CS-GA 3033: Statistical Natural Language Processing Classifying Political Bias from U.S. Presidential Speeches

Bharathi Priyaa T
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
bt978@nyu.edu

Qiyu Xiao
Department of Mathematics
Courant Institute of Mathematical Sciences
New York University
qx344@nyu.edu

Shivam Verma
Department of Mathematics
Courant Institute of Mathematical Sciences
New York University
sv1239@nyu.edu

1 Introduction

Distributed representations (embeddings) for paragraphs and documents can be leveraged to solve problems like document classification, sentiment analysis and semantic relatedness. These are natural extensions of word embeddings, and it is believed that these approaches can better capture invariant, latent variable information across documents.

We are interested in to what extent a speech can indicate the political ideology of its author. Earlier work has been done for sentences[10]. In this paper, we generate embeddings using Paragraph Vectors as well as Hierarchical Networks for U.S. presidential speeches. Our aim is to use these embeddings to investigate if a given presidential speech document can indicate the president's political bias, in other words, how well can these speeches be classified based on their contents.

2 Related Work

Word embeddings are well-studied, and are typically generated using word2vec[1] and GloVe[2] implementations. Sentence-level embeddings[3, 4] aim to capture semantic information for natural language inference tasks. Extension of these to paragraph and document level embeddings have been created using recurrent neural networks [5,6], attention models[7] and topic-model based approaches. A. Benton et al.[8] also created embeddings for Twitter users using Generalized Canonical Correlation Analysis (GCCA).

3 Data Collection and Preprocessing

Raw text of the U.S. presidential speeches, beginning from President George Washington to President Barack Obama was obtained from the Presidential Speech Archive [2]. Since the data was not readily available, we had to scrape the website [2] using a web scraping tool [17]. Data preprocessing involved just extracting the speech embedded within certain HTML tags from the scraped documents. In aggregate, there were roughly 1000 speeches by 43 U.S. presidents, with an average of roughly 17 speeches per president.

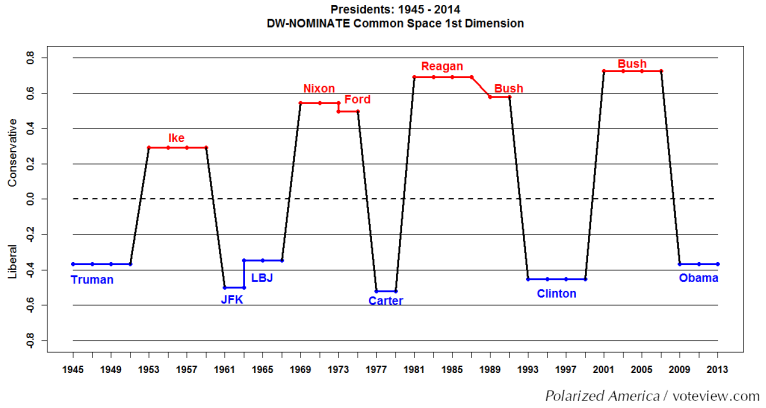


Figure 1: Results from DW-Nominate [13], a ranking system for the relative liberalism/conservatism of members of Congress, which they extrapolated to presidents using publicly stated support or opposition for legislation.

To obtain labels, i.e., numbers representing political bias, we used the parametric bootstrapped standard errors model by the Carrol, Royce et al.[13]. This statistical model is based on data collected from the U.S. House and Senate proceedings. Specifically, the House model contains scores for most U.S. Presidents, with the exception of 9 presidents. For the label, we use the 1st Dimension Coordinate from the statistical model[13], which pertains to "political bias", a floating point number ranging from -1 (very liberal) to +1 (very conservative). The labels for a few presidents is shown in Fig. 1. Since we structure the problem as a classification task, we categorized these numbers into 8 classes (where a class consists of a gap of 0.25).

The speech data is available at [15], and the labels are available at [13].

4 Models and Evaluation

We implement a number of models, including a baseline model derived from averaging word embeddings. Our main approaches to this problem are Paragraph Vectors [5] and Hierarchical Networks [7]. We also use a modified version of Hierarchical Networks, which use sentence embeddings from the skip-thought vector model [14]. Overall, we implement the following:

1. **Baseline**
2. **Paragraph vector model**
3. **Hierarchical-Mean**
4. **Hierarchical-Attention**
5. **Skipthought-Mean**
6. **Skipthought-Attention**

4.1 Baseline model

The baseline model is trained by summing the 80-dimensional word embeddings of all words present in a speech. This embedding is fed into an SVM or Multi-Layer Perceptron (MLP), with a NLL loss. The parameters for the best model were: nDim = 80, nwindow = 7, minCount = 3.

4.2 Paragraph Vector

Paragraph Vectors are distributed representations of paragraphs or documents, i.e., collection of sentences, trained in a manner similar the the original word2vec models (skip-gram and c-bow). The algorithm gives fixed dimensional embeddings for documents of various lengths, like in our case, speeches with different amounts of contents. While each document has its own vector, the

word embeddings which are trained at the same time are shared among different documents. There are two basic ways to train a Paragraph Vector, discussed in the following sections.

4.2.1 Distributed Memory

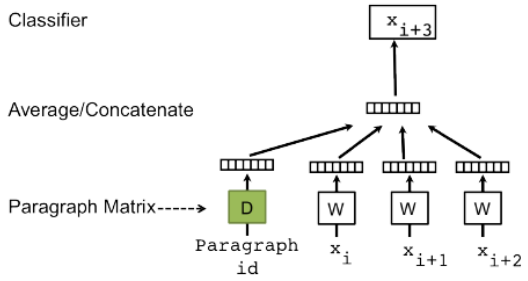


Figure 2: Distributed Memory model

This algorithm resembles the continuous Bag-of-Words implementation from word2vec [1]. In this algorithm, the memory of word vectors is 'distributed' to documents. With averaging or concatenation of local context of words and document representation, the next word in the document is predicted and updates on word and paragraph embeddings are based on minimizing this prediction loss.

4.2.2 Distributed Bag of Words

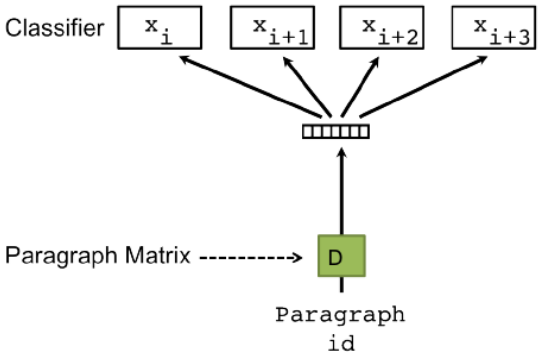


Figure 3: Distributed Bag of Words model

This approach shares the same idea as the Skip-gram Model from word2vec [1]. The word embeddings are trained first, following which the embeddings of documents are trained such that loss of prediction based on the document vector to words in it is minimized.

While Distributed Bag of Words model has less computation, its performance is not as good as Distributed Memory model in general, but not in our case. In this project, we trained both models, as well as a concatenation of them, which gave better result than either model solely.

The paragraph vector implementation involves training paragraph vectors for each speech, which are fed into an MLP, similar to the baseline model. The different models with their accuracy (for 20 epochs) are in table 1. The parameters for the best model are in table 2.

Model Parameter	Value
DISTRIBUTED MEMORY	50%
DISTRIBUTED BAG OF WORDS	74%
CONCATENATED MODEL	77%

Model Parameter	Value
EMBEDDING DIM	80
WINDOW SIZE	5
NEGATIVE SAMPLING SIZE	5
EPOCH	20

4.3 Hierarchical Neural Networks

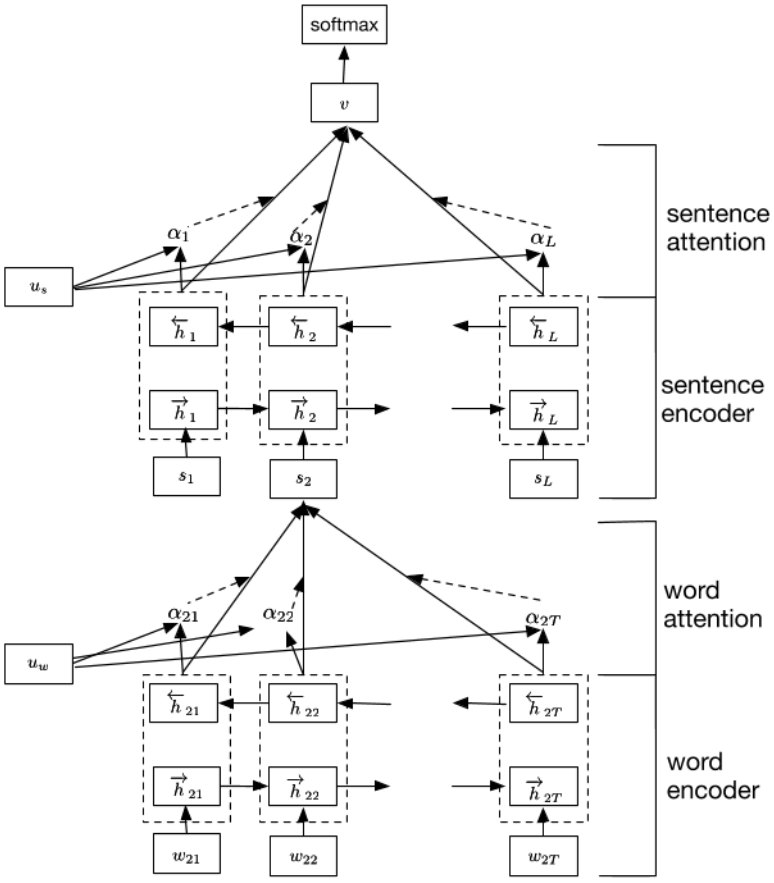


Figure 4: Hierarchical Attention Networks [7].

4.3.1 Hierarchical-Mean model

Each sentenced is trained using an LSTM, and the resulting hidden states are averaged to get a sentence vector. We worked on a hierarchical model, drawing ideas from [7] since there is no existing implementation of the paper. We have made some assumption to fit the nature of our dataset and training feasibility. The model has the following layers. We fixed the following parameters.

MAX WORDS	100
MAX SENTENCES	100
EMBEDDING DIM	50

- Input Layer : The input layer tensor is basically of the shape {number of sentences, max words x max sentences }

- Embedding layer: The input to the embedding layer is a one dimension tensor of size { max words x max sentences } and the output from the layer has the size { max words x max sentences, embedding dimensions }

- LSTM Layer : We have two layers of LSTM's each to capture word level information and sentence level information. The first LSTM is a unidirectional LSTM which gets annotations of all the words within a sentence by summarizing the contextual information of words along one direction. The LSTM layer outputs a 2d tensor for each sentence with the embedding dimension, called the word vector.

- Average Layer : We take the word vectors(denoting each sentence) from the above layer and average them along the embedding dimension. We treat the result vector as a sentence vector which encodes all the information about the words contained withing

- LSTM Layer, As before We train another LSTM layer to capture information across sentences. The output from this layer is a 2d tensor for each document representing the learned sentence vectors.

- Average Layer : We take the sentence vectors from the above layer and average them along the embedding dimension to produce a document representation which is further passed into a soft max layer for the purpose of classification

4.3.2 Performance of Hierarchical mean

- Loss function was chosen as binary cross entropy.

- We experimented with both SGD and RMSprop as optimizers. There was no significant difference in accuracies obtained

- Learning rate was fixed at 0.1. Learning rate of 0.01 made the training very slow.

- Although there was a quick convergence in training loss after Epoch 2, We noticed some oscillation in both loss and accuracies.

- Hierarchical mean gave a best accuracy of 63 % while Hierarchical sum performed a little sub-par at 50 %

- Vanishing Gradients : The longer documents/sentences and wider the network , the amount of information that can be inferred from fades away as is expected in any RNN. This is far more prominent where we have end to end training of all sentences within a documents. This was one of the main reasons we did not see any significant results

4.4 Hierarchical - Attention networks

The above hierarchical models make the inherent assumption about the treatment of words within sentences and sentences within documents. The LSTM layer which encodes contextual information about words and sentences does so across entire document. While this encodes the overall context, sentence specific or word specific contexts are not taken into consideration. To make the model more expressive, we need a stacked LSTM layer for each sequence of words. Each of these LSTM extracts only the most useful context from within the sentence. So there is one LSTM per sentence which encodes the word-level context within each sentence. While the paper [7] works on two levels of attention, we have implemented a single level attention focusing on word-sequences. The architecture of this model is given in the below figure.

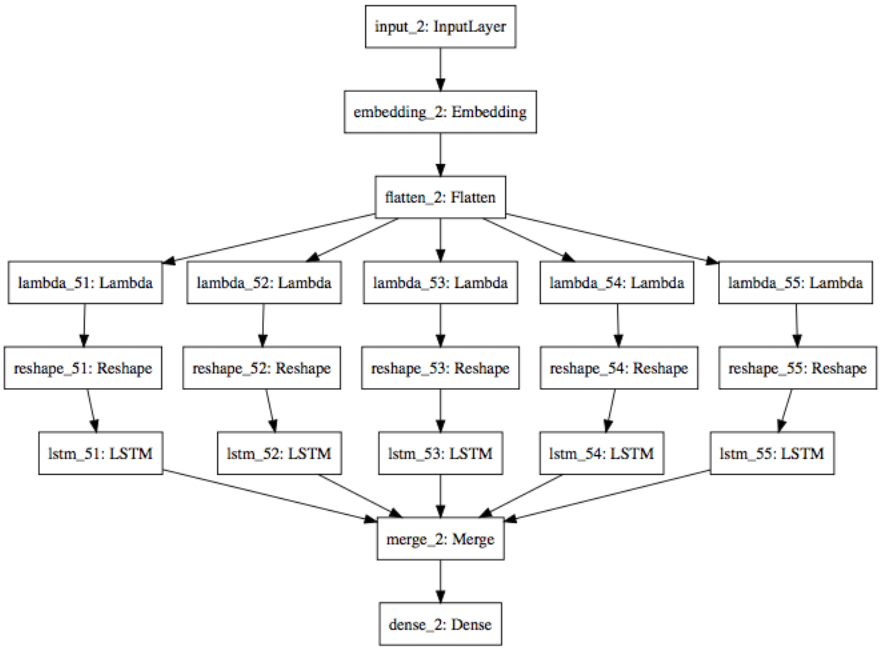


Figure 5: Computational graph for 5-Sentence Hierarchical Attention network

- Embedding layer - Pretrained glove vectors are used to construct an embedding layer of dimensions { number of sentences, number of words }
- Flatten Layer - Provides a 1 D tensor of size { number of sentences * number of words * embedding dimension }
- Stacked LSTM - Adds an LSTM layer for each sentence vector by using Lambda's to split the Flattened tensor into $MAX_{SENTENCE}$ tensors. Each of these tensors are further reshaped to represent { max words, embedding dimension} which is passed to and LSTM layer which outputs a tensor of length 128..
- Merge Layer - The output from the above LSTM layers each have a length of 128 and represent the most important words within a sentence. We have designed the LSTM layers as independent encoders with no input sharing across sentences. Adding an LSTM on top of the Merged layers connects representation across sentences and would further improve accuracies, but owing to the number of trainable parameters, we opted out it .
- Dense Layer - The penultimate layer is a dense layer which works on the combined LSTM output from previous layers to produce a 128 dimensional document representation which is passed to a Softmax layer

4.4.1 Performance of Attention network

- Number of learnable parameters : There was 11058652 parameters to be learned and only about 862 documents each with 10000 dimensions. In order to reduce the number of learnable parameters, we opted out of using Bidirectional LSTMS.
- Fixed input size : While the nature of a document could be of variable size with different parts of the documents carrying different information, for our model we had to truncate/pad the input vectors to a fixed size. This had varying amounts of sparsity across the input representation. Adding a Tanh layer on top of the sentence vectors is used to measure the importance of the sentences. This is precisely the attention part of the network .
- Adding attention to each hierarchy (words and sentences) addressed the problem of Vanishing gradient observed in the previous model.
- Adding attention improved the training accuracy to 76% from 63 % .

- The above model can be further extended to add Word-Level attention to capture the important words which capture the meaning of the sentence

4.4.2 Hierarchical Networks using Skip-Thought Vectors

Skip-thought vectors [15] is an unsupervised sentence-to-vector encoding algorithm. It uses an encoder-decoder model, where the decoder tries to reconstruct the surrounding sentences of an encoded passage. Thus, sentences with similar syntactic and/or semantic properties are mapped to similar distributed representations. We use R. Kiros et al.'s Theano code [17] to train sentence vectors for the Presidential speeches.

This approach consists of two parts:

1. **Getting sentence vectors**: The skip-thought model was used to create both forward and reverse-sentence 2400-dimensional vector representations, and these were concatenated to produce a 4800-dimensional vector for each sentence.

2. **Hierarchical Neural Network with Averaging or Attention**: Each input document/speech consisted of a different number of sentences. For this model, we restricted the speeches to 100 sentences (with padding). Similar to the second part of the Hierarchical Network (previous section), we train an LSTM on the sentence vectors, and average/apply attention to the resulting 1000-dimensional hidden states, followed by a softmax layer. This network is described in figure 5, and the corresponding Keras computational graph in figure 6.

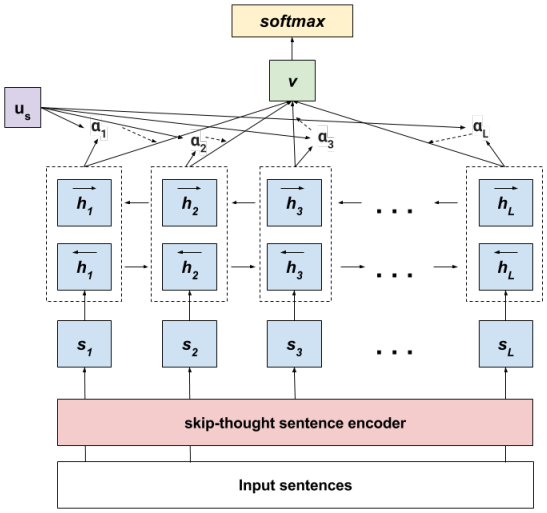
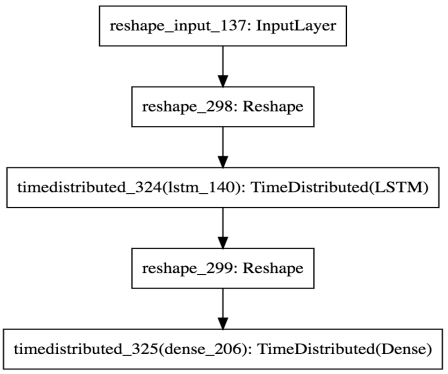


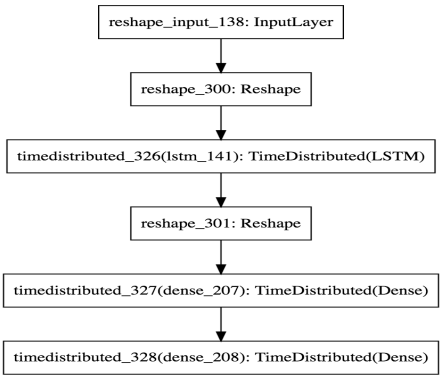
Figure 6: Combine Hierarchical Network with Skip-Thoughts Vector

5 Results

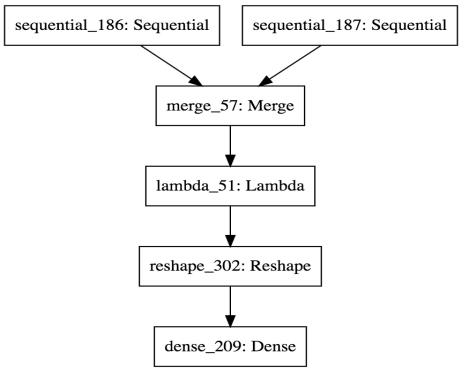
The script and ipython notebook files that produce the following results can be found at `https://github.com/reachbp/President-Speeches-Embeddings`



(a) Model 1: LSTM



(b) Model 2: LSTM with Attention



(c) Model 1 2 merged.

Figure 7: Computational graph for Hierarchical Network with skip-thought vectors model. The Lambda function indicates a weighted sum.

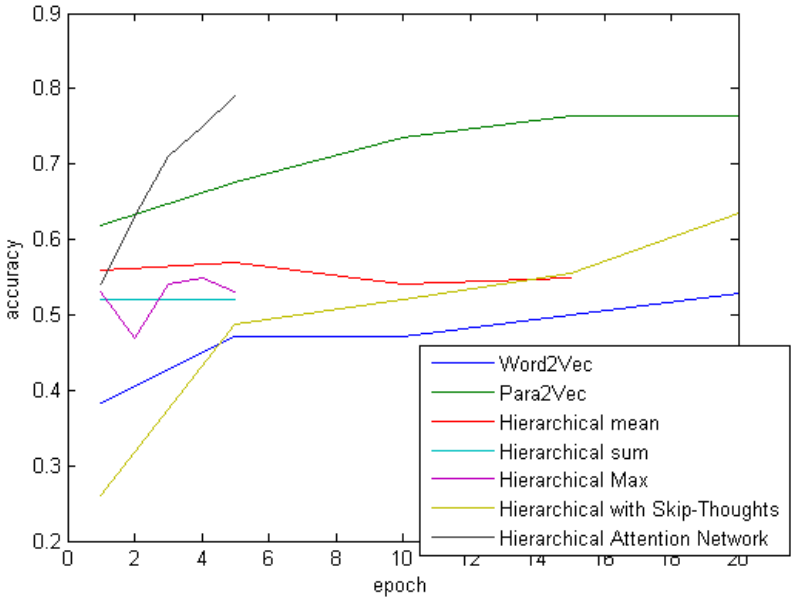


Figure 8: speech embedding convergence(test)

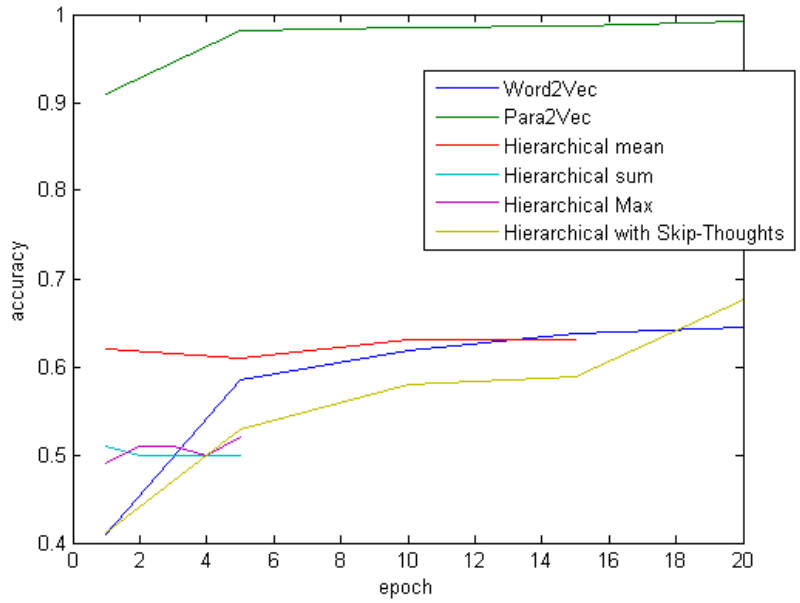


Figure 9: speech embedding convergence(training)

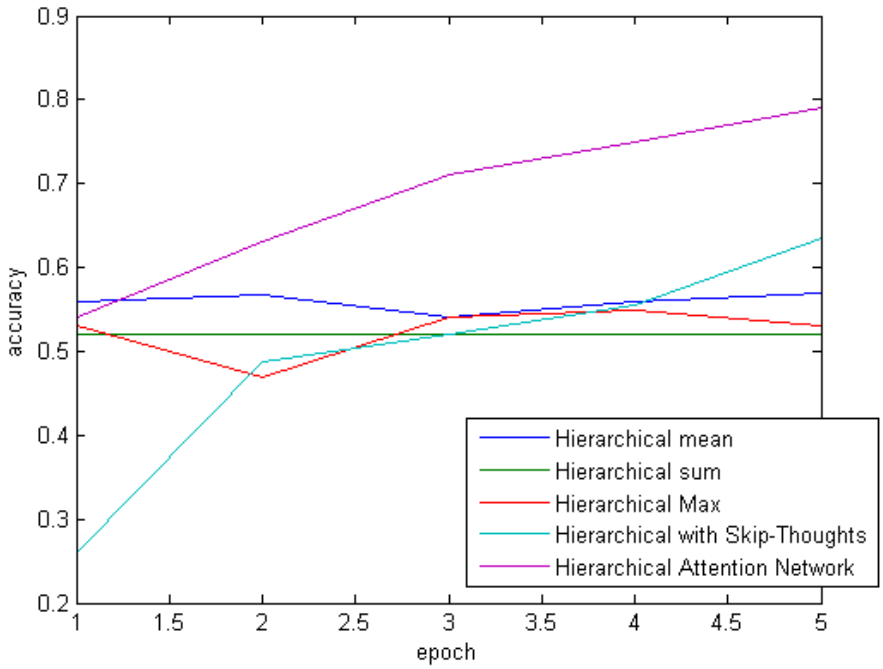


Figure 10: speech embedding convergence(training)

Figure 8 and 9 depicts the progression test and train set accuracies across 20 epochs for each of the models. Some models only have data for the first 5 epochs because those large models are time consuming and more training actually not improving the results.

Figure 10 depicts the training progression up till 5 epochs for each of the models.

The train and validation accuracies are summarized as follows:

Models	Validation Accuracy	Training Accuracy
Word2Vec(baseline)	53%	65%
Paragraph2Vec	70%	94%
Hierarchical-Mean	55%	63%
Hierarchical-Sum	52%	50%
Hierarchical-Max	53%	52%
Hierarchical-Mean + Skip-Thoughts	64%	67%
Hierarchical-Attention	65%	79%

From our experiments, we note the following:

1. The Hierarchical Networks perform up to 10% better than the baseline word2vec model.
2. The Hierarchical Network with skip-thought vectors perform better than the simpler Hierarchical Networks with the mean/max operation.
3. The Hierarchical Attention network performs better than most other networks, except the Paragraph Vector model.
4. The Hierarchical-skip-thought model with the mean operator performs almost as well as the HAN.

6 Conclusions

From our experiments with a wide variety of neural network-based models, we note that using the Hierarchical Network model, either with attention, mean/max operations or using skip-thought sentence embeddings, performs considerably better than the baseline. However, we note that the accuracy is still below the results from the Paragraph Vector implementation. This could be because the PV model incorporates context better than the mean/max Hierarchical networks, which merely average or take the maximum of the hidden states. Moreover, using skip-thought sentences as inputs, instead of training the sentence embeddings using an LSTM within the larger network leads to gains of roughly 8-10%, which is likely because the skip-thought algorithm, which uses a sentence encoding to predict its surrounding sentences, likely capturing a better representation of the context. Thus, using attention in such a model is likely to be better than the PV model. On the other hand, these results are encouraging considering that we have very few data points (800 labeled speeches) - these document classification results are similar to those obtained from [5,7] for the Yelp Reviews Dataset, which consists of nearly 2M reviews (where a maximum accuracy of 71% with the HAN model).

7 References

1. T. Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. NIPS 2013.
2. J. Pennington et al. Glove: Global Vectors for Word Representation. EMNLP 2014.
3. R. Kiros et al. Skip-thought Vectors. NIPS 2015.
4. S.R. Bowman et al. A Fast Unified Model for Parsing and Sentence Understanding. ACL 2016.
5. Q. Le, T. Mikolov. Distributed Representations of Sentences and Documents. ICML 2014.
6. A.M. Dai et al. Document Embedding with Paragraph Vectors. arXiv preprint, 2015.
7. Z. Yang et al. Hierarchical Attention Networks for Document Classification. NAACL 2016.
8. A. Benton, R. Arora, M. Drezde. Learning Multiview Embeddings of Twitter Users. ACL 2016.
9. T. Schnabel et al. Evaluation methods for unsupervised word embedding. EMNLP 2015.

10. Iyyer, Mohit, et al. "Political ideology detection using recursive neural networks." Proceedings of the Association for Computational Linguistics. 2014.
11. Benton, Adrian, Raman Arora, and Mark Dredze. "Learning multiview embeddings of twitter users." Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Vol. 2. 2016.
12. Yang, Zichao, et al. "Hierarchical attention networks for document classification." Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016.
13. Carrol, Royce et al. DW-NOMINATE scores with bootstrapped standard errors. Available at: http://voteview.com/dwnomin.htm (2011).
14. Kiros, Ryan, et al. "Skip-thought vectors." Advances in neural information processing systems. 2015.
15. `http://millercenter.org/president/speeches`
16. `https://scrapy.org/`
17. `https://github.com/ryankiros/skip-thoughts`