Homework 2 - NLP (MSAI-337)

James Wilkinson, Josh Cheema, Lili Barsky, Devyani Gauri, Kaleem Ahmed

We implemented two neural networks (feedforward and LSTM) to create a language model using the Wikitext-2 corpora. After spending a substantial amount of time tuning our models, we performed sanity checks by inspecting the predicted output tokens our models were generating. We learned that the output was often common but less meaningful tokens, such as “a”, “an”, “the”, which is not in line with the true objective of a language model, despite yielding good perplexities. We then went backwards and considered several methods of preprocessing the corpora, including removing stopwords, tagging numerical values including dates/months/years, and tokenizing on words along with a subset of the major punctuation marks (periods, commas, exclamation marks, but excluding various symbols such as “@”, and the double-equals “==” that marks passage titles). We experimented with and without tied embeddings, various forms of regularization, and other architecture choices including penalizing our models from predicting words relative to the frequency of those words. As a bonus, we also implemented a vanilla RNN system. All of this is discussed in further detail below.

Feed Forward Network

Our feed-forward network was implemented using the PyTorch Lightning platform. We used a batch size of 20 with a context of 5, proceeding through an embedding layer, 1 hidden layer, followed by batch normalization, a non-linearity, and dropout, before going back through the same embedding layer to produce a probability distribution over the vocab, as was done in the Bengio model.

Embedding Layer	~28,000 x 100
Linear1	500 x 100
Batch normalization	1 dimensional
Nonlinearity	tanh
Dropout	p=0.8
Decoding Layer (weights tied to encoding)	100 x ~28,000

We experimented with a number hyperparameters, focusing early on on the analysis of the corpora performance using the feedforward network due to its relatively fast runtime. We tested various learning rates, degrees of dropout regularization, different weightings of cross entropy loss assigned to each class along with experimentation with the tokenization of the corpus. The experiments into learning rates and dropouts are illustrated in Figure 1.

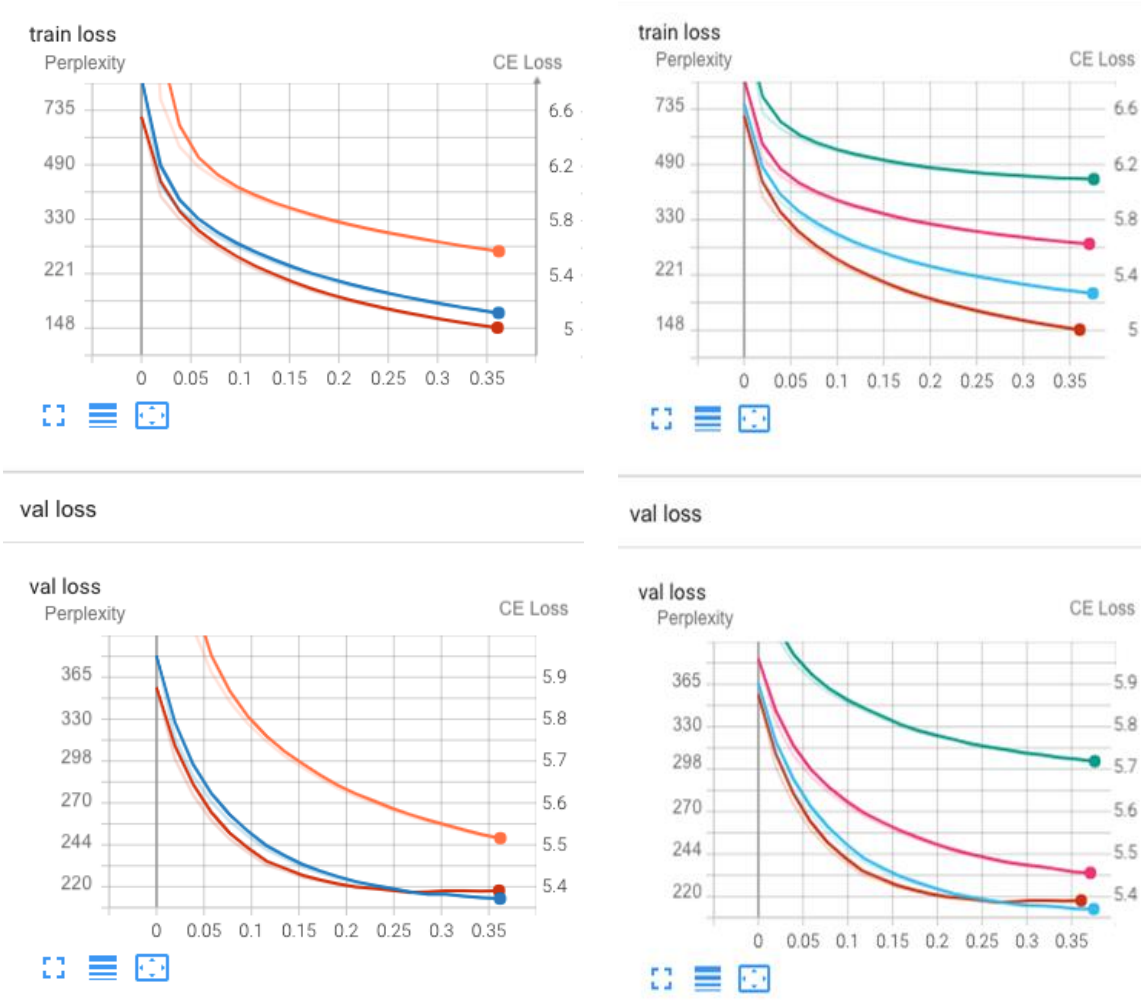


Figure 1: (Left) Feed Forward network learning curves resulting from learning rates of 1e-4 (orange), 5e-4 (blue), 1e-3 (red). (Right), learning curves for varying dropout values of 0.0 (red), 0.2 (blue), 0.5 (pink) and 0.8 (green). Note the y-axis is linear in cross-entropy space, and exponential in perplexity space.

From Figure 1, we can see that a learning rate of 5e-4 (blue line, left) produces an optimal result for the feedforward network by learning faster than slower rates (orange, left) whilst also converging at a lower loss than higher learning rates (red, left) and yielding the lowest perplexity. Our analysis into dropouts showed that a dropout rate of 0.2 was optimal (blue, left), preventing the overfitting that can be observed with no dropout at all (red, right) whereby the validation loss fails to improve even though training loss continues to improve.

We note that quick testing of our results (by decoding the sentences and predictions in the test data to plain english) revealed that the models were learning to predict the most common tokens without much regard for context. For example, continually predicting the period token “.” would grant a model at least one correct prediction per sentence, which could superficially appear to be a good result when considering the cross entropy loss and perplexity (with the average sentence at around 20 tokens, you can make 5% of predictions correctly by guessing “.” every time!). We highlight a number of example outputs of the FF network to illustrate this in

Text 1, which demonstrates the network is learning a bias to output the most common tokens in the corpus (such as stop words, common punctuation, and <unk> tokens).

theatre in the london borough (of) [of]
bill , as character connor (price) [,]
of <unk> and fulham . (in) [the]
in <year> <unk> landed a (role) [<unk>]
review of <unk> s performance (in) [,]
as darren , in the (<year>) [<year>]
<unk> s hamlet) , (robert) [and]
<unk> in the play <unk> (written) [.]
in the london borough of (<unk>) [the]
theatre in the london borough (of) [of]

Text 1: A series of examples of 5-token samples from the test dataset, along with the next token (in curved-parentheses) and the predicted next token (in square-parentheses).

To prevent this behavior, we took two approaches. First, we removed stopwords from our corpus. Next, we altered our cross entropy objective to weight tokens according to their sparsity (the reciprocal of their frequency) within the training corpus. We call this a *balanced* weighting, and note that the weightings of classes will follow a power-law under this regime according to the well-known Zipf's Law. We also experimented with an in-between weighting, which was calculated as the logarithm of their sparsity, noting that the class weights in this regime would be approximately linearly dependent on the frequency-rank of tokens. In simpler terms, we have constructed two approaches to address our model's tendency to predict only the most frequent

words in a corpus - an aggressive “balanced” approach, and a more mild “log-sparsity” approach. The effect these methods have on the model’s predictions are shown in Text 2.

of a series which featured (different) [prominently]
as <unk> fletcher . <unk> (starred) [hatfield]
on the <year> episode of (the) [zagreb]
guardian noted , ben <unk> (and) [pasupathy]
paris <unk> . career <year> (<year>) [homosexuality]
hands . <unk> starred as (scott) [hearing]
and fulham . in a (review) [variety]
fletcher . he portrayed an (emergency) [jamaaladeen]
productions of the philip ridley (play) [robotics]

<unk> , harry kent , (fraser) [and]
<year> in <year> <unk> had (a) [been]
. <unk> received a favorable (review) [review]
the independent on <dayofweek> described (him) [the]
factory in london . he (was) [was]
starring role on the television (series) [series]
critical reviews in the herald (,) [,]
in <month> <year> . he (had) [was]
how to curse was performed (at) [by]
performed in <year> at the (royal) [age]

Text 2: Feed-forward predictions using balanced cross-entropy class weightings (top). Using log(sparsity) cross-entropy class weightings (bottom).

In Text 2, we can see the balanced regime does result in the model making more diverse predictions. However, these predictions are mostly gibberish and make no semantic sense. On the other hand, the log-sparsity regime (bottom of Text 2) shows promise. In fact, predictions are not only more diverse, but also make sense in the context and are also often correct when compared to the ground truth. As such, in general, we proceed by using log-sparsity-based weightings on our dataset. In addition, learning curves for the different weight-regimes are shown in Figure 2.

Our final model therefore consisted of using log-sparse class weights in the objective function, a dropout of 0.2, and a learning rate of 5e-4, producing the following results.

Best Feed Forward Network Results	Perplexity
Training set	228
Validation set	221

Test set	217

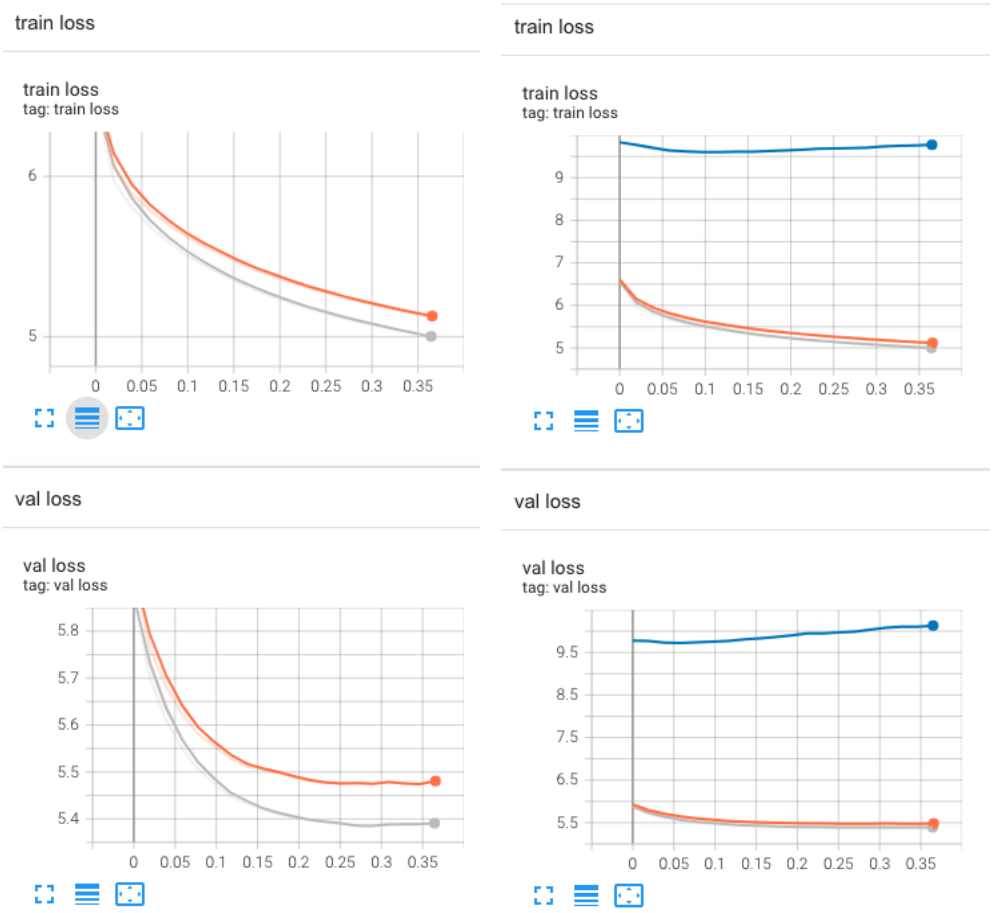


Figure 2: Loss curves for the feed forward network when considering no weightings to the target token classes (grey), log(sparsity) weightings (orange) and balanced weightings (blue). Note the charts on the right are identical to those on the left, but including the balanced weightings which disproportionately affect the y-axis scale.

LSTM

Our LSTM neural network was also constructed utilizing the PyTorch Lightning platform. It is characterized by 30 time steps, 2 hidden layers, with randomly initialized linear encoding/decoding layers with tied embeddings. The encoding/decoding layers are

approximately 28,000x100 in dimension. We implemented cross-entropy as our loss function and used the Adam algorithm for optimization, which was found to significantly improve the rate of convergence compared to SGD. We used a log-sparse weighted cross-entropy loss from our investigations with feedforward networks, noting that this naturally increased the loss and perplexity associated with our model since it prevented the model from capitalizing on “easy-wins” by blindly predicting common tokens. Upon our initial findings that the LSTM was still learning to predict <*unk*> tokens too often, we also implemented a functionality to remove sequences from the train, validation and test sets that contained above a certain proportion of <*unk*> tokens (ie: if more than 40% of a 30-word sequence comprised of <*unk*> tokens, then we exclude it from the dataset). We also removed stopwords and punctuation from the corpus, in order to encourage the model to learn words with genuine semantic meaning. We experimented with different learning rates, dropout rates, and gradient clipping values. A plethora of alterations and runs were completed over the course of the week, of which the most presentable are shown here.

First, we test three learning rates, 1e-4, 1e-3 and 1e-2, based on our intuition for what constitutes a reasonable range for this parameter. Over the week, many other learning rates were attempted, and anecdotally we found the optimum to be in this context. We discovered that the higher learning rate showed signs of overfitting when considering the validation curves (Figure 3). We progressed with a learning rate of 1e-3 based on the indication that it learns correctly, and doesn’t suffer from the poor learning curve of 1e-2, whilst also training faster than 1e-4 which appears not to reach any convergence within 20 epochs. However, whilst proceeding with a learning rate of 1e-3, we learned we must remedy the overfitting with regularization techniques.

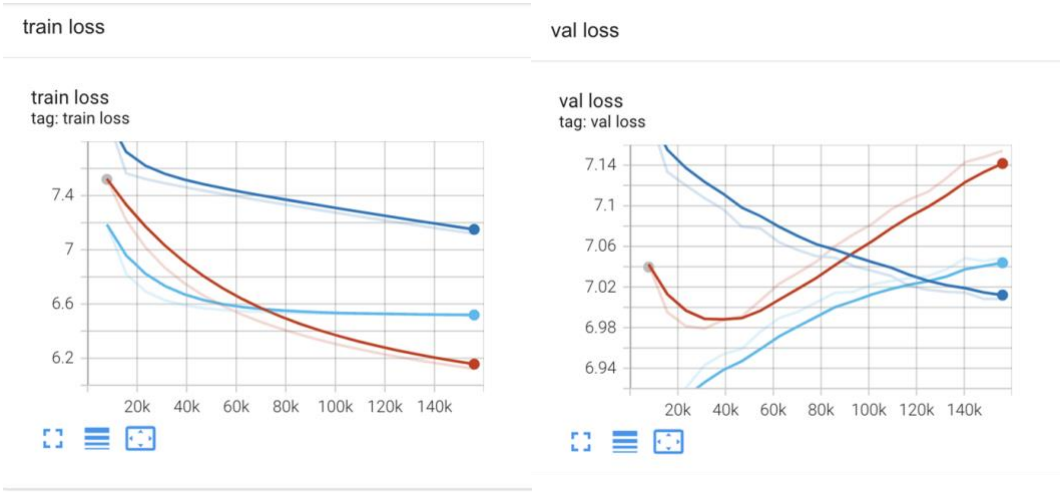


Figure 3: LSTM learning curves (CE loss or logarithm of perplexity on y-axis) for learning rates of 1e-2 (light blue), 1e-3 (red) and 1e-4 (dark blue).

We progress to testing the effect of gradient clipping and dropout on the LSTM (figure 4, left). Notably, we noticed that it did not significantly affect the overfitting of the model with a learning rate of 1e-3. However, a parameter of 0.5 is still marginally beneficial.

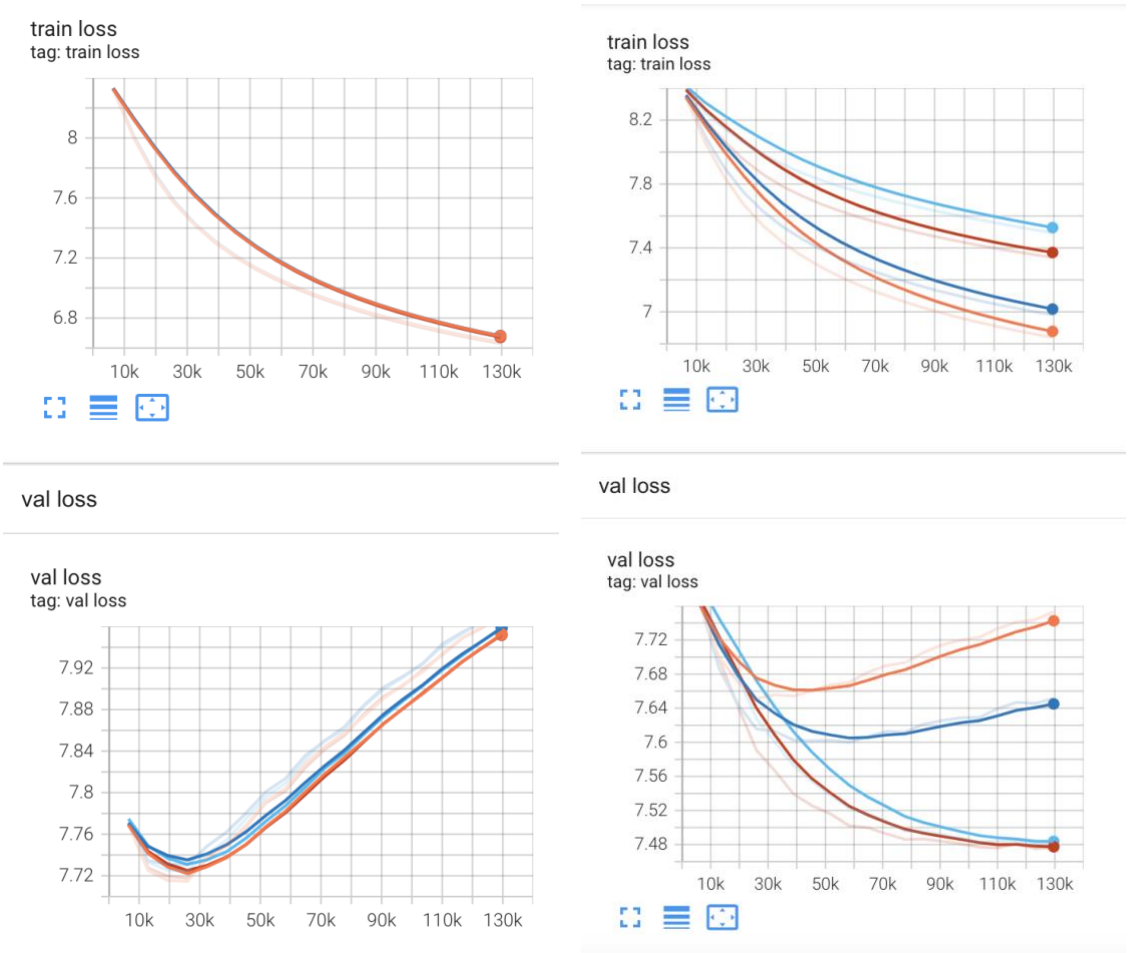


Figure 4: (left) Learning curves (with cross entropy loss on y-axis) for various gradient clipping values: 0.1 (dark blue), 0.5 (orange), 1.0 (red), 1.5 (light blue). (right) Learning curves for various dropout values: 0.1 (orange), 0.3 (dark blue), 0.5 (red), 0.7 (light blue).

Finally, we explore differing values for dropout rates (figure 4, right). We can see that dropout considerably lessens the effect of overfitting in the model, with higher dropout corresponding to less overfitting. We can see an optimal dropout value of 0.5 (figure 4, left, red line) above which the convergence of the validation data appears to show no improvement but the speed of convergence is lessened. Thus, our final implementation of an LSTM at this point uses the following additional architectures: a dropout-rate of 0.5 (50%), a gradient-clipping of 0.5, and a learning rate of 1e-3.

LSTM Network Results (1)	Perplexity

Training set	1600
Validation set	1650
Test set	1650

Given these results, we remark on the result that the feed-forward network performed significantly better than the LSTM when considering their losses. This is intuitive to us given the changes to the corpus we made between testing the feed-forward model and the LSTM model, which arguably made the prediction task significantly harder for the LSTM. Our increased measures to force the LSTM model to predict the less-common tokens in the corpus naturally make the language model's task more difficult, resulting in a much higher loss. However, regardless of the cause, this perplexity is relatively unacceptable for a language model.

After much deliberation, we decided to cutback on our preprocessing measures designed to create a "smarter" language model to attempt and create one that had good perplexity scores. We no longer removed stop words, sequences with high proportions of *<unk>* tokens, or punctuation, and we did not weigh the loss-function, in order to see how the model performs on a more neutral baseline. We did employ the word-tokenizer built into python's *NLTK* library. Our results are shown in Figure 5.

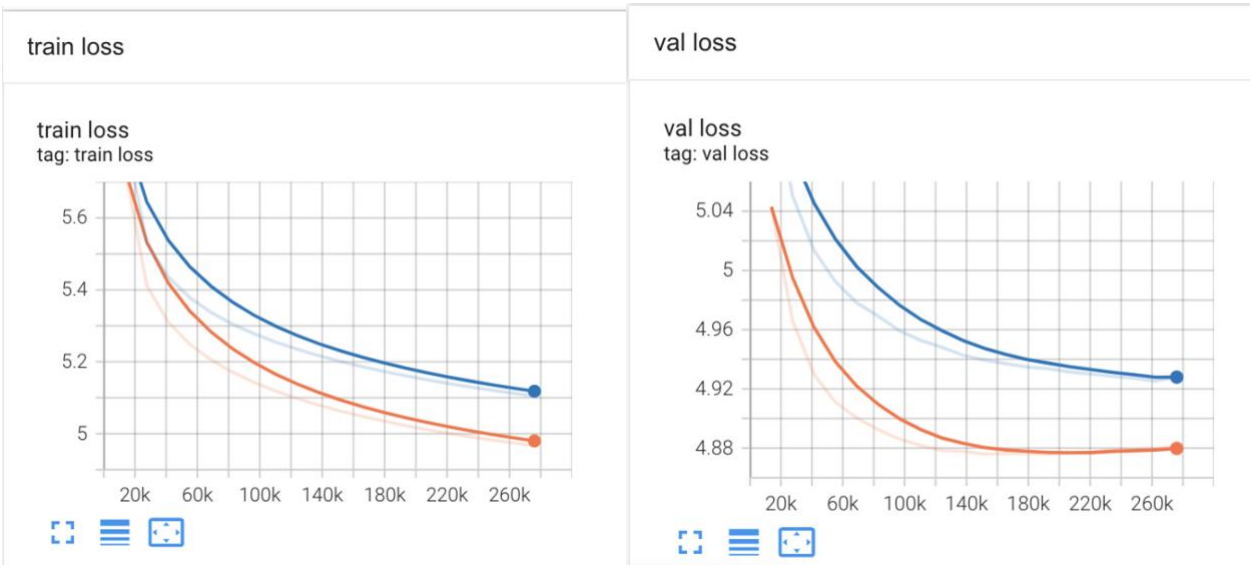


Figure 5: Learning curves (with y-axis showing cross-entropy loss) for an unedited corpus, tokenized by NLTK's word-tokenizer. We use both a log-sparsity weighted cross-entropy loss (blue), and an unweighted cross-entropy loss (orange).

As expected, the loss on our LSTM is significantly improved. However, we strongly believe that this improved loss (and low-perplexity) is not an indication of a better language mode, but in the spirit of this assignment, we present our best performing model below:

LSTM Network Results (2)	Perplexity
Training set	148
Validation set	134
Test set	130

RNN

Additionally, for the benefit of greater familiarity with deep learning platforms and using them to produce language models, we created a vanilla RNN, again using PyTorch Lightning. We went into greater depth with the feedforward model and LSTM model as these were the two main ones we chose to focus on, but our tuning of the RNN went through a similar process. We tried various rates of dropout, learning rates, optimizers, and tested model performance with and without punctuation-type tokens and stop words. Here is the architecture of the network:

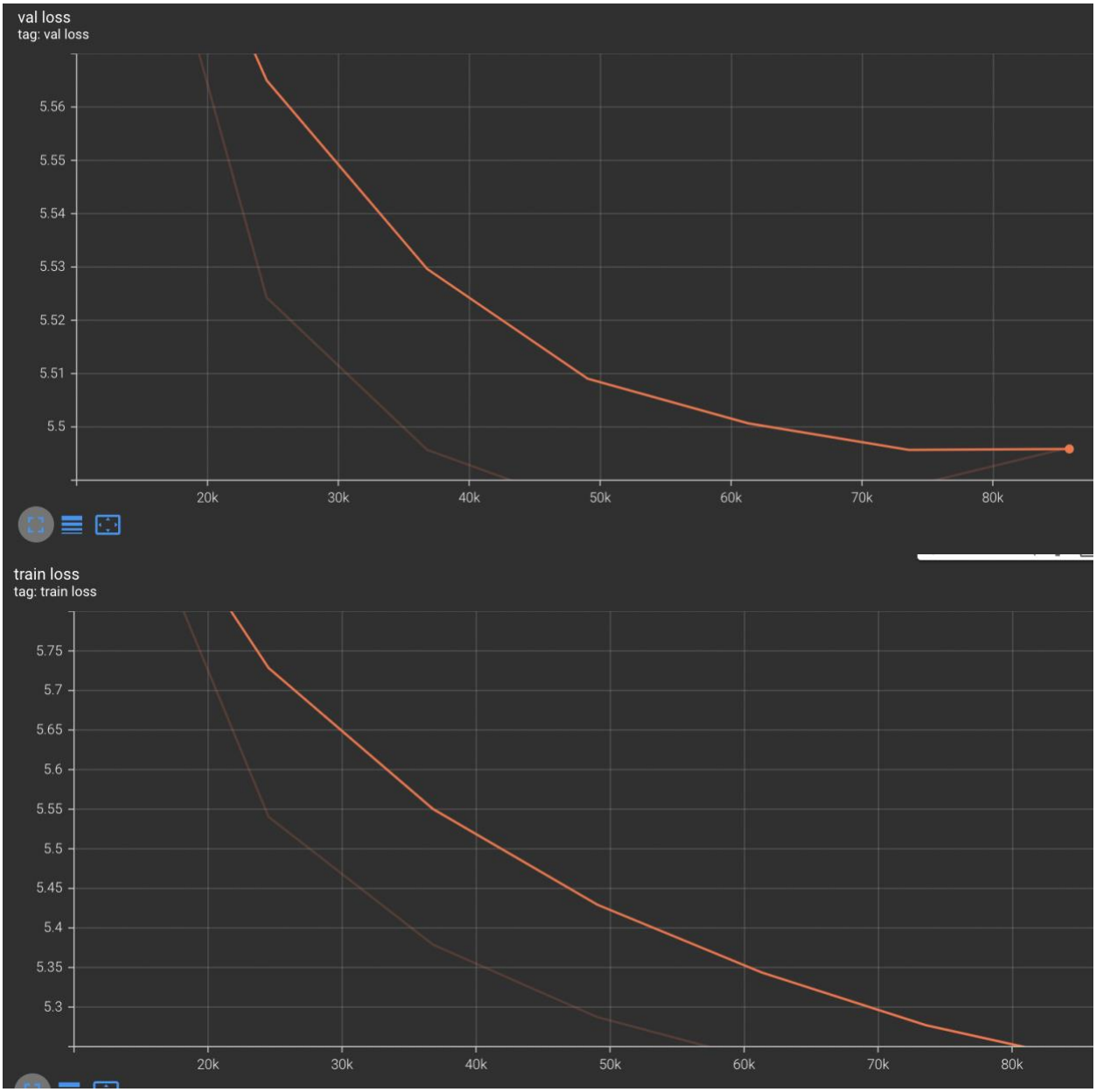
Embedding Layer	~28,000 x 100
RNN	Input size = 100, hidden size = 100, num layers = 2
Nonlinearity	tanh
Decoding Layer (weights tied to embedding)	100 x ~28,000

We stuck with an Adam optimizer with a learning rate of 1e-3. We initially removed stop words in an attempt to make a more useful model and also removed numbers, tagging them (<year>, <int>, <decimal>, etc) as we did in our prior assignment, but noted the loss/perplexity were quite high and we had significant overfitting. We did various things to try and combat this. We tried various forms of regularization, including dropout, L1 regularization, L2 regularization, and gradient clipping. Regardless of L1 regularization, L2, a combination of both (elastic net), and dropout of p=0.8, our model would still overfit, with training loss clearly decreasing and test/validation loss hitting a plateau around 7.2 (perplexity 1340). We also experimented with reducing the embedding and hidden size in conjunction, to continue to allow for tied embeddings in the way we implemented them, at the same time as reducing the total parameter count. We also tried relu as the nonlinearity, a bidirectional RNN, and adding more stacked layers. We took more liberty with experimentation as this model was not technically required for the assignment. We also weighted the loss function in the same way we did with the LSTM, discussed above.

In the end, we struggled to create a model that performed well while removing stop words and tagging numbers in an attempt to make a useful RNN, as described about in the LSTM section. However, in doing minimal pre-processing, using dropout 0.7, gradient clipping of 0.5, Adam optimizer, 1e-3 LR, we generated a model that at least had acceptable perplexity/loss on our dataset. It was stopped after 6 epochs for overfitting.

Best RNN Results	Perplexity
Training set	169
Validation set	242
Test set	228

Here are the learning curves from the best RNN.



If we had chosen a lower learning rate and done more to battle overfitting, we likely could have lowered the perplexity even further, as the training loss had not plateaued in the above curves.