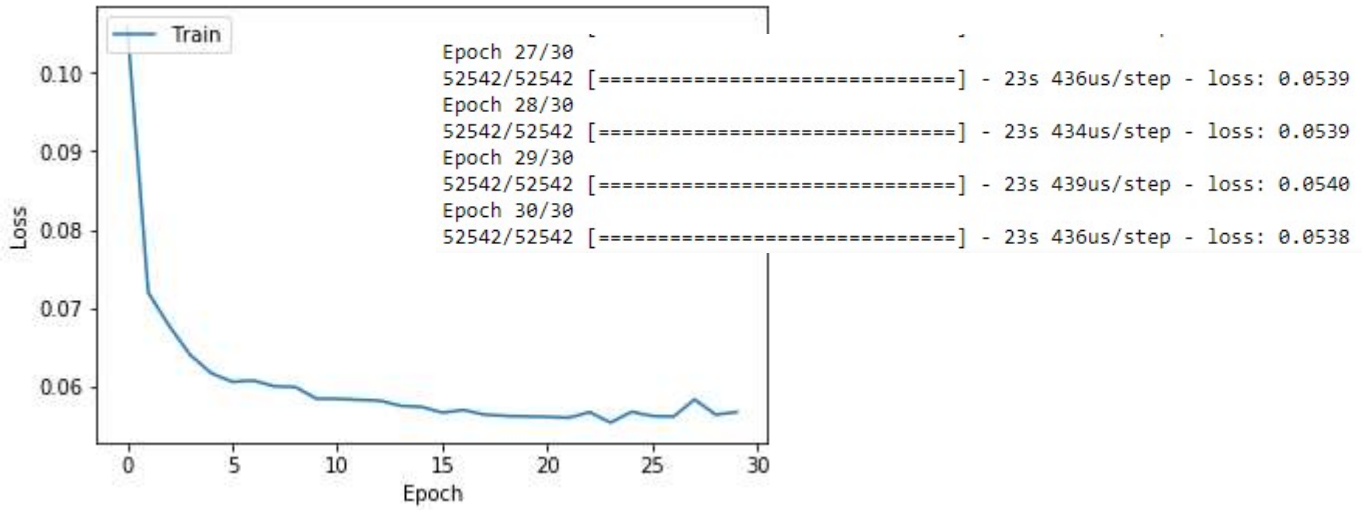
Angelene Arito
Kaggle Username: **AngeleneArito**
Ranking: 19 (public), 18 (private)
CSC578 – 910
Final Project
6/10/19

Overview of Final Model

My journey through training and model building for this project was a bit scattered and I actually started off experimenting with slightly more complex models right out of the gate – models that included multiple layers, dropout, bidirectionality, etc. – but wound up with a final model that was far simpler. The model that produced the lowest score for me in the Kaggle competition was:

```
model = Sequential()
model.add(LSTM(50, input_shape=(timesteps, data_dim)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mae')
model.summary()
```

which trained with a batch size of 25 at 30 epochs.



To get to this final model however, I tried many of the suggested advanced features/architectures in some of my first iterations in an attempt to find a complex model that did a good job of predicting temperature.

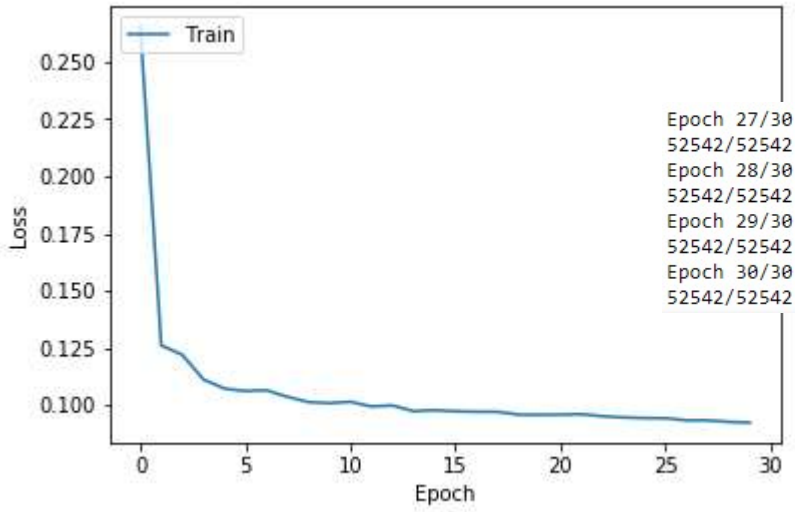
Experimentation

Stacked LSTM Layers

First, I experimented with multiple recurrent LSTM layers. I tested everywhere between two recurrent layers to four recurrent layers but stopped adding beyond that because the epochs were starting to run more slowly and the loss was not improving with more layers. Even at two layers, the overall loss scores over each epoch were not much improvement over a single layer. In addition to testing the multiple

stacked recurrent layers, I also experimented with the parameters within those layers. I mostly kept batch size around 32 consistently, but I did adjust the number of units – testing mostly between the 20-50 range. I also added in dropout (just normal, not recurrent) and tweaked that within the range of 10%-50% dropout. Loss numbers on training epochs consistently demonstrated that higher levels of dropout (at various numbers of layers – two, three, and four layers) resulted in worse predictions, which was also validated by higher error scores upon submission to the Kaggle competition. I ruled out any levels of dropout above 20% moving forwards, but it seemed as though 0% dropout was actually optimal – and didn't seem to cause overfitting.

Below is one example of a stacked model I tested, with four LSTM layers, each set to a size of 50 units with 20% dropout at each. Epochs ran on the slower side taking upwards of around 1 minute 15 seconds each to run. Learning in the first few epochs had rapid improvement, but after that stayed pretty stagnant.




```
Epoch 27/30
52542/52542 [==============================] - 78s 1ms/step - loss: 0.0932
Epoch 28/30
52542/52542 [==============================] - 79s 1ms/step - loss: 0.0932
Epoch 29/30
52542/52542 [==============================] - 81s 2ms/step - loss: 0.0926
Epoch 30/30
52542/52542 [==============================] - 80s 2ms/step - loss: 0.0922
```

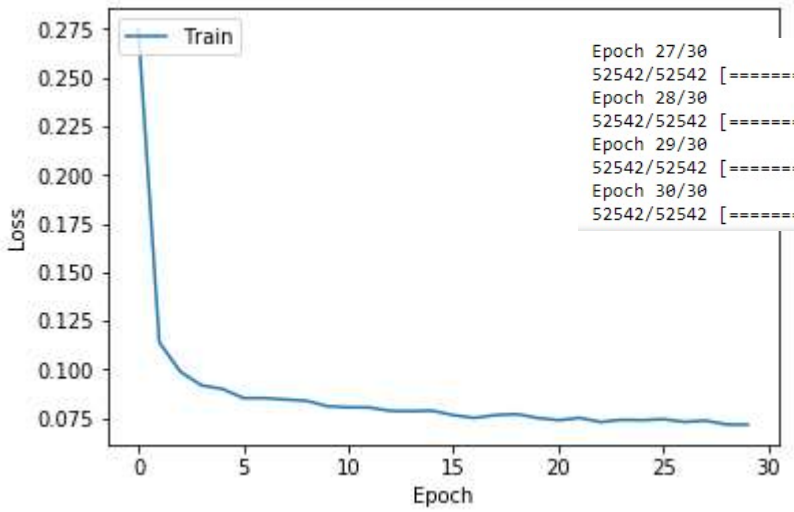
Bidirectionality

In a final effort to try and make stacked LSTM recurrent layers work, I experimented with bidirectionality. Again, I was performing different training iteration based on two, three, and four stacked LSTM recurrent layers. I tested adding in a bidirectional layer as the last layer initially, and then moved it around a little, experimenting with it also as the first layer and as a middle layer as well. In one iteration I added a bidirectional layer between each LSTM layer. Number of units for each layer I mostly kept around 50 and experimented with 20% dropout at each layer vs no dropout at each layer. These experiments resulted in slower running epochs with not a lot of improvement in loss scores – they only appeared to make the model more complex with not a lot of return.

On the simpler side of things, I also experimented with a bidirectional layer next to a single LSTM layer and achieved better results when run against the test data in the competition as opposed to multiple layers, but still not better scoring than the single LSTM layer by itself. I kept the batch size consistent at 32 for this testing and experimented back and forth between 20% dropout and no dropout.

Below is an example of one of the bidirectional iterations I tested out, in this case with three LSTM layers and a single bidirectional layer after the three LSTM layers. Units were set to 50 across the board

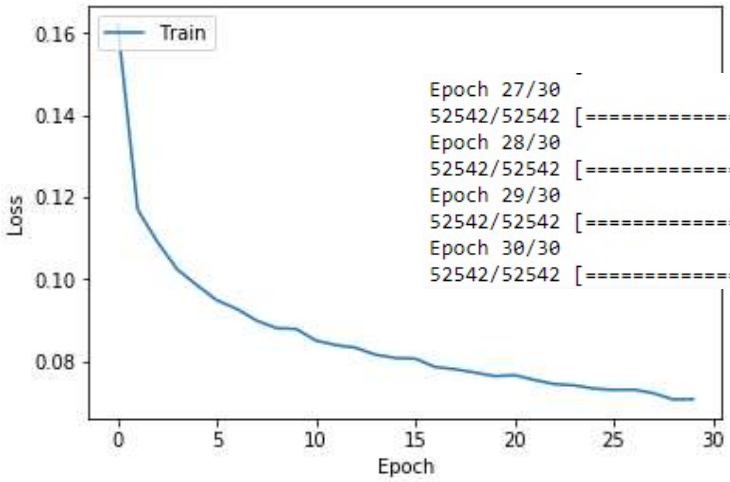
and I did include 20% dropout on each LSTM layer. The epochs ran on the longer side and the loss didn't get quite as low as other models (primarily the one I chose).




```
Epoch 27/30
52542/52542 [==============================] - 88s 2ms/step - loss: 0.0732
Epoch 28/30
52542/52542 [==============================] - 91s 2ms/step - loss: 0.0738
Epoch 29/30
52542/52542 [==============================] - 87s 2ms/step - loss: 0.0719
Epoch 30/30
52542/52542 [==============================] - 87s 2ms/step - loss: 0.0718
```

GRU Layers

After having experimented with LSTM layers only up to this point, I shifted focus to building GRU layers. I started off simple with a single GRU layer and while still keeping the batch size consistent at 32, I mostly tweaked the number of units in the layer – trying units like 32, 40, 50, and 64. I implemented dropout as well, but the resulting models were not proving to be as good as those created using LSTM. Below is one of the iterations of GRU I tried – one with a single GRU layer, 32 units, and 20% dropout. It ran on the faster side per epoch, but the initial learning rate seemed slower and the final loss scores at the end of 30 epochs were not as good as the LSTM layers.

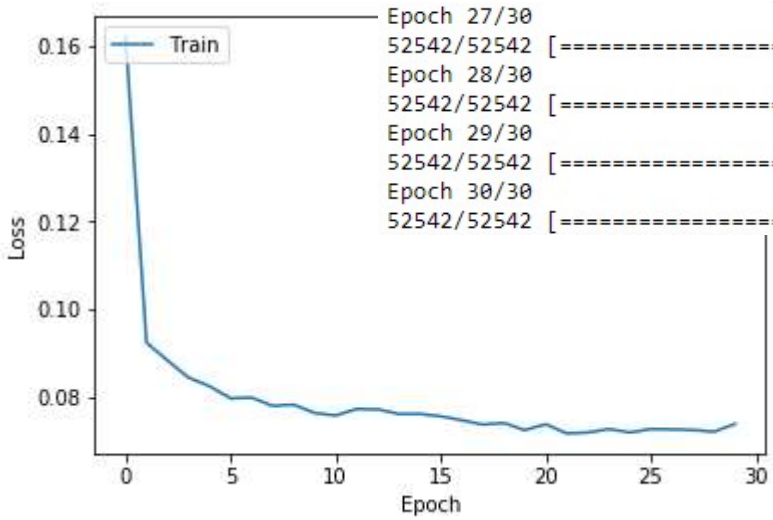



```
Epoch 27/30
52542/52542 [==============================] - 19s 356us/step - loss: 0.0730
Epoch 28/30
52542/52542 [==============================] - 19s 367us/step - loss: 0.0721
Epoch 29/30
52542/52542 [==============================] - 19s 357us/step - loss: 0.0706
Epoch 30/30
52542/52542 [==============================] - 19s 354us/step - loss: 0.0707
```

CNN/RNN Hybrid

After experimenting solely with recurrent layers up to this point, I decided to try adding convolution to the mix, either adding a single convolution and pooling layer on top of a GRU layer or adding multiple of each. I tweaked number of units throughout and a few other parameters like activation functions, but

none of what I experimented with came back with a model that was producing better scores on loss or in the competition. Below is an example of one of the models I tested with a single convolution later with kernel size 2 and a pooling layer with size 2. After that layer I used a GRU layer with 50 units and 20% dropout. Epochs ran on the faster size, but loss scores were not an improvement.




```
Epoch 27/30
52542/52542 [==============================] - 14s 258us/step - loss: 0.0726
Epoch 28/30
52542/52542 [==============================] - 14s 267us/step - loss: 0.0725
Epoch 29/30
52542/52542 [==============================] - 14s 264us/step - loss: 0.0721
Epoch 30/30
52542/52542 [==============================] - 14s 260us/step - loss: 0.0739
```

Statefulness

I did perform a few experiments with a stateful network, but nothing extensive. From my understanding you can only run a stateful network with a batch size that can be divided into the number of samples (both train and test) and there isn't any number that can be divided into both 52542 and 17447 aside from 1, so the run time was astronomical, making it seem like an impractical model for this problem.

Single Layer

The most successful experiments I ran on this project cam from single LSTM layers where I was just played around with the number of units in the layer, dropout, and batch size.

Runner Up Models

Two runner up models that had decent performance would be the final model I submitted, only with a batch size of 75 instead or the model with the single LSTM layer and one bidirectional layer:

```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(timesteps, data_dim)))
model.add(Bidirectional(LSTM(50, activation='relu')))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mae')
model.summary()
```

Reflection

The single hardest part about this assignment was the pre-processing; initially it took a long time to even figure out pseudo code in my head as to what I needed to accomplish in terms of formatting. However, it did help a great deal to have data already pre-processed (the test set) so there was a point of

comparison to break things down and understand what was needed. The instructions were helpful in pointing out what values each row/column should contain and like in other assignments, having the final expected dimensions outline helped - it at least let us know the target we were trying to hit (rather than being blind). Even with the instructions and explanation though, it took me the better part of a week (maybe even longer) to wrap my head around how the data needed to be transformed initially. While I do understand that formatting the data to be RNN ready is an important part of the process, it was very frustrating to feel like I couldn't advance in the project for a while because my pre-processing was off; I had no idea if my poor Kaggle performance had to do with the models I was constructing or my pre-processing being off. I was definitely one of the people who could not get their submission score even close to below 1 (or 1.5 even for that matter) for a long time, even though I thought I was on the right track with my model building. I don't know that I would have suggested something different in terms of set-up, but I do wish I was able to spend more time on the modelling piece itself rather than trying to figure out what was wrong with my pre-processing.

I think overall the project was a fair level of challenging – it wasn't impossible, but it also wasn't easy. I do wish though that the course had been a little less theoretically-based and more implementation-based. I know that the preface on day one was that it was hard to strike a balance between the two, but it would have been easier to do this project I think if we had a taste of more coding on assignments up to this point. Yes, we did have assignments that were in keras, but not a lot of them felt like they quite prepared me for the final project, except for homework 6. That said, I do feel like if the goal of this course was to cater to both people who wanted more theory and people who wanted more application, it struck a pretty decent balance.