CLASSIFYING APHASIA-RELATED SPEECH ERRORS

A FINAL PROJECT WRITEUP

CS562 Natural Language Processing
Professor Steven Bedrick
March 2019
Kendra Chalkley

CONTENTS

1. INTRODUCTION

Aphasia, a neurogenic speech disorder caused by brain injury, affects approximately 1 in 250 people living in America. Researchers evaluate aphasia-related speech errors with tests in which patients are prompted to say a target word and their responses are recorded phonologically. In this dataset responses are written in ARPAbet, and given one of six error labels: A. Neologism, Formal, Mixed, P.R. Neologism, Semantic, and Other. The rules governing classification of errors depends on the semantic and phonological relationship between word produced by the patient and the target word. These rules can be convoluted and subject to human error. As such, an automated method of determining error label based on production and target has value both as a source of potential insight concerning the errors made as well as a tool for increasing classification data accuracy.

In this paper I'll present a few LSTM based architectures for learning this classification, jumpstarted with pretrained character embeddings (Section 3). Generally, these machines achieved approximately 50% accuracy on a six-way classification task and 75% when the task was reduced to two classes (phonological and non-phonological mistakes). In section four I'll present each classifier's results and briefly examine the changes to the character embedding space which are learned specifically for aphasia error classification. First, however, I'll walk through the data sources and representation (Section 2).

2. DATA SOURCES AND FORMATS

ARPABET AND CMUDICT

For this project, phonological representations are recorded in ARPAbet, which is an ASCII based phonological representation of American English. Carnegie Mellon University maintains a dictionary for conversion between standard orthography and ARPAbet: cmudict. The character embeddings for this project are trained on an unknown version of cmudict which has been saved on the department servers and does not match any of the versions available from the current cmudict maintainer's github.

Error Category	N (% of whole)
Phonological Errors	**4063 (58)**
PR. Neologism	1982 (28.3)
Formal	1447 (20.1)
Mixed	634 (9.1)
Non-phonological Errors	**2938 (42)**
Other	1190 (17.0)
A. Neologism	654 (9.3)
Semantic	1094 (15.6)

MAPPD AND PARALG

The department has a github project called paralg., a part of which includes a subset of the Mappd dataset. For this project, I'm using the target and production columns of the Mappd subset as well as the error codes in this set. As shown in the table above, label categories are somewhat, but not dramatically unbalanced. Classifier accuracy is affected by this imbalance but not excessively so.

3. ARCHITECTURES

The structure of this project required pretraining ARPAbet character embeddings and passing embedded target and production sequences to an LSTM classifier. Two sets of character embeddings were trained, one with a unidirectional LSTM, and one with a bidirectional LSTM. Classifier performance was compared across three separate architectures.

CHARACTER EMBEDDINGS

Embeddings were trained as input to a predict-the-next-character LSTM. Training sequences included 133 thousand words from came from Carnegie Mellon's cmudict, each padded with beginning and end of sequence characters. Layer dimensions were as follows:

Character Embedding (figure 1):

- Embedding: 86 char -> 32 embedding
- LSTM: 32 embedding -> 20 hidden
- Linear: 20 hidden -> 86 char

A bidirectional embedding was also trained, though it initially seemed ineffective, bug fixes came through after initial experiments were run. Architecture performance may have differed depending on which architecture was

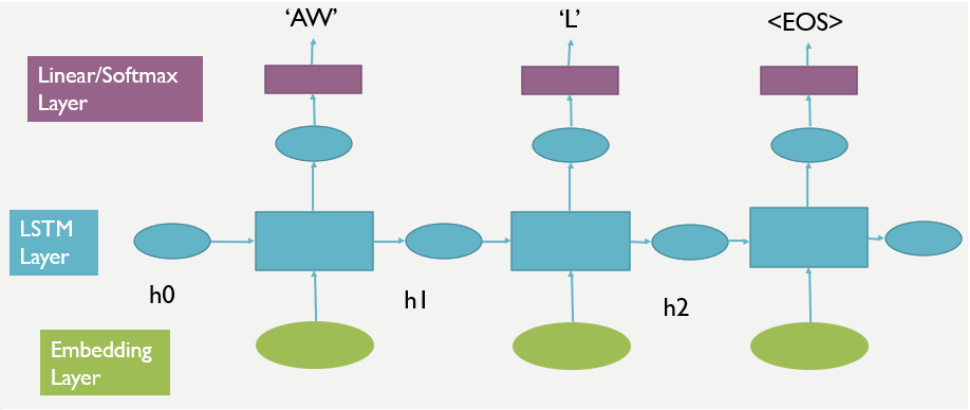


FIGURE 1: CHARACTER EMBEDDING ARCHITECTURE

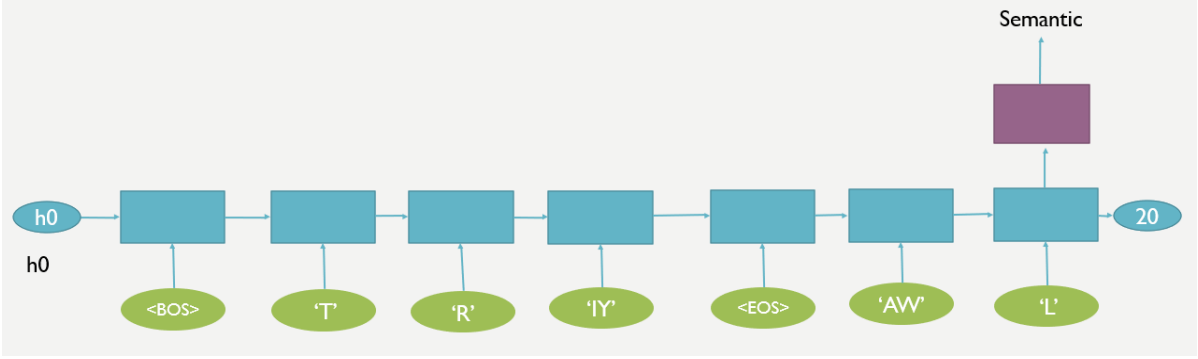


FIGURE 2: SEQUENTIAL LSTM ARCHITECTURE

used to train the embeddings. As such, results for each architecture are reported with both forward and bi-directionally trained embeddings.

LSTM CLASSIFER

Two separate but similar LSTM architectures were trained on the main, six-way classification task. Both passed embedded representations of both target and produced utterances to an LSTM layer which led to a linear softmax layer. In the first, 'sequential', architecture, inputs were concatenated before being processed by the LSTM layer and represented as one 20-dimensional output to the linear layer. The second 'parallel' architecture, processes target and production by feeding the production and target to the LSTM separately and concatenating the two into a 40-dimensional vector linear input. Because this shifts much of the relation learning outside of the LSTM block, an additional linear layer was added to the parallel architecture. Layer dimensions are as follows:

Sequential Architecture (figure 2):

- Embedding: 86 char -> 32 embedding
- LSTM: 32 embedding -> 20 hidden
- Linear: 20 hidden -> 6 classes

Parallel Architecture (figure 3):

- Embedding: 86 char -> 32 embedding
- LSTM: 32 embedding -> 20 hidden (production)
- LSTM: 32 embedding -> 20 hidden (target)
- Linear: 40 hidden (p+t) -> 20
- Linear: 20 -> 6 classes

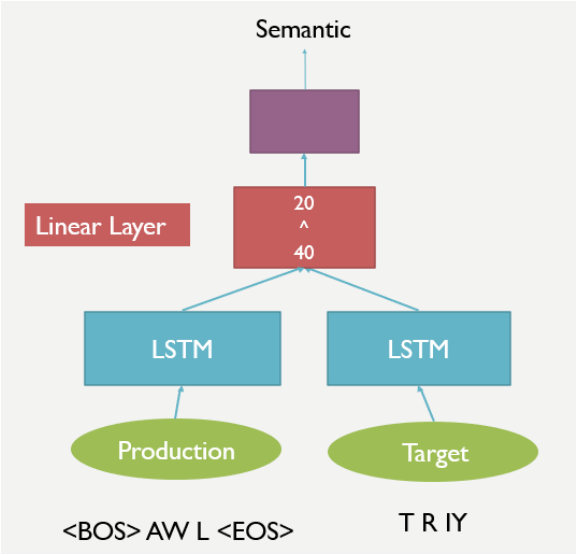


FIGURE 3: PARALLEL LSTM ARCHITECTURE

One additional LSTM architecture was trained—a binary classifier, identical to the parallel structure shown in figure 3, but given binary labels +/- phonological, classified as shown in table 1.

4. RESULTS

CLASSIFIER ACCURACY

Validation and test accuracies were surprisingly consistent across models. The binary classifier achieved around 66% accuracy, and the 6-way classifier about 45% accuracy. The one number that stands out among test and train accuracies is the serial model's 54% validation. In a prior run of these models, a similar thing happened, but at that point the parallel model was out performing on the validation set. In all, this seems to suggest that models have similar general accuracy and more trial numbers would

be required to definitely assert that one architecture is better than the other.

On the other hand, the training accuracy of the serial model seems to be slightly higher than the training accuracy of the parallel model. Combined with their similar test performance, this suggests that the serial model may have begun to overfit (after 10 epochs), despite having the lower number of total parameters.

While bidirectional embedding training initially seemed to have an impact on models, over all the numbers do not suggest that this was important. This as well as a few other factors which I'll return soon, contributes to the impression that character embeddings may not actually be helpful for the classification task as it is currently defined.

Precision, recall, and F1 scores for two of the machines are included below for good measure.

	precision	recall	f1-score	support
A.Neologism	0.26	0.21	0.23	92
Formal	0.42	0.48	0.45	237
Mixed	0.43	0.30	0.35	104
Other	0.38	0.27	0.32	205
P.R.Neologism	0.55	0.65	0.59	331
Semantic	0.49	0.53	0.51	187
avg / total	0.45	0.46	0.45	1156

(Forward emb) Serial LSTM Accuracy 0.4602076124567474

	precision	recall	f1-score	support
A.Neologism	0.26	0.21	0.23	92
Formal	0.40	0.35	0.37	237
Mixed	0.37	0.25	0.30	104
Other	0.32	0.44	0.37	205
P.R.Neologism	0.58	0.61	0.60	331
Semantic	0.47	0.44	0.46	187
avg / total	0.44	0.44	0.43	1156

(Forward emb) Parallel LSTM Accuracy: 0.435121107266436

CONFUSION

Parallel LSTMs are more likely to classify 'Formal' as 'Other' and serial are more likely to classify 'Other' as 'Formal'. Without a better understanding of these two categories I'm at something of a loss to explain this difference.

All machines confused PR.Neo and A.Neo, especially the parallel LSTM, which almost never predicted A.Neo. This confusion is especially disappointing, as the defining difference between these categories is phonological, which one might have hoped the pretrained word embeddings would help the machine understand.

Beyond this, all machines also had some trouble with the mixed category, frequently predicting PR Neo. Given that Mixed is the least common label and PR Neo the most, and that both of them are phonological mistakes, this confusion, at least, makes sense. While misclassifications of true P.R. Neologisms were relatively rare (recall was > .6), the serial machines had a tendency to incorrectly label the neologism 'Other' (shown by the '50' in both serial LSTM confusion matrixes).

EMBEDDING SPACES

Embedding visualizations were not as helpful as I had hoped they might be. The pre-classification embeddings seem to have separated consonants from vowels. There is no obvious structure to the consonant cluster. Vowels seem more likely to cluster based on stress, aspiration or rounding than on placement in vowel space. The separation between consonants and vowels disappears after classification training without developing any other obvious patterns.

5. CONCLUSIONS AND FUTURE DIRECTIONS

Not using character embeddings is probably actually better for this task, given that the human guidance for this decision is based on character matching. A character model may learn that 'S' and 'Z' are similar, but if a human

judge treats them as distinct characters, such knowledge could only distract the model.

This work is deserving of substantial error analysis. What is it that indicates to a machine that something is a neologism, but convinces it that two utterances are phonologically related when a human has decided they are not? The answers may be in the errors.

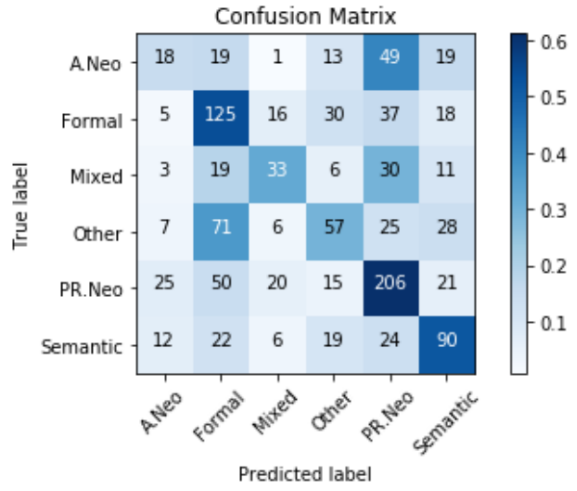
I did not end up measuring the cosine distance between embeddings, but this is a much more thorough avenue through which one might understand differences in character representations than the t-sne visualizations given here. I also haven't drawn training curves for these models and perhaps they could have continued to improve given more than 10 epochs of training.

I do think that t-sne visualizations of LSTM outputs might be interesting to look at. Perhaps one can find clusters of hidden representations and relate them to the classifications of a given machine. Graphing this and investigating the similarities among clusters and the distinguishing features of errors could help with feature engineering. This was one of my favorite end of project to-dos that will just have to wait for next time.

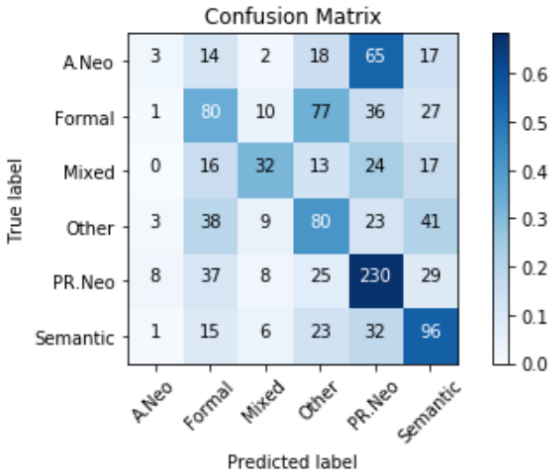
6. APPENDIX OF RESULTS FIGURES AND TABLES

Classifier	Train Acc		Val Acc		Test Acc	
	Binary Emb	Forward Emb	Binary Emb	Forward Emb	Binary Emb	Forward Emb
Binary	76%	77%	66%	67%	66%	67%
Parallel	56%	59%	44%	43%	45%	44%
Serial	61%	61%	46%	**54%**	46%	46%

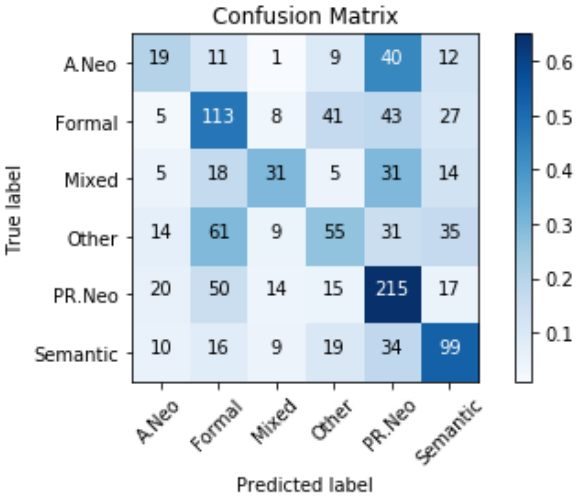
```
Serial LSTM Accuracy 0.45761245674740486
```



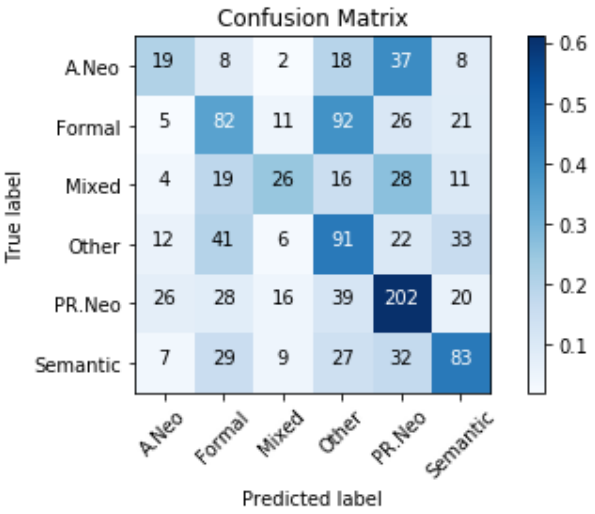

```
Parallel LSTM Accuracy: 0.45069204152249137
```




```
(Forward emb) Serial LSTM Accuracy 0.4602076124567474
```

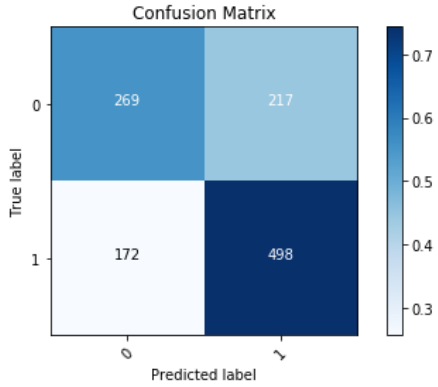



```
(Forward emb) Parallel LSTM Accuracy: 0.435121107266436
```



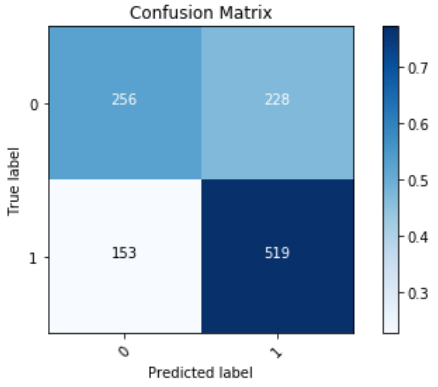
	precision	recall	f1-score	support
0	0.61	0.55	0.58	486
1	0.70	0.74	0.72	670
avg / total	0.66	0.66	0.66	1156

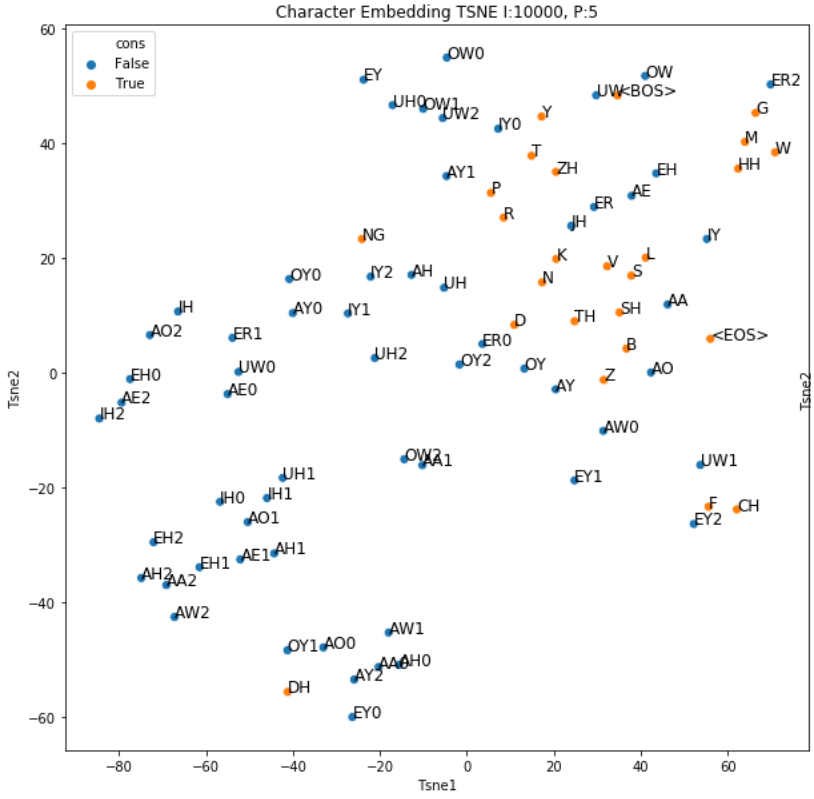
Binary Accuracy: 0.6634948096885813

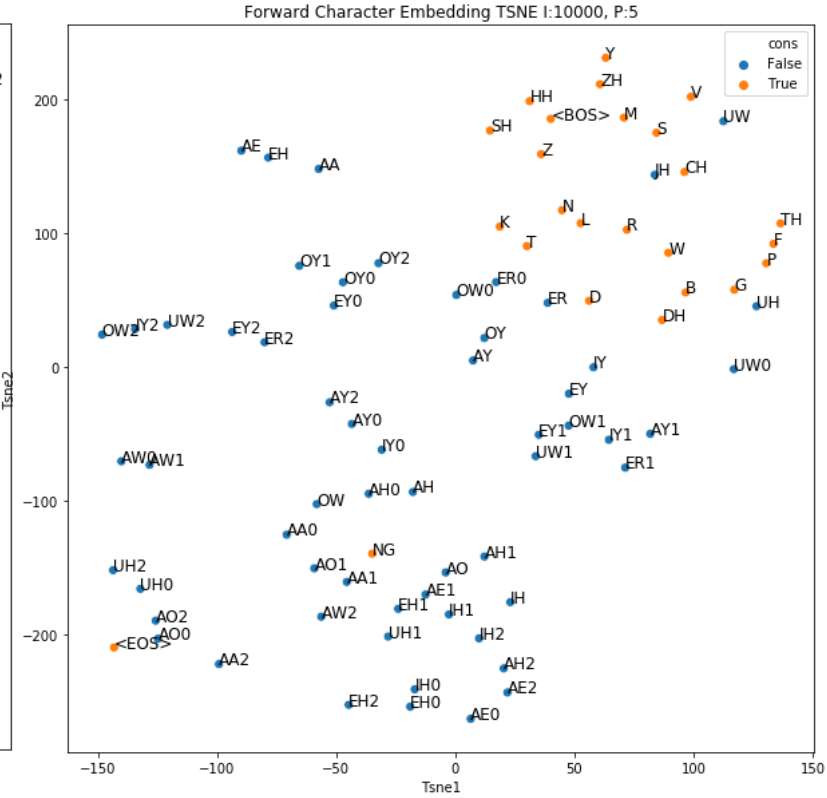


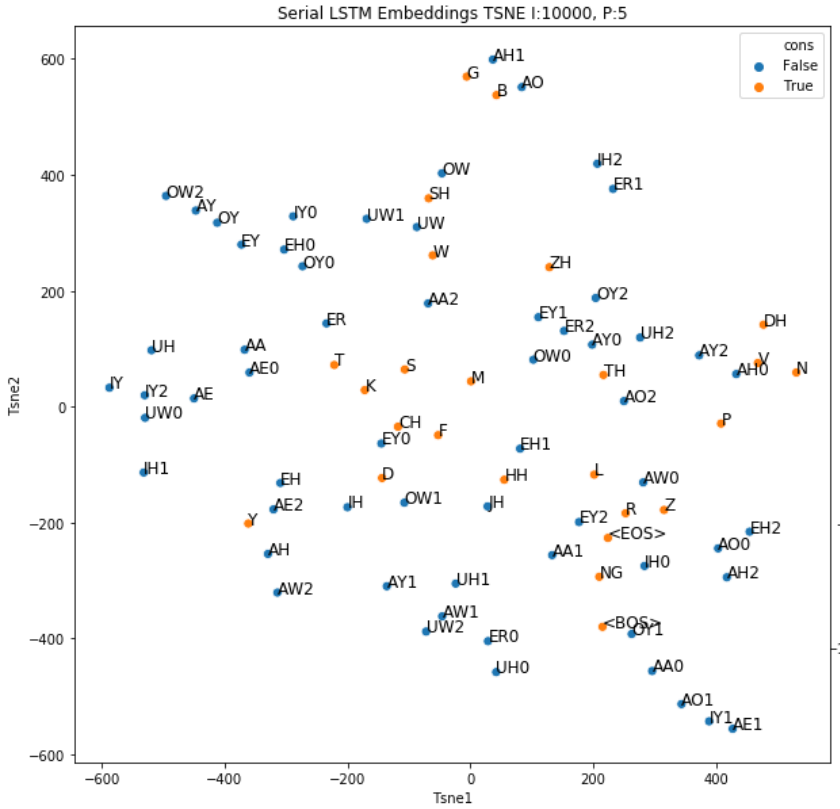
	precision	recall	f1-score	support
0	0.63	0.53	0.57	484
1	0.69	0.77	0.73	672
avg / total	0.67	0.67	0.67	1156

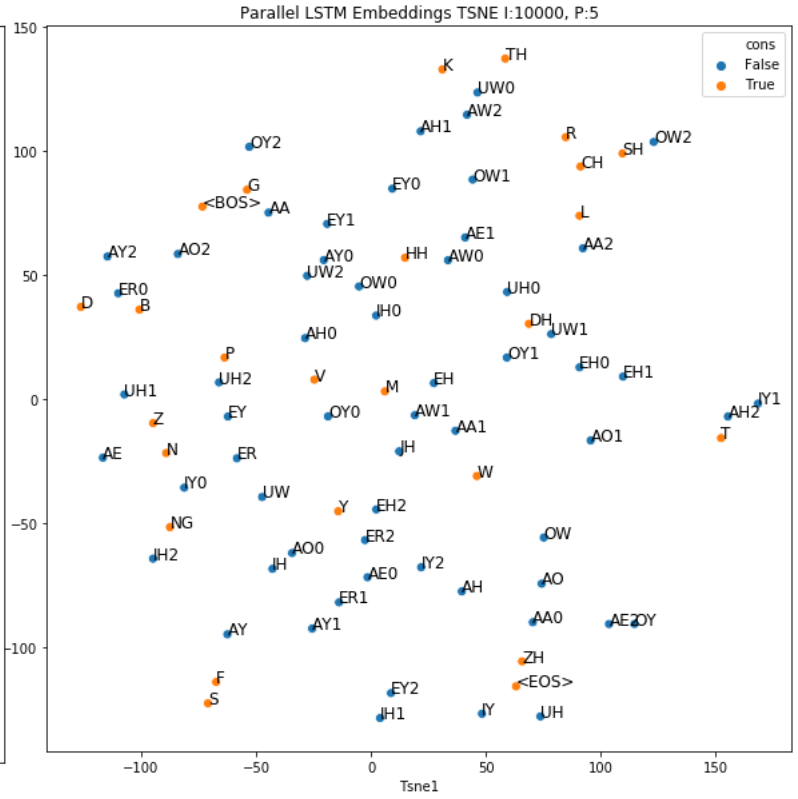
(Forward emb) Binary Accuracy: 0.6704152249134948








Serial LSTM Embeddings TSNE I:10000, P:5
cons
False
True
Tsne1
Tsne2


Parallel LSTM Embeddings TSNE I:10000, P:5
cons
False
True
Tsne1


Binary Classifer Embeddigns TSNE I:10000, P:5
cons
False
True
Tsne1
Tsne2