Online Social Network Analysis - CS579
Fall 20

Final Report

November 30, 2020

Fake New Classification

Tushar Nitave (tnitave@hawk.iit.edu) - A20444211
Varun Shanbhag (vshanbhag1@hawk.iit.edu) - A20452310

Introduction

Overall idea of the project it to classify the given the title of a fake news article A and the title of a coming news article B, into three categories:

Agreed: B talks about the same fake news as A.
Disagreed: B refutes the fake news in A.
Unrelated: B is unrelated to A.

We try to understand the semantic meaning of the given text article and then try to classify in one of these 3 categories. We have discussed how we plan to go about this in the method section.

Related Work

Tremendous amounts of research work has been carried out in order to precisely classify the fake news in the last decade. In [1] an research survey was conducted on state-of-the-art technologies for classifying fake news. A hybrid approach which uses linguistic cue and machine learning was found to be useful. A thorough review of fake news detection on social media, data mining algorithms and evaluation metrics was carried out in [2]. In [3] authors proposed Event Adversarial Neural Network (EANN) which uses even-invariant features for fake news detection. An sentence-comment co-attention-sub-network was developed to leverage news contents as well as comments for classification of the fake news. A novel approach which combines string similarity features along with a deep neural network (bi-directional Recurrent Neural Network (RNN)) was proposed. This proposed model outperformed the previous state-of-the-art models.

The provided dataset is highly imbalanced which makes predictions very inaccurate. There are several methods and strategies proposed in literature for making the dataset balanced. In [6] authors showed that synthetic minority oversampling technique (SMOTE) is beneficial for low-dimensional data.

Preprocessing and Feature Engineering

The provided train.csv dataset consists of 256442 data points along with 4 features namely *t_id1, t_id2, title1_en, title2_en.* The label consists of three classes namely *agree, disagree* and *unrelated.* The provided dataset is highly imbalanced which can be seen in the figure 1. So in order to make the dataset balanced we used Random Upsampling to

increase the frequency of ***agreed*** class to that of 175000 data points and ***disagree*** class to 175000 data points.

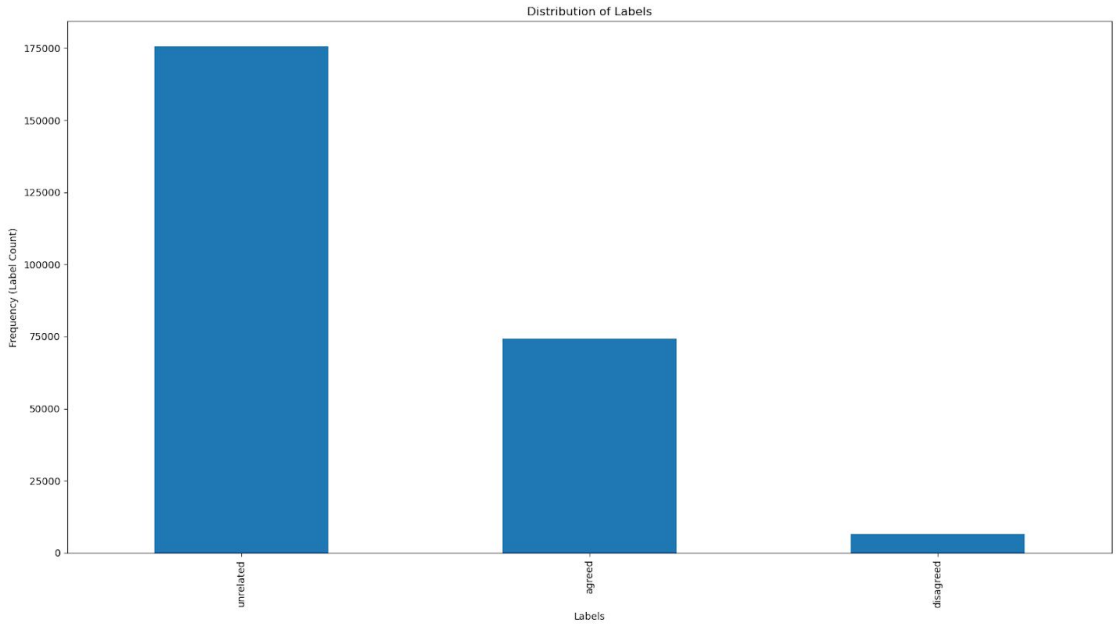


Fig.1 Class frequency in train.csv

Stemming and Lemmatization

We scan through text to remove stop words(A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.) and do stemming(Stemming is the process of producing morphological variants of a root/base word.) All this processed text is used for feature extraction. We also remove null values if any(none in this case)

Universal Sentence Encoding

It encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks.

The model is trained and optimized for greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The universal-sentence-encoder model is trained with a deep averaging network (DAN) encoder.

Cosine Similarity

It measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction in order to establish similarity between given titles.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Model

In this section we talk about different types of neural network architecture we trained to generate the desired results. We also briefly describe our approach of random forest and xgboost.

Random Forest and XGBoost

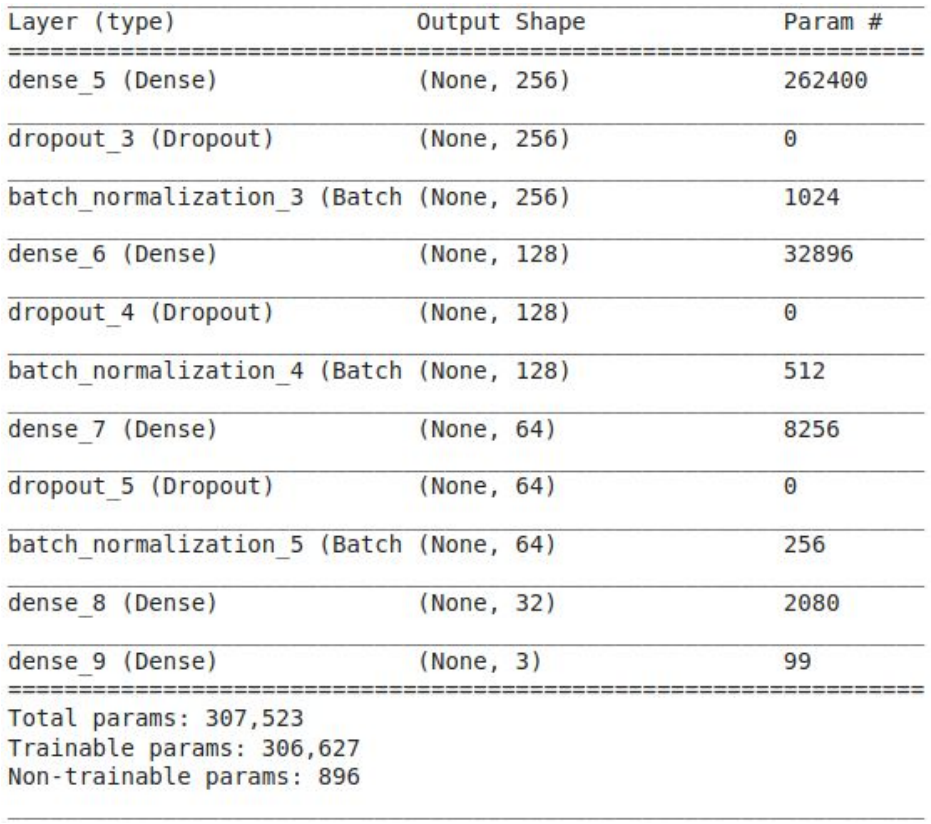
Bagging and bootstrap aggregation is a strategy which is widely used to reduce the variance of a function. Random forest is a modification of bagging that builds a large collection of de-correlated trees and then takes their average. It can be used on a variety of problems and the performance of random forests is similar to that of boosting also they are simpler to train and tune.

XGBoost is a gradient boosting technique which stands for extreme gradient boosting. It is really fast as compared to other gradient boosting methods. Underneath the xgboost uses a gradient tree boosting algorithm which produces a prediction model in the form of an ensemble of weak prediction models like decision trees.

But the performance of these two algorithms was not good for the scope of this project. So, we approached with more complex neural network based methods.

Simple Feedforward neural network

It is the simplest architecture of artificial neural networks which forms a dense network of neurons in hidden layers. It is called feedforward as the information flows only in forward direction. The fig. 2 shows the model summary of the simple feedforward neural network used by us. It was trained for 10 epochs with *adam* optimizer and batch size of 128. However the performance of this architecture was not promising because it was not able to understand the pattern which creates similarity between the two news titles. As it is a simple model it consists of only ~300K trainable parameters and all the units in hidden layers consists of *ReLU* activation whereas output layer consists of *softmax* activation as it is a 3-class classification task.



Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 256)	262400
dropout_3 (Dropout)	(None, 256)	0
batch_normalization_3 (Batch	(None, 256)	1024
dense_6 (Dense)	(None, 128)	32896
dropout_4 (Dropout)	(None, 128)	0
batch_normalization_4 (Batch	(None, 128)	512
dense_7 (Dense)	(None, 64)	8256
dropout_5 (Dropout)	(None, 64)	0
batch_normalization_5 (Batch	(None, 64)	256
dense_8 (Dense)	(None, 32)	2080
dense_9 (Dense)	(None, 3)	99

Total params: 307,523
Trainable params: 306,627
Non-trainable params: 896

Fig. 2 Simple feedforward neural network model summary

Next, we discuss the use of Bi-LSTM which performs extremely well for sequence data and it also has the capability to detect implicit similarity between sentences.

Bi-LSTM

The neural network discussed above does not have any memory and thus they fail to retain the information in the sequence based input data. Bi-LSTM also known as bidirectional long short term memory is a type of recurrent neural network which

processes sequential input data by iterating through the sequence and retaining a state it has seen so far. The fig 3. Shows the basic architecture of RNN. At every timestep *t* the model also takes into consideration the information processed at timestep *t-1*.

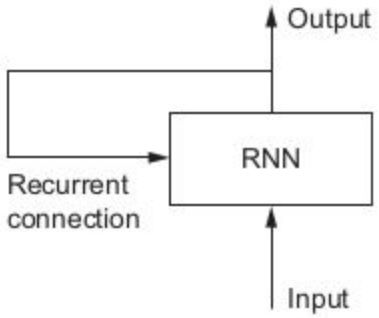


Fig 3. Recurrent neural network

The basic LSTM goes only in one direction whereas Bi-LSTM processes input data in both (forward and backward) directions. This is the reason why Bi-LSTM shows good performance in the case of our project. The downside of using Bi-LSTM is that it uses more number of trainable parameters than LSTM and simple feedforward neural network so it requires more training time and the fig. 4 shows that the our Bi-LSTM consists of more than ~ 12 million trainable parameters.

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional	multiple	6295552
dropout_6 (Dropout)	multiple	0
batch_normalization_6 (Batch	multiple	4096
bidirectional_1 (Bidirection	multiple	6295552
dropout_7 (Dropout)	multiple	0
dense_10 (Dense)	multiple	131200
dropout_8 (Dropout)	multiple	0
dense_11 (Dense)	multiple	16512
dense_12 (Dense)	multiple	387

Total params: 12,743,299
Trainable params: 12,741,251
Non-trainable params: 2,048

Fig 4. Bi-LSTM model summary

In our case the performance of Bi-LSTM was good but we got much better performance at much less training time using convolution network which we will discuss in the next section.

CNN + Dense

CNN is also known as a convolutional neural network which shows promising results for the NLP task. CNN is pretty good at recognizing patterns across the space which is very important in our case. The convolutional filters fire when they detect a specific pattern in the input pattern. Different layers of CNN are good at detecting different types of intricate hidden patterns. Also the *maxpooling* layers allows us to reduce the number of trainable parameters while retaining the useful information. This increases the performance of the model by reducing the training time. In our case the CNN layers are followed by fully connected dense layers. Fig. 5 shows the model summary of our CNN + Dense model. It is evident that performance of this model is better than previously discussed Bi-LSTM architecture and it has only ~ 579K trainable parameters.

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 1, 256)	262400
max_pooling1d (MaxPooling1D)	(None, 1, 256)	0
batch_normalization_7 (Batch	(None, 1, 256)	1024
dropout_9 (Dropout)	(None, 1, 256)	0
conv1d_1 (Conv1D)	(None, 1, 512)	131584
max_pooling1d_1 (MaxPooling1	(None, 1, 512)	0
batch_normalization_8 (Batch	(None, 1, 512)	2048
dropout_10 (Dropout)	(None, 1, 512)	0
conv1d_2 (Conv1D)	(None, 1, 256)	131328
max_pooling1d_2 (MaxPooling1	(None, 1, 256)	0
batch_normalization_9 (Batch	(None, 1, 256)	1024
dropout_11 (Dropout)	(None, 1, 256)	0
dense_13 (Dense)	(None, 1, 128)	32896
batch_normalization_10 (Batc	(None, 1, 128)	512
dropout_12 (Dropout)	(None, 1, 128)	0
dense_14 (Dense)	(None, 1, 128)	16512
dense_15 (Dense)	(None, 1, 3)	387

Total params: 579,715
Trainable params: 577,411
Non-trainable params: 2,304

Fig 5. CNN and Dense model summary

CNN Concat

This approach has proved to be most efficient for our project. Instead of processing both the new titles jointly we pass both news titles separately to convolutional nets and then concatenate them together and then feed to the fully connected layer. Fig. 6 shows the architecture of this model. As this model uses CNN followed by dense layers, training is extremely fast and also it produces good results. Fig 7 the model summary and it consists of ~4 million trainable parameters.

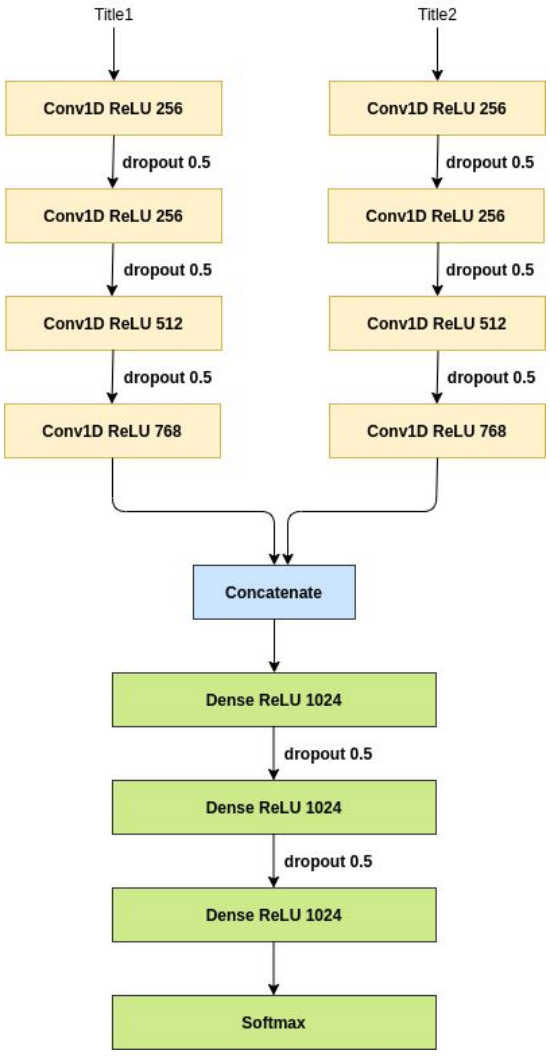


Fig. 6 CNN Concat model architecture

Layer (type)	Output Shape	Param #	Connected to
input_5 (InputLayer)	[(None, None, 512)]	0	
input_6 (InputLayer)	[(None, None, 512)]	0	
concatenate_2 (Concatenate)	(None, None, 1024)	0	input_5[0][0] input_6[0][0]
dense_8 (Dense)	(None, None, 1024)	1049600	concatenate_2[0][0]
batch_normalization_34 (BatchNo	(None, None, 1024)	4096	dense_8[0][0]
dropout_34 (Dropout)	(None, None, 1024)	0	batch_normalization_34[0][0]
dense_9 (Dense)	(None, None, 1024)	1049600	dropout_34[0][0]
batch_normalization_35 (BatchNo	(None, None, 1024)	4096	dense_9[0][0]
dropout_35 (Dropout)	(None, None, 1024)	0	batch_normalization_35[0][0]
dense_10 (Dense)	(None, None, 1024)	1049600	dropout_35[0][0]
batch_normalization_36 (BatchNo	(None, None, 1024)	4096	dense_10[0][0]
dropout_36 (Dropout)	(None, None, 1024)	0	batch_normalization_36[0][0]
dense_11 (Dense)	(None, None, 1024)	1049600	dropout_36[0][0]
dense_12 (Dense)	(None, None, 3)	3075	dense_11[0][0]

Total params: 4,213,763
Trainable params: 4,207,619
Non-trainable params: 6,144

Fig. 7 CNN Concat model summary

There were other model architectures such as ResNet, Inception block, CNN Concat followed by Bi-LSTM which could have proved better and yielded better results. Experimenting with these architectures remains unexplored for the scope of this project.

In the next section we will talk about the results obtained from these above mentioned neural network architectures.

Ensemble

The final results of our predictions is the output of the ensemble architecture shown below in the fig 8. It takes predicted results from three different models and produces the final predictions based on the max output. This approach has proved extremely for this task. The final accuracy obtained from the ensemble model is **~86%** on the test data set provided on kaggle.

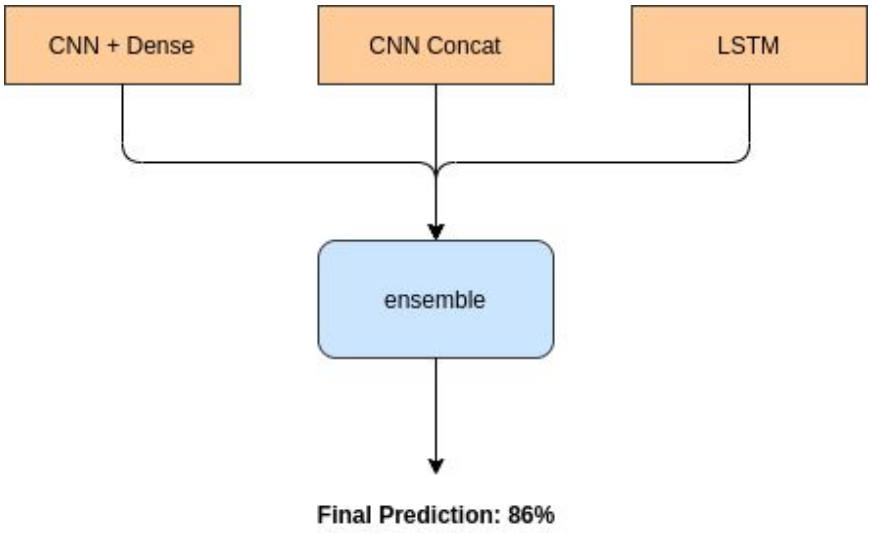


Fig. 8 Ensemble model architecture

Result

We used 10K samples for validation and remaining data for training the model. All the models mentioned before were trained using *adam* optimizer with batch size of 128 for 10 epochs.

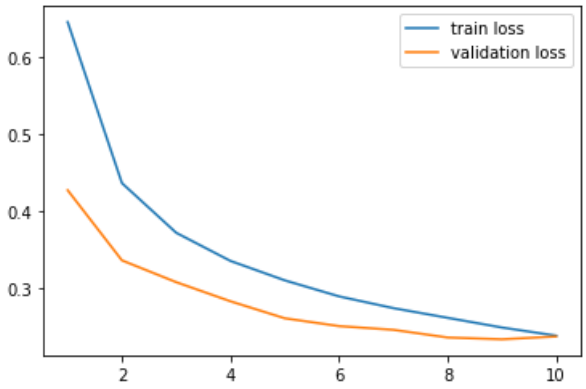


Fig. 8 (a) Simple feed forward loss

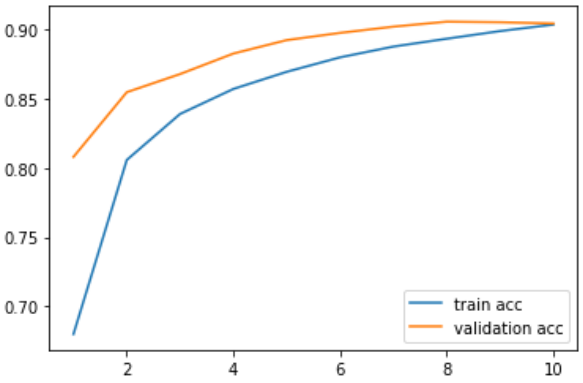


Fig. 8 (b) Simple feed forward acc

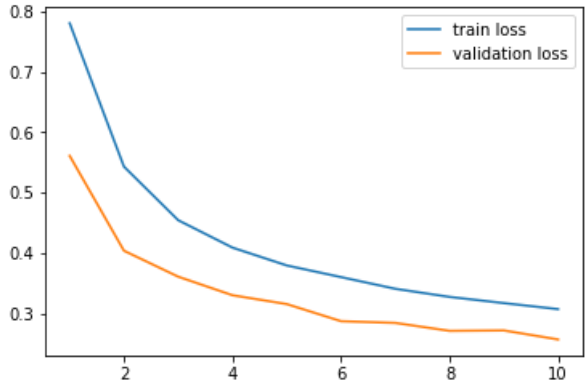


Fig. 9 (a) CNN + Dense loss

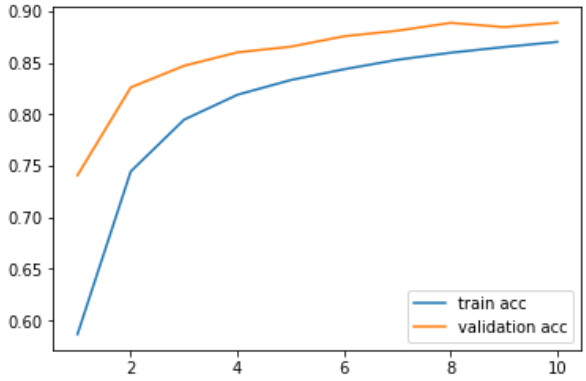


Fig. 9 (b) CNN + Dense acc

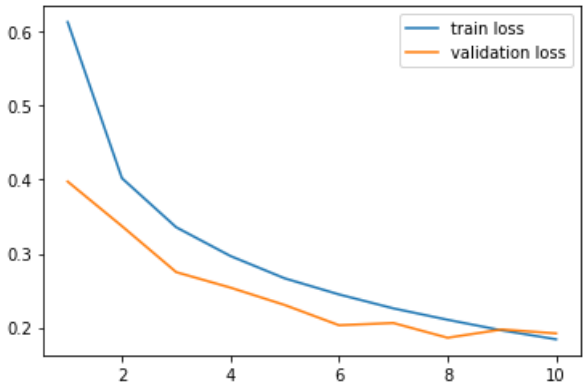


Fig. 10 (a) CNN + Concat loss

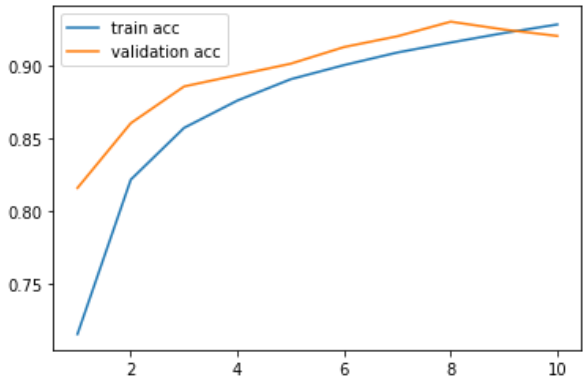


Fig. 10 (b) CNN + Concat acc

Method	Accuracy (%)		
	Train	Val	Test (Kaggle)
Feed forward	88.33	89.042	76.509
Feed forward + added features	90.33	90.42	80.05
LSTM	84.87	86.17	78.037
Bi-LSTM	91.87	92.17	81.882
CNN + Dense	87.01	88.86	81.289
CNN + Concat	**93.98**	**93.74**	**84.908**

Word2Vec	75.23	78.34	77.234

Table 1 Accuracy on train, test and validation for different proposed architectures.

Group Justification

Idea for this project was to approach with an easy solution and gradually build more complex models. We initially started with a Random Forest model and took that as our base then we collaboratively discussed and decided how to improve upon this and trained other models(as mentioned above) to improve our overall accuracy. Most of the time we worked together as one of us doesn't have a GPU, however, we independently looked upon ways to further improve our base model.

We also individually trained different neural network architectures by performing different parameter tuning.

Source Code

The source code for our project can be found at the following GitHub repository. It consists of all the code required for training the model, inferencing, preprocessing and feature engineering. It also contains the final results predicted on test.csv

https://github.com/tushar-nitave/fake-new-classification/

References

1. Conroy, N. K., Rubin, V. L., & Chen, Y. (2015). Automatic deception detection: Methods for finding fake news. Proceedings of the Association for Information Science and Technology, 52(1), 1-4.
2. Shu, Kai, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. "Fake news detection on social media: A data mining perspective." ACM SIGKDD explorations newsletter 19, no. 1 (2017): 22-36
3. Wang, Y., Ma, F., Jin, Z., Yuan, Y., Xun, G., Jha, K., ... & Gao, J. (2018, July). Eann: Event adversarial neural networks for multi-modal fake news detection. In Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining (pp. 849-857).
4. Shu, K., Cui, L., Wang, S., Lee, D., & Liu, H. (2019, July). defend: Explainable fake news detection. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (pp. 395-405).
5. Borges, L., Martins, B., & Calado, P. (2019). Combining similarity features and deep representation learning for stance detection in the context of checking fake news. Journal of Data and Information Quality (JDIQ), 11(3), 1-26.
6. Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. Journal of artificial intelligence research, 61, 863-905.
7. https://tfhub.dev/google/universal-sentence-encoder/4
8. https://en.wikipedia.org/wiki/Gradient_boosting
9. https://en.wikipedia.org/wiki/XGBoost
10. https://web.stanford.edu/~hastie/ElemStatLearn/
11. https://towardsdatascience.com/text-classification-rnns-or-cnn-s-98c86a0dd361
12. Wu, H., & Gu, X. (2015, November). Max-pooling dropout for regularization of convolutional neural networks. In *International Conference on Neural Information Processing* (pp. 46-54). Springer, Cham.
13. https://github.com/Cisco-Talos/fnc-1