Your final report should be a .pdf or .ipynb document that summarizes all work you did on this project from your initial proposal to your last experiments. You should push it to your GitHub repo with a filename of `report.pdf` or `report.ipynb`.

Your report should be self-contained. That means if you want to refer to a goal you wrote in your proposal or update, you should copy it here. If you want to reference a plot that shows your model's accuracy over time, include that plot in your report. If it's not in your report, it doesn't contribute to your grade. The important exception to this is that **your written report should not contain your code**. See "code and documentation" below for how we will grade your code.

Goals and discussion (16 points)

The broad purpose of this project was for you to learn how to apply deep learning methods by working towards goals you chose. Every project team had different goals, and those goals varied in terms of their ambitious and specificity. Thus, your grade is not determined by counting up the number of goals you achieved. It's possible to get full points without completing many goals, and completing all your goals does not guarantee you full points. You should approach this section as an exercise in *persuasive writing*; try to convince us that you spend a substantial amount of time pursuing these goals and that you learned some specific and interesting things.

For completed goals that involve dataset collection or preprocessing, please include at least one example of what the output of that preprocessing was. For completed goals that involve experiments, please include at least one plot or table showing an experimental comparison.

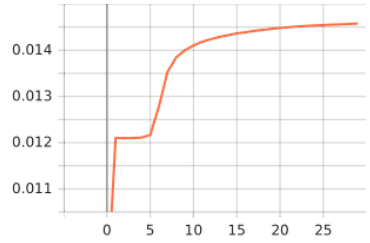
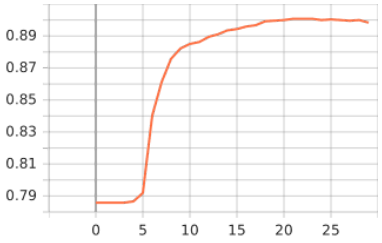
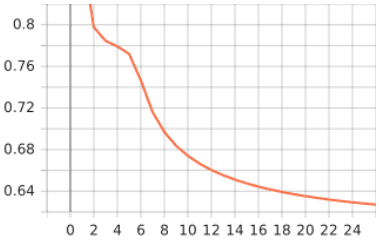
This section will be graded holistically; you can still receive full points even if you abandoned all your stretch goals, for example if you went above and beyond on your desired goals.

Essential goals

List all essential goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1.) Train a bag of words model on the twitter data

We completed this by building a simple model and tested it across a variety of pre-processing methods detailed in the third essential goal Interestingly, our best results came from a very small network - 1 hidden layer, possibly reflecting a flaw in the dataset (detailed in next steps). This achieved a peak testing accuracy of 90%.

Training Accuracy	Test Accuracy	Training BCE Loss
		

Confusion matrix:

Prediction / True	Hate Speech	Offensive Language	Neither
Hate Speech	0	0.87	0.12
Offensive Language	0	0.98	0.024
Neither	0	0.21	0.79

2.) Classify data into a) hate speech or offensive language b) neither

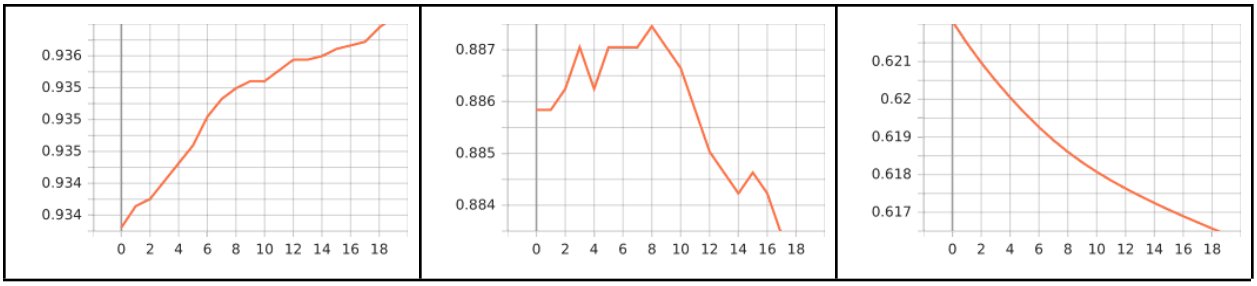
This was completed through the prior bag of words model. The bag of words model was not able to separate between hate speech and offensive language,as seen in the confusion matrix above. We found it surprising that even when the loss class weighting was 10:1, the model didn't learn to separate these two. We use more advanced models for future goals that are effective at this.

3.) Perform a comparison of pre-processing methods

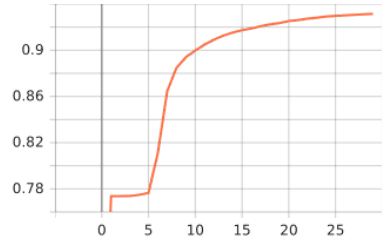
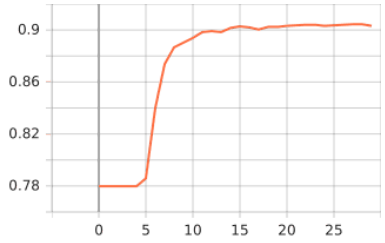
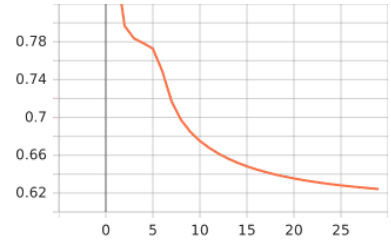
We also used the bag of words model for this part. The final model used tokenization, Porter stemming, and part-of-speech tagging. We also tried bi-grams with poor results, but this was too computationally expensive to fully explore, even when filtering out mentions, hashtags, and emojis. Interestingly, part-of-speech tagging had adverse effects on our results, we think this could use more data to account for words being mis-classified in linguistically questionable tweets.

No preprocessing - 88.7% accurate

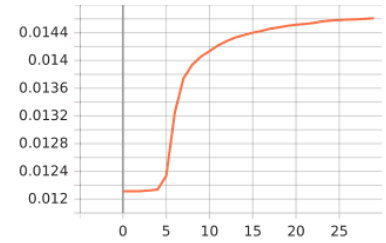
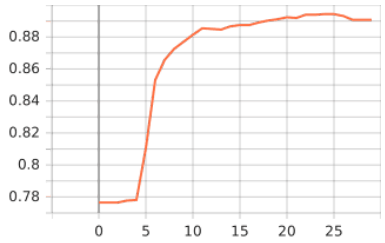
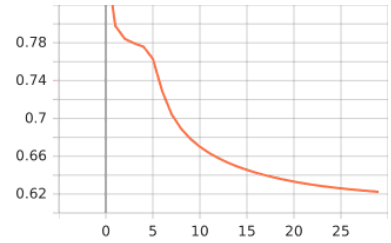
Training Accuracy	Test Accuracy	Training BCE Loss



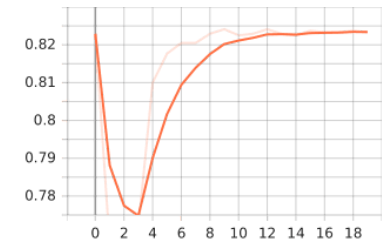
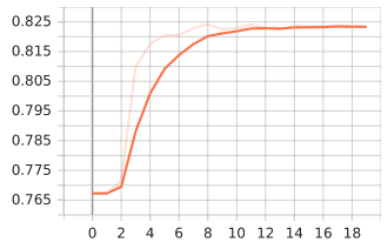
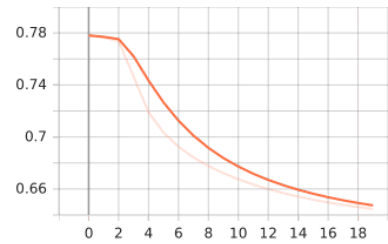
Porter Stemming - 90.2% accurate

Training Accuracy	Test Accuracy	Training BCE Loss
		

Porter Stemming + Part-of-speech Tagging - 88.8% accurate

Training Accuracy	Test Accuracy	Training BCE Loss
		

Tagging + Bigram (filtered) - 82.3% accurate

Training Accuracy	Test Accuracy	Training BCE Loss
		

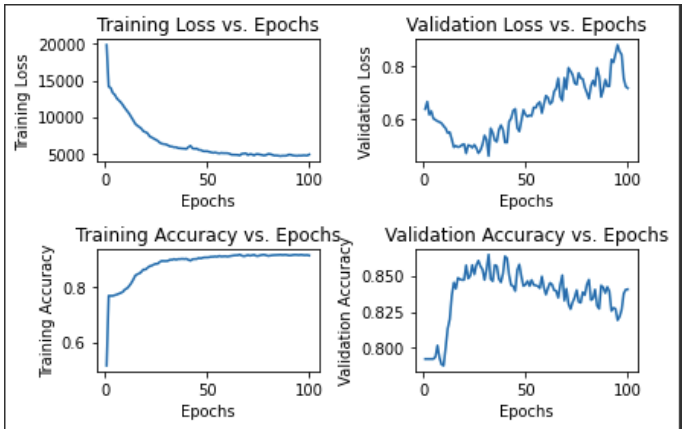
Desired goals

List all desired goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1.) We will recognize hate speech tweets based on the meaning of the sentence, not just the presence of a word using LSTM/GRU RNN models. This will allow the model to recognize false-negative hate speech that doesn't use specific words, as well as false-positive tweets that use those words but aren't hate speech, such as commentary or words with multiple definitions and uses. We will evaluate this using a Bags of Words Model.

Because we are working with a classification problem, we used one LSTM encoder with a cross entropy loss, which was initialized using Pytorch's implementation. In order to feed data into our LSTM model, we first needed to apply tokenization (split all words into an array), remove unnecessary punctuation using regular expressions, build a vocabulary using vocab2index, and encode the words into a numerical vector. Finally, we had to define the training loop to collect metric data and calculate loss, accuracy, their gradients, and taking gradient steps. GRU was done in a similar manner. The most interesting part about this goal is that we learned more about our data: after applying a regular expression to remove punctuation, we realized that all the tweet lengths are no longer than 40! The most difficult part of this goal is to get the model to train because we were having trouble setting up the training loop.

The following graphs show the validation loss/accuracy and the training loss/ training accuracy of our first model!



This initial model is worse than the BOW model 86.5% vs. 90%

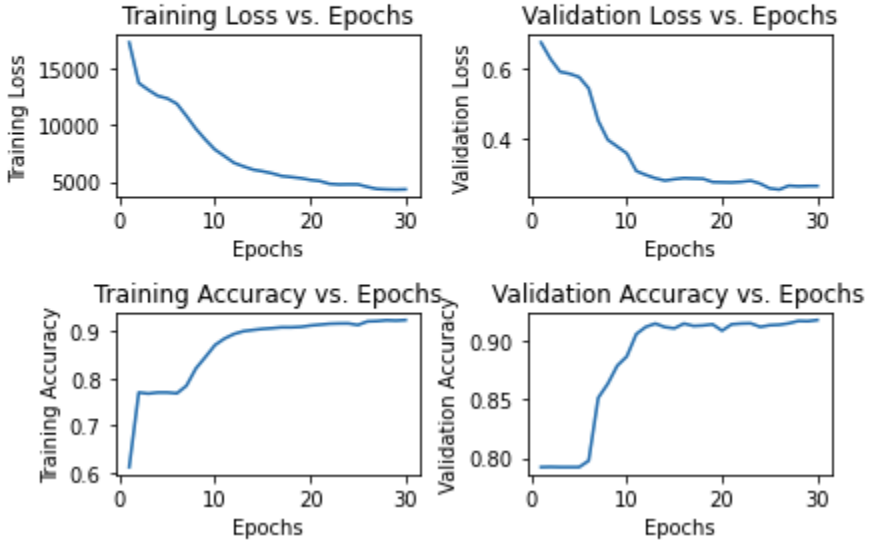
2.) We want to try to achieve high accuracy (perhaps even higher than the source paper's 91% for offensive speech and 61% for hate speech) by using different regularization methods, including weight decays, dropout, batch normalization, and weight constraints and compare the performance.

We did this by doing a lot of trial and error since there are many different hyperparameters (dropout and weight constraints in the model and weight decay is in the adam optimizer). To avoid getting overwhelmed with the combinatorial number of possible experiments we can run, we started with a set of parameters that yielded acceptable results and began our grid search there. One “algorithm” we used to search parameters was by pruning which is to stop searching in the path if the most recent change shows a decrease in accuracy or increase in loss. The interesting thing about this goal is that we found weight decay values need to be small, otherwise the model will have trouble learning. The most difficult part is finding good parameters such that it improves from our first model!

We abandoned the batch normalization regularization technique because we found that it is impossible (ill-advised) to apply batch normalization when training an RNN.

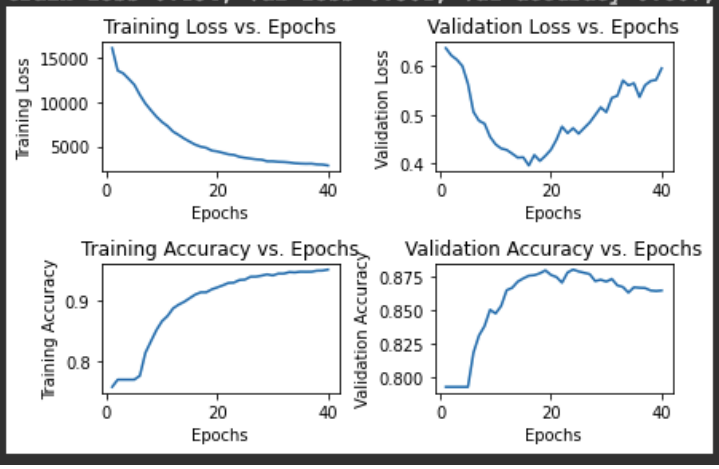
The following figure shows the best accuracy (91.5%) we achieved using the LSTM, which was done using both weight decay and dropout. This beats the best BOW by 1.3%. The hyperparameters we used were: Embedding = 8, hidden dimension = 8. Dropout = 0.5, epochs = 30, lr = .1, weight decay = .0001, max_norm = 20

```
train loss 0.693, val loss 0.627, val accuracy 0.793, and val rmse 0.455
train loss 0.546, val loss 0.452, val accuracy 0.852, and val rmse 0.435
train loss 0.337, val loss 0.297, val accuracy 0.912, and val rmse 0.350
train loss 0.278, val loss 0.287, val accuracy 0.913, and val rmse 0.336
train loss 0.243, val loss 0.278, val accuracy 0.915, and val rmse 0.331
train loss 0.220, val loss 0.267, val accuracy 0.915, and val rmse 0.332
```



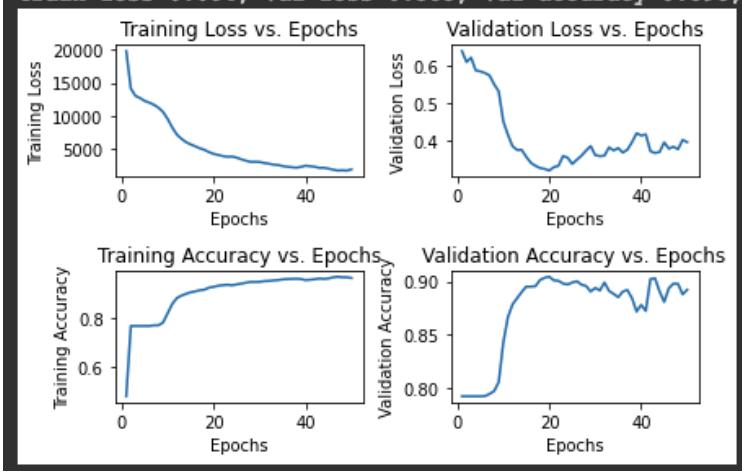
Other Experiments include: Using just low dropout doesn't get past the 90% threshold (87.7% peak).

```
train loss 0.685, val loss 0.622, val accuracy 0.792, and val rmse 0.456
train loss 0.497, val loss 0.488, val accuracy 0.831, and val rmse 0.452
train loss 0.337, val loss 0.428, val accuracy 0.865, and val rmse 0.404
train loss 0.251, val loss 0.418, val accuracy 0.877, and val rmse 0.382
train loss 0.208, val loss 0.476, val accuracy 0.871, and val rmse 0.392
train loss 0.179, val loss 0.484, val accuracy 0.877, and val rmse 0.394
train loss 0.164, val loss 0.539, val accuracy 0.869, and val rmse 0.413
train loss 0.154, val loss 0.561, val accuracy 0.867, and val rmse 0.423
```



Using just weight decay shows a more significant increase in accuracy (90.1%), but still not the best.

```
train loss 0.712, val loss 0.612, val accuracy 0.792, and val rmse 0.456
train loss 0.588, val loss 0.575, val accuracy 0.795, and val rmse 0.457
train loss 0.360, val loss 0.385, val accuracy 0.879, and val rmse 0.408
train loss 0.255, val loss 0.332, val accuracy 0.896, and val rmse 0.381
train loss 0.195, val loss 0.332, val accuracy 0.901, and val rmse 0.365
train loss 0.160, val loss 0.359, val accuracy 0.897, and val rmse 0.377
train loss 0.137, val loss 0.359, val accuracy 0.899, and val rmse 0.367
train loss 0.110, val loss 0.375, val accuracy 0.892, and val rmse 0.376
train loss 0.113, val loss 0.371, val accuracy 0.902, and val rmse 0.357
train loss 0.084, val loss 0.383, val accuracy 0.898, and val rmse 0.366
```



3.) We would like to construct a GRU model and compare it with an LSTM. Just like with the LSTM, we will also train it across different hyperparameters. We hope to see better performance on the GRU because our dataset is relatively small (GRU tends to work better for smaller dataset)

This goal was listed in our update.ipynb, but it was just another part of the 1st goal.

We did this by first running the GRU model with the same parameters as the LSTM to compare their performance. We found that overall performance was very similar across both models. This could be that the optimal set of hyperparameters are just similar for this problem or that the complexity of the data is high enough for the LSTM to be significantly better. For example, using embedding = 5, hidden dimension = 5, dropout = .5, epochs = 50, learning rate = .1, weight decay = .0001, and max norm = 10 yielded these results.

Below are the results for the GRU:

```
train loss 0.735, val loss 0.650, val accuracy 0.787, and val rmse 0.461
train loss 0.571, val loss 0.500, val accuracy 0.822, and val rmse 0.444
train loss 0.386, val loss 0.359, val accuracy 0.883, and val rmse 0.424
train loss 0.300, val loss 0.339, val accuracy 0.895, and val rmse 0.374
train loss 0.279, val loss 0.326, val accuracy 0.894, and val rmse 0.368
train loss 0.261, val loss 0.340, val accuracy 0.893, and val rmse 0.375
train loss 0.252, val loss 0.352, val accuracy 0.891, and val rmse 0.380
train loss 0.248, val loss 0.326, val accuracy 0.891, and val rmse 0.368
train loss 0.235, val loss 0.336, val accuracy 0.892, and val rmse 0.373
train loss 0.224, val loss 0.313, val accuracy 0.895, and val rmse 0.362
train loss 0.218, val loss 0.318, val accuracy 0.897, and val rmse 0.362
train loss 0.208, val loss 0.327, val accuracy 0.893, and val rmse 0.372
train loss 0.209, val loss 0.327, val accuracy 0.893, and val rmse 0.375
train loss 0.206, val loss 0.325, val accuracy 0.892, and val rmse 0.373
train loss 0.203, val loss 0.326, val accuracy 0.891, and val rmse 0.373
train loss 0.201, val loss 0.336, val accuracy 0.891, and val rmse 0.374
train loss 0.201, val loss 0.334, val accuracy 0.893, and val rmse 0.367
train loss 0.201, val loss 0.344, val accuracy 0.891, and val rmse 0.371
train loss 0.197, val loss 0.322, val accuracy 0.897, and val rmse 0.369
train loss 0.198, val loss 0.343, val accuracy 0.893, and val rmse 0.369
```

Below are the results for the LSTM:

```
train loss 0.675, val loss 0.619, val accuracy 0.792, and val rmse 0.456
train loss 0.424, val loss 0.347, val accuracy 0.881, and val rmse 0.394
train loss 0.315, val loss 0.327, val accuracy 0.889, and val rmse 0.357
train loss 0.286, val loss 0.285, val accuracy 0.902, and val rmse 0.345
train loss 0.261, val loss 0.286, val accuracy 0.900, and val rmse 0.348
train loss 0.257, val loss 0.288, val accuracy 0.892, and val rmse 0.349
train loss 0.250, val loss 0.291, val accuracy 0.894, and val rmse 0.353
train loss 0.239, val loss 0.297, val accuracy 0.892, and val rmse 0.356
train loss 0.239, val loss 0.293, val accuracy 0.893, and val rmse 0.359
train loss 0.241, val loss 0.288, val accuracy 0.899, and val rmse 0.351
```

Training and validation loss in the above results differ slightly while accuracy is no different in either model. However, a few experiments were interesting in that the LSTM performed relatively poorly compared to its other experiments while performance in the GRU was still solid. Using the hyperparameters from above and changing max norm to .005 yielded the following.

Below are results for the GRU:

```
train loss 0.666, val loss 0.630, val accuracy 0.787, and val rmse 0.461
train loss 0.436, val loss 0.377, val accuracy 0.864, and val rmse 0.429
train loss 0.332, val loss 0.322, val accuracy 0.891, and val rmse 0.387
train loss 0.286, val loss 0.294, val accuracy 0.903, and val rmse 0.347
train loss 0.263, val loss 0.286, val accuracy 0.900, and val rmse 0.342
train loss 0.252, val loss 0.288, val accuracy 0.901, and val rmse 0.343
train loss 0.238, val loss 0.296, val accuracy 0.902, and val rmse 0.342
train loss 0.241, val loss 0.300, val accuracy 0.903, and val rmse 0.339
train loss 0.230, val loss 0.300, val accuracy 0.902, and val rmse 0.343
train loss 0.233, val loss 0.318, val accuracy 0.901, and val rmse 0.339
```

Below are results for the LSTM:

```
train loss 0.725, val loss 0.629, val accuracy 0.792, and val rmse 0.456
train loss 0.672, val loss 0.628, val accuracy 0.792, and val rmse 0.456
train loss 0.672, val loss 0.625, val accuracy 0.792, and val rmse 0.456
train loss 0.672, val loss 0.624, val accuracy 0.792, and val rmse 0.456
train loss 0.672, val loss 0.624, val accuracy 0.792, and val rmse 0.456
train loss 0.671, val loss 0.624, val accuracy 0.792, and val rmse 0.456
train loss 0.671, val loss 0.624, val accuracy 0.792, and val rmse 0.456
train loss 0.671, val loss 0.624, val accuracy 0.792, and val rmse 0.456
train loss 0.670, val loss 0.622, val accuracy 0.792, and val rmse 0.456
train loss 0.662, val loss 0.617, val accuracy 0.792, and val rmse 0.456
```

While the GRU seems to get better validation performance with this change, the LSTM performs considerably worse all around. This could be due to the low max norm being more effective at preventing overfitting in the GRU due to its lower capacity.

We then experimented with different sets of hyperparameters to further improve GRU performance. Increasing embedding, increasing max norm, and increasing weight decay seemed to be most effective in yielding better accuracy during these experiments while other changes seemed to change training loss. The highest accuracy achieved was .905 using embedding = 7, hidden dimension = 5, dropout = .5, epochs = 100, learning rate = .1, weight decay = .001, and max norm = 75 yielded these results:

```
train loss 0.711, val loss 0.647, val accuracy 0.787, and val rmse 0.461
train loss 0.478, val loss 0.394, val accuracy 0.863, and val rmse 0.437
train loss 0.355, val loss 0.323, val accuracy 0.902, and val rmse 0.355
train loss 0.322, val loss 0.319, val accuracy 0.903, and val rmse 0.345
train loss 0.318, val loss 0.321, val accuracy 0.901, and val rmse 0.351
train loss 0.314, val loss 0.335, val accuracy 0.903, and val rmse 0.353
train loss 0.311, val loss 0.307, val accuracy 0.902, and val rmse 0.353
train loss 0.306, val loss 0.301, val accuracy 0.904, and val rmse 0.336
train loss 0.308, val loss 0.306, val accuracy 0.903, and val rmse 0.336
train loss 0.303, val loss 0.303, val accuracy 0.900, and val rmse 0.343
train loss 0.306, val loss 0.303, val accuracy 0.903, and val rmse 0.339
train loss 0.303, val loss 0.305, val accuracy 0.902, and val rmse 0.330
train loss 0.304, val loss 0.304, val accuracy 0.901, and val rmse 0.338
train loss 0.301, val loss 0.296, val accuracy 0.901, and val rmse 0.338
train loss 0.299, val loss 0.296, val accuracy 0.902, and val rmse 0.341
train loss 0.300, val loss 0.289, val accuracy 0.901, and val rmse 0.331
train loss 0.304, val loss 0.297, val accuracy 0.898, and val rmse 0.341
train loss 0.301, val loss 0.286, val accuracy 0.902, and val rmse 0.335
train loss 0.295, val loss 0.300, val accuracy 0.904, and val rmse 0.336
train loss 0.293, val loss 0.283, val accuracy 0.905, and val rmse 0.332
```

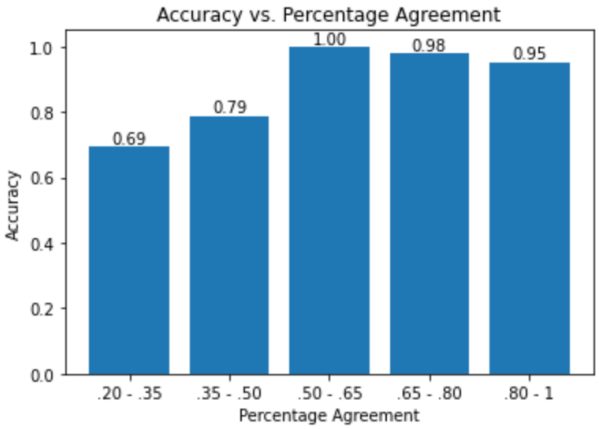
4.) Newly added goal, based on update feedback: seeing how conflicting crowd-sourced labels correlate with model accuracies for LSTM, BOW, and Transformer

To accomplish this goal, we first looked for metrics that can measure “conflicting crowd-sourced labels.” We came across with descriptive statistics that could’ve potentially helped us, which were the Cohen Kappa value or the generalized form Fleiss Kappa. Unfortunately, these metrics calculate the disagreement across the entire dataset, which is not what we want(we want a disagreement for 1 particular data point). As a result, we found and used the following formula:

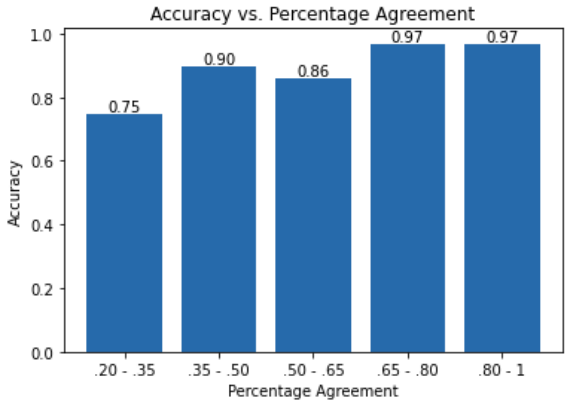
$$\sum_{k=1}^{q} \frac{r_k(r_k - 1)}{r(r-1)}$$

Where q is the total number of categories, r is the total number of raters that assigned the item to any category, and r_k is the number of raters that assigned the item to category k. We can easily see that if all raters agree, this will yield a value of 1 and if all disagree, then this will yield 0. Then, we looped through the validation dataset, and counting the number of predictions from each of our models:

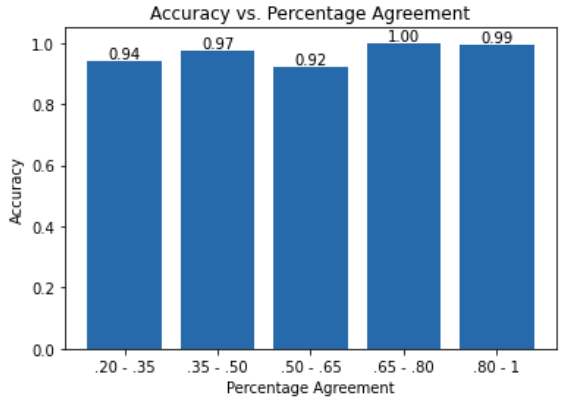
Bag of Words:



LSTM:



Transformer:



Each of these graphs show the accuracy of the best model we achieved against data points that have a certain percentage agreement. We grouped them in bins of 15%.

One interesting thing is that, as shown in the graphs, the transformer outperformed both the BOW and LSTM for data points that do have clear agreements, which was unexpected, while both the BOW and LSTM had an increasing trend, which was expected.

The most difficult part in this goal is figuring out the mismatch (bug) between the graphs from this goal and the validation/training loss and accuracy graph from the training loop.

Stretch goals

List all stretch goals that you proposed either in your second proposal or project update. For each, (a) include one or two sentences on how you completed it, what progress you made, or why you abandoned it, and (b) include one to two sentences on what was the most interesting or difficult part.

1.) Transfer to Different Dataset

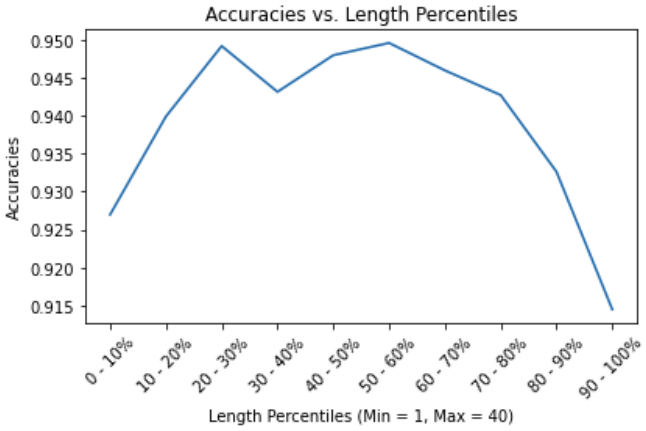
We completed this goal, but not with good results - 54% accurate without change (considering offensive language to be non-hate speech) and 84% when changing the final layer. We used the bag-of-words model as we wanted to try just re-training the final layer which didn't take well to LSTM, and our transformer model was a stretch goal. This dataset just didn't perform well with this architecture at all, as when we re-trained the full model on it it achieved just 84%, always outputting "not hate-speech".

2.) We want to compare several models and their performances: We think that a different sequential model may perform better than an LSTM given the relatively lengthy tweets; we would like to compare the performance of LSTMs and transformer models on hate speech samples of different lengths.

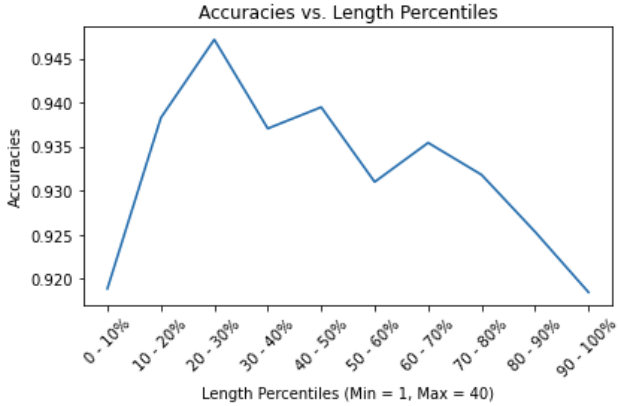
To accomplish this goal, we first defined the Transformer architecture using the positional encoding module (for self-attention) provided in a Pytorch tutorial and a Transformer encoder. Then, I had to sort the data by the lengths of the tweet and separated them into groups of 0-10 percentiles, 10 - 20 percentiles, and so on and calculated the accuracies of the best models I trained. Here, we had to make a decision: whether or not we should sort by the unprocessed tweet or the tokenized tweet because of the amount of punctuation in certain tweets. we Did the latter because we are interested in the number of words, not punctuations. The most difficult part in this goal is getting torchtext to work, which is how we loaded our data because some methods were deprecated!

One interesting thing we found was that the Transformer had no significant advantage against the LSTM in terms of accuracy in our data set (see graphs below). However, we can argue that the transformer's accuracy decreased at a later length (not significant). This may have to do with how I processed the data, the length of the data (longest tweet was 40 words after processing), and the amount of data we have.

Transformer:



LSTM:



These graphs show the accuracy of each model against the length of the tweets.

Everything else

Discuss the additional work you did as part of this project that was not covered by the goals you set for yourself. Try to organize this into "goals" that you could have set for yourself if you had known what needed to be done. For each of these goals, write one or two sentences on why you focused on it, and one or two sentences on what was the most interesting or difficult part. This can be something practical like "we couldn't load our data/model on Colab" or something conceptual like "we tried to understand the fancy model from this new paper but couldn't figure out how to implement it."

Code and documentation (10 points)

While you should not include code in this report, all your code should be uploaded to your GitHub repository. In this part of your report, you should list all .py or .ipynb files that contain your code and a one or two sentence description of what that file contains. Then, in each of those

files, you should make sure that there is enough documentation that we can read through the repository and understand what each part of the code does. This code does not necessarily have to be in immediately working order (e.g., if you cannot upload your data or model files to GitHub), but we should be able to read through your code and notebooks to understand how you implemented things. You absolutely do not need to include a comment for every line of code, but you should include docstrings for (most of) the functions you wrote and descriptions for the coding cells within any .ipynb files.

List:

bag_of_words.ipynb - pre-processing experiments, bag-of-words model, and transferring it to the Stormfront dataset.

LSTM.ipynb - tokenizes the tweets and uses an LSTM classifies the tweets into hate speech, offensive speech or neither. It also runs grid search experiments to find the best parameters, graph its performance against tweet lengths, and accuracies by rater disagreements.

GRU.ipynb - tokenizes the tweets and uses a GRU classifies the tweets into hate speech, offensive speech or neither. It also runs grid search experiments to find the best parameters

Transformer.ipynb - this tokenizes the tweets using torchtext and uses a Transformer to classify the tweets into hate speech, offensive speech or neither. It also graphs its performance against tweet lengths, and accuracy by rater disagreements.

load_stormfront.ipynb - Processing the stormfront dataset from individual files into a csv.

csv_generator.py - generates 10 csv test files by tweet length percentiles, which is used for our stretch goal #4 to see the correlation between accuracy and length of tweet.

Reflections (6 points)

This section is your chance to reflect on this project. You should write at least a few sentences for each of these questions. However, try to be concise; a longer answer is not necessarily a better answer. If you want to write several paragraphs about something you're excited about, that's great! On the other hand, don't just write several paragraphs listing all the hyperparameter values you tried; if we fall asleep reading your answers, you might lose points.

What was interesting?

What did you learn from this project that wasn't covered by other parts of the class? What concepts from the lectures and readings were most relevant to your project?

This project taught us several approaches to sequential data, see the real world limitation of data, and how effective sequential models are in this context and outside of theory. For instance, the bad of words model we have does not ever predict the hate speech label; this is because we have

limited data, conflicting raters, and perhaps not enough data. This taught us that most real-world data isn't all fun like the MNIST dataset, which was very interesting and annoying at the same time. In addition, we expected the transformer to show a large theoretical advantage over the LSTM on longer sequences, which was not the case from stretch goal #2. Of course, this may be due to several factors ranging from incorrect implementation, not enough data, or the tweets were too short to show a significant difference.

Furthermore, we learned more statistic measures for our tasks in the context of disagreements; Though we didn't use it, we found it interesting and learned about the Cohen Kappa and Fleiss Kappa as it provides a systematic way to measure rater disagreement that could have directly helped us in desired goal #4. Of course, we also learn the disagreement statistic that we did use, which is this very simple formula:

$$\sum_{k=1}^{q} \frac{r_k(r_k - 1)}{r(r-1)}$$

Finally, we learned how to search for hyperparameters in a more systematic way, such as writing everything down in grid search and choosing a well performing set of parameters as the starting point. Having learnt this can save us a lot of time and frustration in the future.

The lectures and readings most relevant to the concepts in this project are the sequential models sequence - including, RNN, LSTM, and the Transformer, which unfortunately was taught near the end of the quarter. This, however, still helped with our theoretical understanding of how the sequence models worked and allowed us to recognize expected results or unexpected results that we find interesting (ie. The transformer performed really well even when raters disagreed a lot).

What was difficult?

What were the hardest or most frustrating parts of this project? If someone were about to start on a similar project from scratch, what would you encourage them to do differently?

It was frustrating investing a lot of time in ideas that didn't end up working (or we didn't know how to make work). For instance the bag-of-words model could have been better if it was first designed as an ensemble model with dedicated model/s detecting hate speech vs offensive language. By the time we noticed it wasn't effectively making this distinction, it was too late.

Furthermore, it was also frustrating that I didn't modularize my code in the beginning. This made my experiment code extremely hard to read as they are just filled with repeated code blocks so it made debugging difficult and wasted me a lot of time. I also didn't want to restart modularizing because I was already so deep into the project, but that just hurt the productivity of testing and experimenting so I modularized it at the end. Based on this, I would encourage them to modularize their code even if it doesn't seem like you need it at the moment and do not fall into a sunk-cost fallacy!

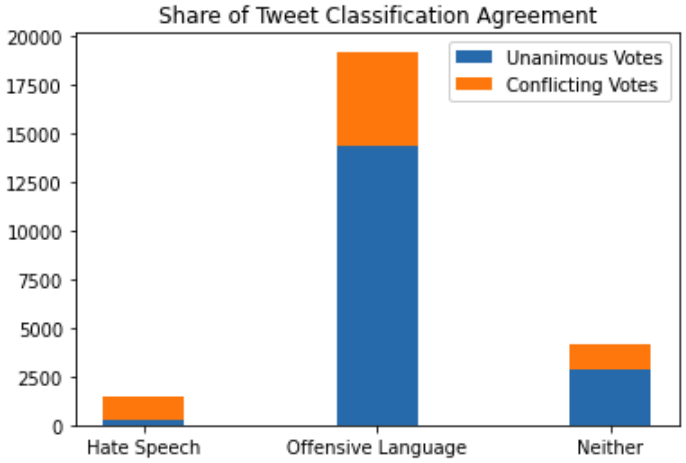
Another difficult part of the project is the search for good parameters for the models. This is because it is very easy to be overwhelmed by the exponentially increasing number of parameters you can change. I actually didn't write down the experiments at the beginning so I probably reran a lot of the same experiments and wasted a lot of time. Therefore, I would encourage people who work on a similar project to do a grid search and random guess parameters at the very beginning until you can find a good set as a starting point. Then, apply our "pruning" algorithm to stop searching in a particular path if metrics didn't show improvements and to take "gradient steps" that find parameters that increase accuracy or decrease loss.

What's left to do?

If you were going to spend another month working full-time on this project, what would you try to accomplish? Why? If I gave your group 1,000,000 USD to use for data collection or compute resources, what would it spend it on? Why?

We would work on building a dataset that is less dependent on certain often-hateful words. From the original paper, the candidate tweets were first chosen by filtering by these words, before being labeled by humans. However, we think there could be better ways to do this, such as going by often-reported tweets. There is a lot of hate-speech that would not be found by this method, with potential to train a better model. This would also better demonstrate the advantages of RNN's and transformers over bag-of-words.

In addition to that, the dataset we have lacks (relative ratio) labels from the hate speech category and that it has the highest percentage of conflicting votes(see graphs below), so the Bow actually never predicts the hate speech label. This may be due to the fact that there is a semantic perception difference or the lack of data. If we have more resources, we would like to go by often-reported tweets or collect more data with more raters so that our models can actually learn the label instead of ignoring it.



Finally, our experiments have given us some directions we would've liked to explore in a project with a greater scope. We think ensemble models would be promising for differentiating between hate speech and offensive language, with specific model/s for each. Working under the limitations of this dataset, we think it would be interesting to explore applying pre-trained models such as BERT, as we were fighting overfitting in all of our experiments.

References:

https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df

https://n8henrie.com/2021/08/writing-a-transformer-classifier-in-pytorch/

https://stats.stackexchange.com/questions/164965/raters-agreement-for-each-item

DAVIDSON, T.; WARMSLEY, D.; MACY, M.; WEBER, I.. Automated Hate Speech Detection and the Problem of Offensive Language. International AAAI Conference on Web and Social Media, North America, may. 2017. Available at: https://aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/view/15665/14843. Date accessed: 31 Jan. 2023.