Fake News Detection

Final Report

Debjyoti Roy
112070373

Priyanka Bedarkar
112046765

Siddarth Harinarayanan
112026390

Abstract

Fake news, deliberate disinformation, hoaxes, parodies and satire are various ways to mislead people in order to damage an agency, entity, or person, and/or gain financially or politically. Of late, fake news has been in the spotlight of mainstream journalism and general public because of how it can have an effect on the political scenario of a country(Fake News). In this project, we attempt to detect authenticity of specifically political news. We work to improve upon the existing model of Wang(2017), inculcating various ideas we learned during the course of the subject.

1 Introduction

Our main aim of the project is detection and classification of fake news. The primary channel for spreading such content is the social media and it sometimes finds its way into the mainstream media as well. Day by day it is becoming increasingly important to detect and classify fake news as such, because of the grave impact it can have on the political results of an election. The primary challenge for solving the issue of fake news is how loose the definition of the term Fake news is. For e.g. fake news can be classified into various categories: a statement which is known to be completely false, or a speech stating some statistics as facts for which no real analysis has been done, or a piece of text which is satirical.

Hanselowski et al.(2018) uses fake news challenge dataset which classifies the news based on four classes namely, agree, disagree, discuss, and unrelated. The model is trained using two stacked LSTM for embedded token sequence and three layered neural network to estimate the probability of which class it belongs to. This paper uses only LSTM based model to get the test predictions. Kim(2014) prominently discusses the idea of sentence classification using CNN and max-over-time-pooling. They have utilized dropout on the penultimate layer with l2-norm constraint of weight vectors for regularization. Wang(2017) proposes a solution that involves convoluted neural network for news statement, and bidirectional LSTM for other features of the news such as speaker, venue, etc. The hidden layers of a CNN typically consists of convolutional layers, pooling layers, fully connected layers, and normalization layers.

The ideas mentioned above fail to take into consideration the temporal and syntactic content of the statements. For e.g., Fake news sentences generally have a complex dependency parse as they tend to include a lot of phrases and punctuations. So, we have used POS tags of the sentences and the Dependency parsing of the sentence to try to include these information while training the models.

We tested these ideas by fact checking in the training dataset. We have used the LIAR dataset presented in Wang(2017). The baseline model that is presented in the same paper uses LSTM and CNN for training text embeddings and metadata dense features for

training the model.
Main outcomes:

- We implemented the baseline model of using Bi-LSTM.
- We developed a new addition on top of Bi-LSTM model using POS tags and Dependency Parse of the sentence.
- Using the additions, our proposed model produced better results than the baseline model.
- We also used SVM for classification but it did not improve upon the baseline model.
- Based on our experiments, we conclude that utilizing more syntactical features of the sentence for training more robust Deep Learning models might be promising avenues for future work.

2 Our Task

LIAR dataset contains 12.8K manually labeled short statements in various contexts by diverse speakers from POLITIFACT.COM. Details about the dataset is mentioned in Section 3.1. Using LIAR dataset, the task of fake news detection is now framed as a 6-way classification task into fine-grained labels for the truthfulness ratings: *pants-fire, false, barely-true, half-true, mostly-true, and true*.

2.1 Baseline Models

Our baseline model uses bi-LSTM. The input consists of two embeddings. One for the news statement and other for the meta-data. Meta-data consists of information about the speaker of the statement, the party to which the speaker belongs, job-title, the venue where the statement was made and the subject of the statement. These two embeddings are fed as input to a Deep Learning model to produce softmax probabilities over the 6 output classes. The choice of Deep Learning model varies viz. CNN, LSTM, biLSTM, LSTM+CNN (hybrid) and many more. For our baseline, we have experimented with bi-LSTM.

- **Statement Embeddings:** We used GloVe pre-trained vector representations for words. We generated the embedding matrix of (vocabulary_size x 100) dimensions wherein each word in the vocabulary is represented as a 100-dimensional vector. We use this embedding matrix as weights to generate the Embeddings for each input statement.
- **Meta-data:** For each of the Meta-data (viz. speaker, job, subject, party, state, venue) of the statement, we take top-k frequently appearing categories and club remaining infrequent categories under "others" label. Hence, we obtain a k-dimensional one-hot vector encoding for each of the meta-data. The value of k is different for different meta-data and is selected after carefully evaluating the dataset and distribution of meta-data information. For more details about the metadata encoding, refer section 2.3.

So we input the statement embedding to a bi-LSTM and the binary metadata features (83-dimensional) to a Dense layer. Finally, we concatenate these two layers to obtained richer representations and pass them to a Dense Layer which performs the classification by generating softmax probabilities over the 6 output classes.

2.2 The Issues

We tried experimenting with different set of input features (viz, only statement as the input, statement and anyone of the metadata as the input) and have analysed the results in the results section 3.4. However, using the statement embedding with all the metadata features, we obtained an accuracy of 27.78%. Our baseline using statement + metadata surpassed the best model with 27.4% mentioned in (Wang, 2017). However, in most of the fake news statements, we observed that there are syntactical features which are not incorporated in the said model. Hence, to address this

shortcoming, we encorporate POS tags and dependency parse of the statement in our feature set which helps us learn richer representations of the data thus increasing the accuracy of predictions.

2.3 Our Approach

In the baseline model, our bi-LSTM uses just the statement embeddings and metadata information. Now, we modify the architecture to include POS tags of the statements and the relations given by dependency parser.

2.3.1 Idea 1 - Using richer Features

For each statement, *we remove the stopwords and clip the statement to length of 15 words*. If there are fewer than 15 words, we use post-padding to get a statement containing 15 words.

- **Statement Embeddings :** As stated in the previous section, for each word in the statement, we use 100 dimensional word-vector representation obtained from GloVe.
- **Metadata :** For subject, we chose 14 subjects frequently appearing in the training data and clubbed all the other subjects under a common 15th label. We represented subject as 15-dimensional one-hot vector. Similarly, we represented party as 6-dimensional, state as 17-dimensional, venue as 14-dimensional, job as 13-dimensional and speaker as 18-dimensional one-hot vectors. So, in total, we represented the metadata as a binary vector of 83-dimensions. Note that mapping the values to classes is not always an exact matching. We used string matching to map similar identifiers together. Example, 'television interview' and 'tv interview' fall under the same broad category and must be given the same label.
- **POS Tags :** For each statement (without removing the stopwords), we generated the POS tags for each word. We selected top 9 frequently occuring POS tags (NOUN, VERB, ADP, PROPN, PUNCT, DET, ADJ, NUM, ADV) from the training set as individual labels and clubbed the remaining tags under a common label X. Further, we represented each POS tag as a 10-dimensional one-hot vector. We created a POS embedding matrix of (10 x 10) dimensions which basically is an identity matrix wherein row_i corresponds to the embedding for the i^{th} POS tag.
- **Dependency Parse Tags :** For each statement (without removing stopwords), we considered dependency parse which are frequently appearing (punct, prep, pobj, compound, det, nsubj, ROOT, amod, dobj, aux) in the training set as individual labels and other infrequent relations under one common label. Hence, each Dependency Parse relation is now represented as 11-dimensional one-hot vector. We created DEP embedding matrix of (11 x 11) dimensions which basically is an identity matrix wherein row_i corresponds to the embedding for the i^{th} DEP tag.

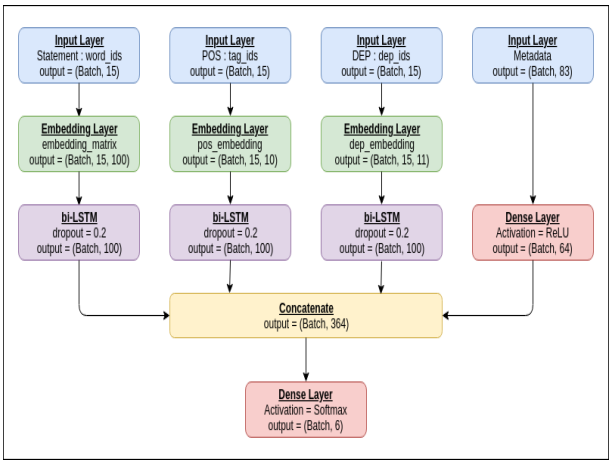

Figure 1: LSTM model architecture

For each statement, we pass it through the Embedding Layer with weights as embedding matrix. This is further passed as input to the Bi-LSTM. Similary, for each statement, we pass the POS tags through the Embedding Layer with weights as the POS embedding matrix. This is further passed as input to another Bi-LSTM. Similarly, we pass

the DEP tags through the Embedding Layer with weights as the DEP embedding matrix. This is further passed as input to another Bi-LSTM. All the three Embedding Layers are non-trainable. All the three bi-LSTMs have output layer size of 100 and a dropout of 0.2. The 83-dimensional binary feature vector of metadata is fed as input to a Dense Layer with ReLU as the activation function. We finally concatenate the outputs of these 4 above mentioned Layers and pass the output to a Dense Layer which servers as a classifier generating softmax probabilites over 6 output classes. This model is shown in Figure 1.

2.3.2 Idea 2 - Different Network Architecture

Apart from bi-LSTM, we tried two other network architectures. Firstly, we tried LSTM (non-bidirectional). However, this didnot perform at par our bi-LSTM model which makes sense since bi-LSTM is able to capture better context/relations since it gets information from forward and backward states simultaneously.

Second architecture we tried is CNN. This model is shown in Figure 2. Again for each training sample, we created three Embedding Layers - one for each of the statement embedding, POS tags embedding and DEP tags embedding. These embedding Layers are non-trainable. Now, consider the Statement Embedding Layer's output. This is simultaneously fed into three Conv1d $\rightarrow$ MaxPool1d Layers. The output of three MaxPool layers are concatenated and passed through a Dropout Layer to reduce overfitting. Output from Dropout Layer is then fed to Dense Layer. Similar layers are constructed for POS embedding and DEP embeddings. So these three embeddings are propogated simultaneously and independently. Metadata is also passed through the Dense Layer. Finally, we concatenate the output from all four paths and pass it our classifier Layer which generates softmax probabilities over 6 output classes. Analysis of how CNN performed compared to bi-LSTM is in the Results section 3.4.

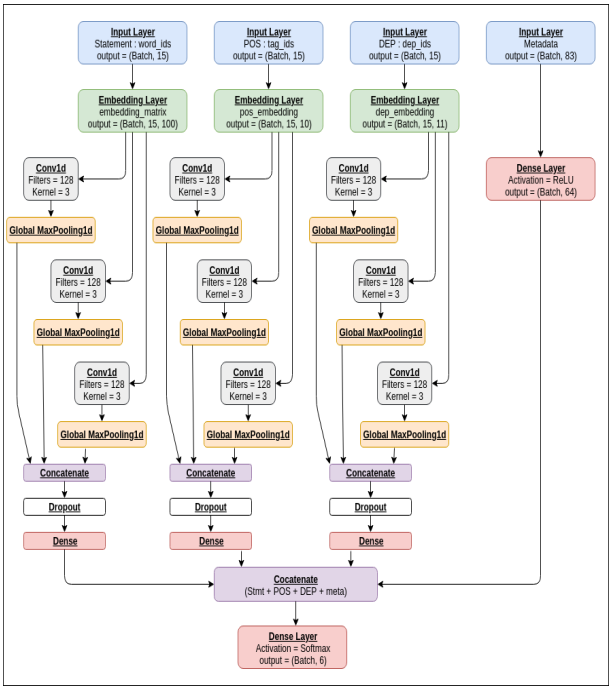

Figure 2: CNN model architecture

2.3.3 Idea 3 - Hybrid Model

CNNs are in general good at dealing with spatial features whereas LSTMs are good at temporal. So we came up with the hypothesis that there will be some features that CNN would be capturing better and some features that LSTM would be capturing better which may or may not be captured by the CNN. On the other hand, the metadata is 83-dimensional binary vector where at a time only 6 out of 83 values will be 1s and others would be 0s (since we have 6 metadata viz. speaker, subject, job, venue, state, party and for each of these 6, one of its category will have value 1). The metadata vector is indeed a *sparse representation*. So, we employed the power of Machine Learning - by training a SVM model to let it learn mapping from metadata (83-dim binary) to output labels (6-dim - pants-fire, false, barely-true, half-true, mostly-true, and true). At this stage, we have 3 models viz. bi-LSTM, CNN, SVM that outputs the predictions. Each of these model are capable of learning representations which the other model may not be learning. We pass the input training sample to each of these three models and then train another MultiLayer Percep-

tron over the predictions generated by these three models. The idea was that MLP will learn how much weight to give to each of the three models' predictions and then generate a final output label. However, the accuracy of this Hybrid model was not even close to that of the baseline model. This was not obvious at first, but could be attributed to one of the three models being a bottleneck in the training process. If good performance of one model can improve the performance of the overall hybrid model, the poor performance of the same model can bring down the performance of the overall hybrid model too.

2.3.4 Implementation Details

- **Inputs :** For each training data, we clipped the statement to be of 15 words. Each of the 15 words would be represented by 100-dimensional vector representation as from GloVe. Statement-embeddings will have (None, 15, 100) shape. For each such statement, we will have 15 POS tags (one for each of the 15 words) wherein each tag is a 10-dim one hot vector. The pos_embedding will have (None, 15, 10) shape. And similarly, dep_embedding layer will be producing output of shape (None, 15, 11).

- **LSTM :**
  - embedding_dim = 100
  - hidden_size = 100
  - lstm_size = 100
  - num_steps = 15 : words in a stmt
  - num_epochs = 30
  - batch_size = 40
  - dropout at bi-LSTM = 0.2
  - optimizer = SGD(lr=0.025, clipvalue=0.3, nesterov=True)
  - loss function = categorical crossentropy

- **CNN :**
  - kernel sizes for each of conv1d = 3
  - filter_size for each of conv1d = 128
  - num_steps = 15 : words in a stmt
  - num_epochs = 30
  - batch_size = 40
  - dropout at CNN = 0.6
  - optimizer = SGD(lr=0.025, clipvalue=0.3, nesterov=True)
  - loss function = categorical crossentropy

3 Evaluation

We used Accuracy as an evaluation measure. Since this dataset is a balanced one, it was observed that the accuracy results from different models were equivalent to respective f-measures. Hence, we used accuracy as the evaluation metric.

3.1 Dataset Details

LIAR dataset includes 12,836 manually labeled short statements from POLITIFACT.COM. It has annotated data for truthfulness, subject, context/venue, speaker, state, party, and prior history. With such volume and a time span of a decade, LIAR is an order of magnitude larger than the currently available resources. The fine-grained labels for the truthfulness ratings are : pants-fire, false, barely-true, half-true, mostly-true, and true. The statistics of the dataset are shown in Table 1. The distribution of labels are shown in Figure 3.

Table 1: Dataset Statistics

Training Set Size	10269
Validation Set Size	1284
Testing Set Size	1283
Average Statement Length	17.9

3.2 Evaluation Measures

Our Fake News Detetction System would generate softmax probabilities over the six output labels viz. pants-fire, false, barely-true, half-true, mostly-true, and true. The model will output the confidence value for each of the 6 labels. The one with maximum confidence is chosen to be the prediction. This is compared to the groundtruth values. Accuracy is chosen as an evaluation measure.

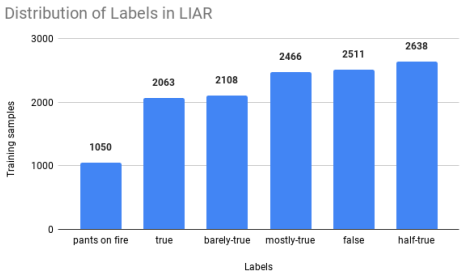

Figure 3: Distribution of labels in dataset

3.3 Hyperparameters Tuning

- **Optimizer :** Though Adam is computationally efficient and requires less tuning, for our case, we noted that LSTM model overfits very easily with Adam. Same was observed for RMSProp. For our case, SGD with a learning rate of 0.025 and with nesterov momentum worked the best.

- **Number of words clipped :** Since it was observed that average length of statement in the LIAR dataset is 17.9, we restricted each training data to have statement of length 15. We also experimented by permitting the statement to have length 20 (post-padding the shorter statements). However, the performance was not as good as what we observed with length 15. This behaviour is expected as it would be difficult for LSTM to work on bigger statements.

- **Dropout :** To reduce overfitting, we added dropout. For LSTM, we set the keep_prob = 0.2. We experimented increasing the keep_prob to 0.5. It resulted in poor performance.

- **Batch Size :** Currently we have batch size of 40. We experimented with lesser batch size of 16. This resulted in overfitting the training data over 30 epochs. Maybe the model fastly converged to a local optima.

- **Epochs :** The model mostly learns in first 25 to 30 epochs. Later it just overfits the training data or oscillates around the optima.

- **Learning Rate :** Learning rate directly affected the convergence of the model. Decreasing the learning rate, we observed the loss was decreasing very slowly. However, with increasing the learning rate, we observed random results i.e. oscillating losses. Current learning rate of 0.025 worked the best.

3.4 Results

The main results from our experiments are listed in Figure 4. We observe that Bi-LSTM with statement, metadata and dependency parse provides the best accuracy closely followed by the model using Bi-LSTM with statement, metadata, dependency parse and POS tags. We observe that just utilizing POS tags along with statements does not give good accuracies. Also, SVM does not produce comparable results to LSTMs.

Model	Features	Val Acc	Test Acc
bi-LSTM	stmt	26.09%	23.76%
bi-LSTM	pos	24.84%	22.57%
bi-LSTM	stmt + party	28.27%	24.86%
bi-LSTM	stmt + state	27.26%	25.26%
bi-LSTM	stmt + venue	27.18%	25.10%
bi-LSTM	stmt + job	26.64%	25.34%
bi-LSTM	stmt + subject	27.73%	26.20%
bi-LSTM	stmt + speaker	27.88%	25.34%
bi-LSTM	stmt + meta	29.67%	27.78%
bi-LSTM	stmt + meta + pos	30.14%	27.62%
bi-LSTM	**stmt + meta + dep**	**28.51%**	**28.41%**
bi-LSTM	**stmt + meta + pos + dep**	**29.36%**	**28.33%**
CNN	stmt	27.65%	24.55%
CNN	**stmt + meta**	**29.21%**	**27.86%**
CNN	stmt + meta + pos	28.89%	27.55%
CNN	stmt + meta + dep	28.97%	26.91%
CNN	stmt + meta + pos + dep	29.83%	27.15%
SVM	meta	27.01%	26.20%

Figure 4: Experimentation Results

3.5 Analysis

- For bi-LSTM, Passing only the statement embedding to bi-LSTM produces 23.76% accuracy. With only statement, our model is not able to learn good representations. We need richer features as input to our model.

- For bi-LSTM, Statement and information about the subject of the sentence

produced good results than statement and anyone of the other metadata. i.e. Subject information is useful for the model to predict more accurately.

- For bi-LSTM, Statement + metadata and Statement + metadata + POS gave almost the same results. Including POS did not affect the performance much.

- **For bi-LSTM, Statement + metadata + DEP produced the best results of 28.41%. This is an improvement over Wang(2017) which quotes its best model to have 27.40% accuracy** . Dependency relations were indeed important features which helped the model predict more accurately.

- For CNN, the same set of features, Statement + metadata + DEP decreased the performance. CNN are not able to use the temporal information from DEP features accurately.

- For CNN, Statement + metadata produced better results.

- For SVM, using only the metadata, suprisingly is able to perform very close to bi-LSTM with statement + metadata. This may be because the binary 83-dim data was suitable for a machine learning model to perform well on.

- The hybrid model described in Section 2.3.3 performs around 20%. Passing the predictions of bi-LSTM, CNN, SVM through a MLP to output prediction by correctly weighing the three inputs did not work well. We attributed rhis to one of the three models being a bottleneck in the training process. If good performance of one model can improve the performance of the overall hybrid model, the poor performance of the same model can bring down the performance of the overall hybrid model too.

In all, the DEP embeddings provided rich contextual and structural meaning of the statement, hence improving the prediction accuracy.

3.6 Code

The code can be found here: Github Repo. The readme file in the repo contains details about the code run parameters and comments in the code are written to explain each step.

4 Conclusions

Our bi-LSTM model with rich input features (statement + metadata + POS + DEP) outperforms the best results from Wang(2017). POS tags and DEP parse information improves the performance of the baseline models. LSTM are better than CNN for textual data. The take away from this project is that, we understood what kind of problem requires which model architecture, based on the dataset spread and metadata.

5 Sample Predictions by our model

Statement	Says President Obama told a room of students, Children, every time I clap my hands together, a child in America dies from gun violence, and then a child told him he could solve the problem by not clapping any more.	Illinois suffered 1,652 overdose deaths in 2014 ... of which 40 percent were associated with heroin
Subject	guns	drugs
Speaker	mail	richard-durbin
Job	NaN	Senator
State	NaN	Illinois
Party	none	democratic
Venue	a chain email	a press release
GroundTruth	pants-fire	TRUE
Predicted	pants-fire	TRUE
Confidence	0.7088227	0.483048
Comments	Model learnt that if the source is not known, it probably is a fake news.	When, no NaNs in the data, the model predicts label true.

Model learnt that when quoted with statistical/numerical data, news must be true.

References

Fake News. Fake news — Wikipedia, the free encyclopedia.

Andreas Hanselowski, Avinesh PVS, Benjamin Schiller, Felix Caspelherr, Debanjan Chaudhuri, Christian M Meyer, and Iryna Gurevych. 2018. A retrospective analysis of the fake news challenge stance detection task. *arXiv preprint arXiv:1806.05180*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

William Yang Wang. 2017. " liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*.