Eric Zacharia

log-reg-text-class.ipynb Plots

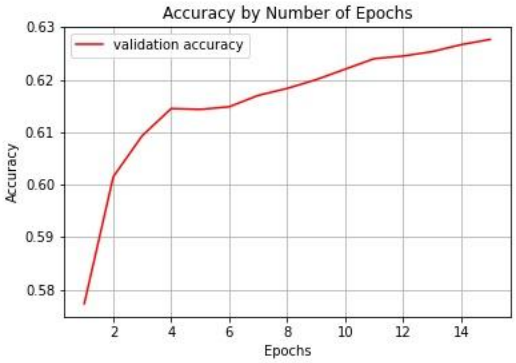

Figure 1 - Adam optimizer 15 Epochs

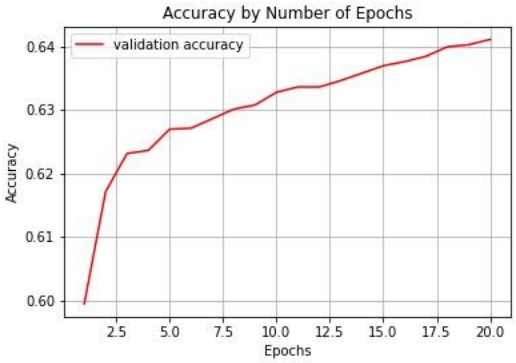

Figure 2 - Adam optimizer 20 Epochs

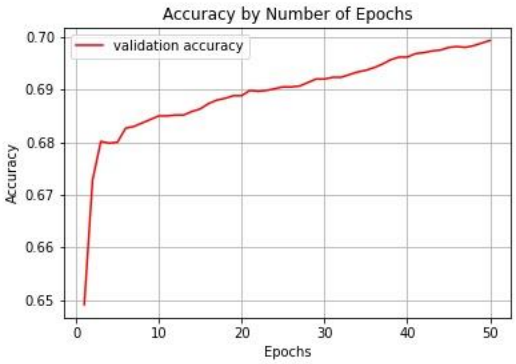

Figure 1 - Adam optimizer 50 Epochs

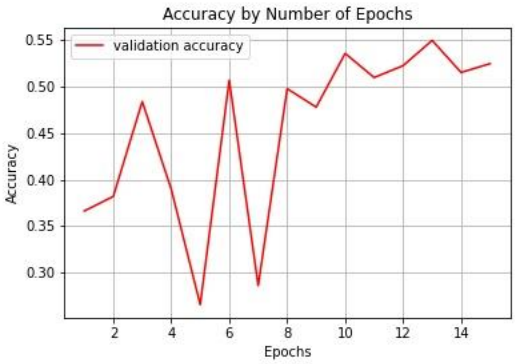

Figure 4 - SGD optimizer 15 Epochs

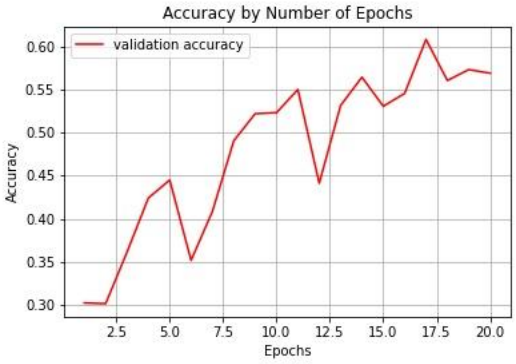

Figure 5 - SGD optimizer 20 Epochs

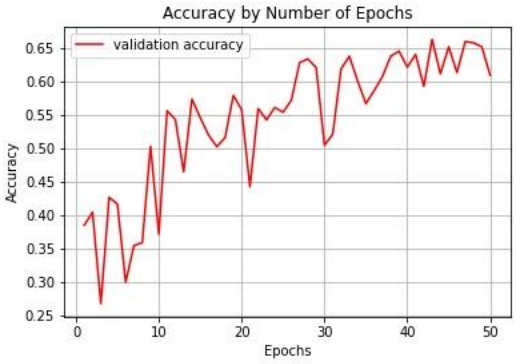

Figure 6 - SGD optimizer 50 Epochs

Task 8: Determine the optimal number of epochs to train for.
We can see from the graph that the accuracy increases rapidly until just before 10 epochs, at which point the accuracy continues to increases slow and steadily. For the sake of time, 10 epochs should be enough time to capture the spike in accuracy, while not being too long to get just a few percentage points higher.

Task 9: This high variability may be caused by a learning rate that is too high.
This is a gradient decent algorithm, and the gradient should to converge to zero with ideal parameters.
Currently the learning rate is set to 3, and perhaps the model is overcorrecting the bottom of the loss curve each time.
I would try reducing the learning rate to something that more easily converges to zero without taking too much longer to compute.

Task 13:

1. We could increase the number of words in our vocabulary by decreasing the min_freq parameter. This would give us more words in our dataset, which could assist with the accuracy of our model.
2. We could implement k-fold cross validation resampling to optimize the training of our model, which could assist with its accuracy.
3. We could run an ensemble of classifiers in addition to BoW, such as BERT and word2vec, and use the model with the greatest properly-fitted accuracy.

pos-tagger.ipynb Plots

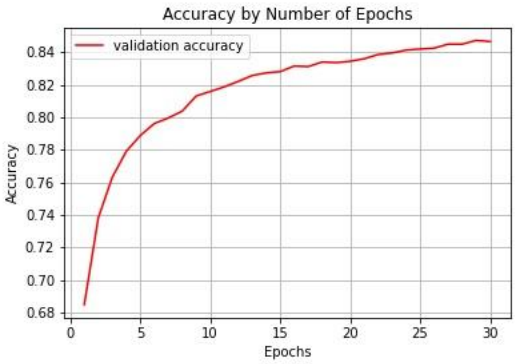

Figure 7 - Adam optimizer 30 Epochs

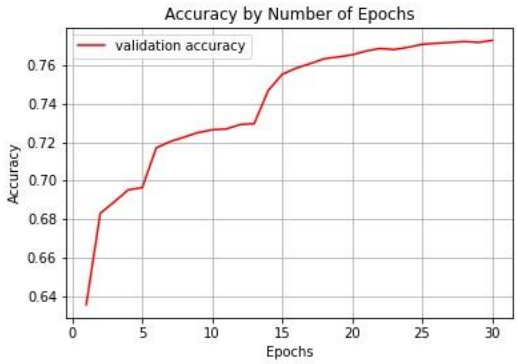

Figure 8 - Adam optimizer, Pretrained Embedding, Frozen, 30 Epochs

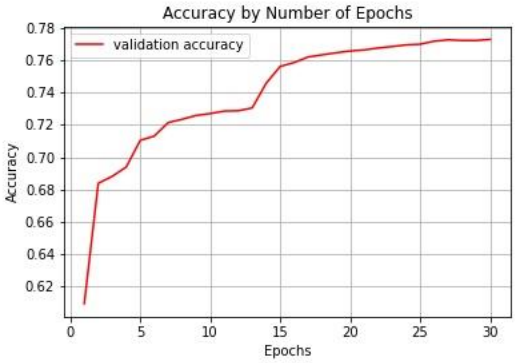

Figure 9 - Adam optimizer, Pretrained Embedding, Unfrozen, 30 Epochs

Tasks 6 and 7:

The performance of the frozen embeddings is noteably quicker. On my local machine, training took approximately 4 seconds per epoch for the frozen embeddings, while the other embeddings took approximately 145 seconds per epoch. This intuitively makes sense since no vector update is performed during each iteration of the frozen embedding matrix training model.

rnn-lang-model.ipynb

Notes on perplexity:
Perplexity is the probability of the test set, normalized by the number of words.
Minimizing perplexity is the same as maximizing probability.
Perplexity is "on average, how many different words could come next?"
If 10 words could come next each with 1/10 probability, then perplexity is 10.
This is an average of the weighted probabilities. "average branch factor"

Task 4:

Small Development Mode: (written before full version completed)
Sentence 6 outperforms all of the other sentences with the lowest perplexity.
This appears to be nonsense because the model is evaluating a set of words that we know make sense in each short sequence, but would never fly as a proper sentence concatenated together. Based on improper training, I suspect the short sequences are confusing the model.

The model also impressively increase the perplexity after swapping the word "world" for "cat" as if to note that it makes more sense for us to be talking about world (plural implied) herd immunity to the virus, rather than a single cat.

I do not have an explanation for why sen5 has a lower perplexity than sen1 - sen4.
I suspect this is an error of running in Developer mode with a lack of training.

Full Version:
I was not able to run on Google Colab using GPU. I did however run the full version on my local computer. Runtime: approximately 12.5 hours. The perplexities for each of the sentences make a lot more sense now. Results shown below in Task 5. sentence 1 had the lowest perplexity, which I would agree with. Sen3 was the runner up, which I would again agree with because the word “run” can be used in a way similar to crushing. Understandably, there is again a noticeble increase in perplexity when cat is substituted for world. Sentence 5 is jumbled nonsense, but appears to at least try to follow proper ordering for parts of speech. Sentence 6 is doesn’t even try to be a comprehensible sentence, and agreeably it received the highest perplexity by a long shot.

<u>Task 5:</u>
I was able to run the "Full Version" for my model on my local computer (12.5 hour runtime), since I could not get the Google Colab GPU working. Understandably, I do not have time to rerun this locally for the various dropout, GRU/RNN/LSTM, LSTM layers, and gradient clipping options to find the optimal combination using the full version. Instead, I tried finding the optimal combination of parameters using the small development version, then followed up with what was hopefully the best combination for the full version. My results for perplexities are below.

Small Development Mode:
Perplexities after running the models with various numbers of layers:

NUM LAYERS		LSTM	LSTM	LSTM sen1-3		LSTM sen4	LSTM sen5	LSTM sen6	RNN	RNN	GRU	GRU	average
2	1974	1784	1899	21.82	1627	762	1881	1679	2179	1942	1574.882		
3	2216	1974	2145	2552	1852	816	2024	1788	1867	1665	1889.9		
4	1973	1757	1863	2210	1668	774	1922	1720	1971	1739	1759.7		
5	1862	1658	1805	2162	1625	787	1970	1744	2050	1807	1747		
6	2208	1978	2048	2439	1773	807	2097	1837	1962	1734	1888.3		

2 layers appears to be optimal.

Perplexities after running the models with various Dropout values:

Dropout	LSTM	LSTM	LSTM sen1-3		LSTM sen4	LSTM sen5	LSTM sen6	RNN	RNN	GRU	GRU	average
0	1908	1721	1715	1963	1573	677	1872	1676	1887	1688	1668	
0.25	1934	1744	1767	2039	1588	710	1848	1660	2105	1885	1728	
0.5	1974	1784	1899	2182	1627	762	1881	1679	2179	1942	1790.9	
0.75	2069	1872	2069	2409	1718	847	1973	1753	2119	1909	1873.8	
1	1995	1847	2103	2440	1743	997	2075	1860	2112	1925	1909.7	

Dropout = 0 appears to be optimal.

Perplexities after running the models with various clipping values:

Grad Clip	LSTM	LSTM	LSTM sen1-3	LSTM sen4	LSTM sen5	LSTM sen6	RNN	RNN	GRU	GRU	average
0	1995	1847	2103	2440	1743	997	2075	1860	2112	1925	1909.7
0.5	1911	1722	1718	1968	1569	681	1851	1661	1907	1723	1671.1
1	1908	1721	1715	1963	1573	677	1872	1676	1927	1738	1677

There was not much difference between 0.5 and 1.0 clipping, but both were better than 0 clipping.

I would choose 2 layers, 0 dropout, and 0.5 clipping.

Full Version (Using 2 layers, 0 dropout, and 0.5 clipping):

LSTM	LSTM	LSTM sen1	LSTM sen2	LSTM sen3	LSTM sen4	LSTM sen5	LSTM sen6
377	442	1163	1257	1206	1657	12980	55678