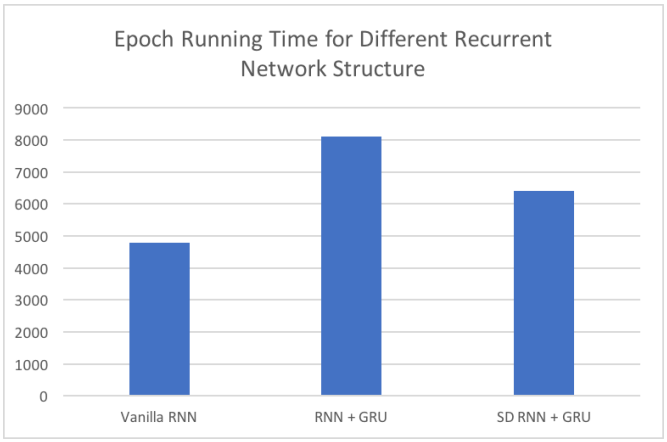

Figure 2. This figure compares the running time performance of the implementations of RNN. RNN + GRU model outperformed other models by reducing running time by 20%.

Dataset is analysed and a few historical models are implemented, stochastic depth model is added onto a Recurrent Neural Network while implementing the model for this project. The first primary motive to build the stochastic depth model to analyse running time of the model once with stochastic depth and to check whether it can be improved and the second motive was to check if there is a better model in terms of accuracy of test data that can be generated which also performs well with variations in training data. Firstly, we put focus on running time of the model. This objective is easy to achieve. Stochastic depth RNN takes less time than the baseline model. Survival probability is the main factor to deice the range of the speedup. Figure 2 shows a chart depicting the running time of multiple models, where stochastic depth added to the GRU model reducing the running time.

Survival probability was set to 50% in this case. While run time is taken down to 50% when survival probability is reduced to 25%. A reasonable training set accuracy was achieved (~80%), but it was slightly difficult to achieve the model to make any noticeable progress in order to show, acceptable generalization error on the development set. Keen observations yielded some interesting results. When survival probability got too low, the performance dropped off. Yet reasonable values were added to the performance in a specific range of probabilities, where the model was able to perform much better than the Naïve Bayes model. As shown in figure 3, model performance against the survival probability.

An unforeseen phenomenon was noticed, model training often converged much slowly than rest other baselines.

Even after many iterations, dev set loss could be seen going down on the

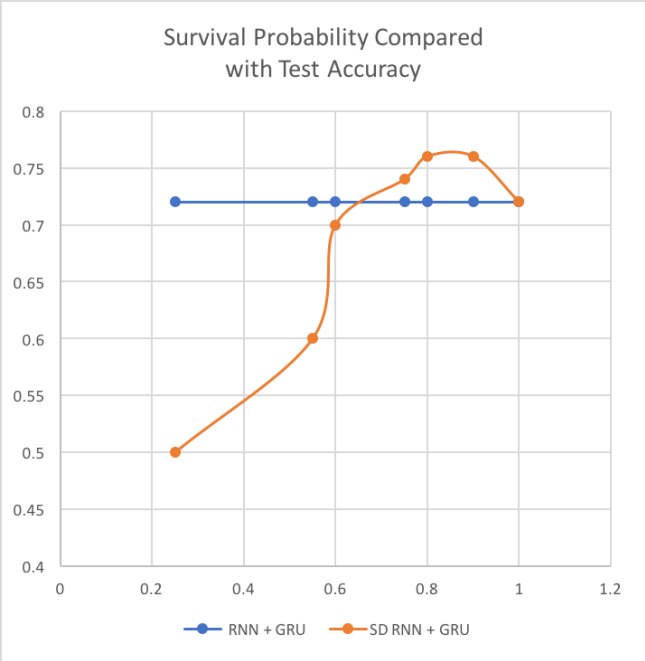

Figure 3. This figure compares survival probability performance of RNN+GRU and SD-RNN+GRU. Noticeably, our model outperformed Naïve Bayes model, because of lack of stochastic depth for small-scale probabilities in the same.

contrary for the same iteration on the non-stochastic model loss on the development set was steadily increasing. The reason for this could be the stochastic nature of the model and
that search space is huge in the ensemble models, and each pass through the data can be very different. Running for more iterations almost counteracts any gains that were achieved by saving time due to smaller networks.

Sentence Level Binary Classification		
Classifier	Train Accuracy	Test Accuracy
Naïve Bayes	74.1 %	72.9 %
RNN - Vanilla	80.1 %	73.1 %
RNN - GRU	79.4 %	75.9 %
RNN – GRU + SD	81.3 %	76.8 %

Table 1. The above table expresses the results of baselines and proposed model.

On comparison, the performance of the Naive Bayes model was reasonably well. Naive Bayes model can even detect negation to a small extent, which was a bit amazing. e.g., in the sentence “He was not happy”, was correctly identified in negative class, when intuitively this could be identified in positive class in bag of words model because of the presence of the word “happy” (happy is positive). Nonetheless, it looks positivity of the word “happy” has been overcome by the at least slightly negativity and negativity enough of the word “not”, hence it rightly labels the sentence as negative sentence. However, there were multiple observations where ensemble model performed better than the baseline Naive Bayes model to identify negation properly. For example, the phrase “Not a bad system at all”, Here we can clearly see the presence of negative sentiment word in “bad”, but notably, it is followed by the word “not”. In the above example ensemble model is able to identify correctly as positive sentiment, but baseline Naive Bayes model incorrectly identified it as negative sentiment. Similarly, “disappointing but not really sad”, has the word “sad” which contains a negative sentiment as a standalone word. Here, the negation makes it a positive sentence, which this model can capture and forecast.

If it is permitted to select the data from which it learns machine learning algorithms accuracy can be enhanced even with lesser training labels. An active learner may create queries, usually in the form of unlabelled data instances to be labelled by an oracle (e.g., a human annotator) [8]. We required phrase level sentiment tagging for our dataset. This tagging is very cumbersome and expensive, so an active learning approach is suitable.

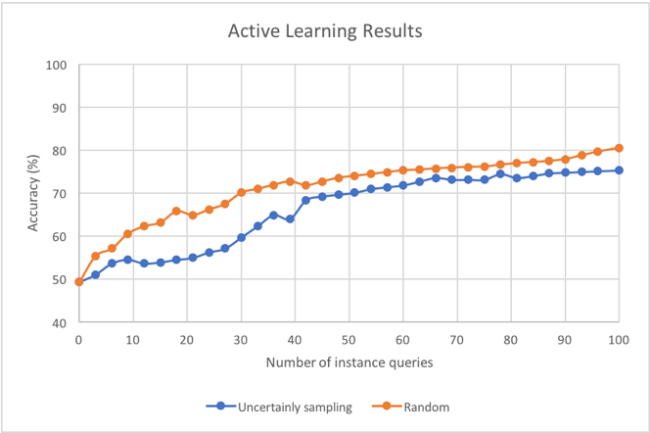

Figure 4. Here, classification accuracy is expressed as function of the number of documents queried for two selection strategies: Uncertainty sampling and Random sampling.

To generate queries for oracle to label, we use Uncertainty Sampling technique by Lewis and Gale, 1994[4]. Since our model is one of probabilistic learning, we find the data points which produce results with the lowest posterior probabilities. To do this, we query the instances for whom are model least confident:

$$x_L^* = \text{argmax}_x \, 1 - P_\theta(\hat{y}|x) \ldots (13)$$

where $\hat{y} = \text{argmax}_y P_\theta(y|x)$, is the class label with the highest posterior probability for the model θ. We made 100 such queries on our final RNN – GRU + SD model which lead to an increase an accuracy of about 4.08%.

V. CONCLUSION

We compared various models for sentiment analysis and propose Stochastic Gated Recurrent Unit (SD + GRU) based on active learning as it performs significantly better than Naive Bayes, vanilla gated recurrent unit. We have applied uncertainty sampling querying approach which is clearly superior to random sampling as seen in figure 4. Since we are using recursive models, we can do a lot of hyper-parameter tuning to get better results. Sentiment analysis has many applications in public opinion, customer satisfaction, market value etc. This work can be extended towards expression of emotions as huge amount of data from social media channels and review platform can be obtained. This would also be helpful to avoid overfitting and accuracy would further increase.

VI. REFERENCES

[1] Bojanowski, Piotr, et al, *"Enriching word vectors with subword information"*, arXiv preprint arXiv:1607.04606 (2016).

[2] Huang, Gao, et al, *"Deep networks with stochastic depth"*, European Conference on Computer Vision. Springer International Publishing, 2016.

[3] Socher, Richard, et al, "*Recursive deep models for semantic compositionality over a sentiment treebank*", Proceedings of the 2013 conference on empirical methods in natural language processing. 2013.

[4] Lewis, David D., and William A. Gale, *"A sequential algorithm for training text classifiers"*, Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval. Springer-Verlag New York, Inc., 1994.

[5] McCallum, Andrew, and Kamal Nigam, "*A comparison of event models for naive bayes text classification"*, AAAI-98 workshop on learning for text categorization. Vol. 752. 1998.

[6] Mikolov, Tomas, et al, *"Efficient estimation of word representations in vector space"*, arXiv preprint arXiv, pp.1301.3781 (2013).

[7] Mikolov, Tomas, et al, "*Recurrent neural network based language model*", Interspeech. Vol. 2. 2010.

[8] Settles, Burr, "*Active learning literature survey*", University of Wisconsin, Madison, Vol.52, pp.55-66, 2010.

Authors Profile

Mr. Tulsi Jain received his Bachelor of Technology degree from Indian Institute of Technology Delhi in the year 2015. After graduation, he joined Oracle Corporation as an application developer. At present, he is pursuing research in the field of Artificial Intelligence, Natural Language Processing and Computer Vision.

Mr. Kushagra Agarwal pursued Integrated Master of Technology from Indian Institute of Technology Delhi from 2011-2016. He is currently pursuing research in the field of Applied Machine Learning, Natural Language Processing and Computer Vision.

Mr. Ronil Pancholia pursued Master of Science (Technology) from Birla Institute of Technology and Science Pilani from 2012-2016. He is currently working in the field of Applied Machine Learning, specifically Natural Language Processing and Computer Vision.

