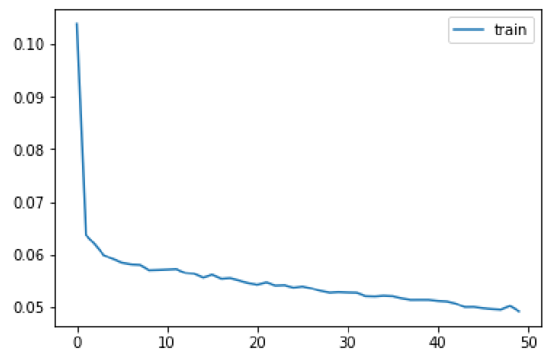
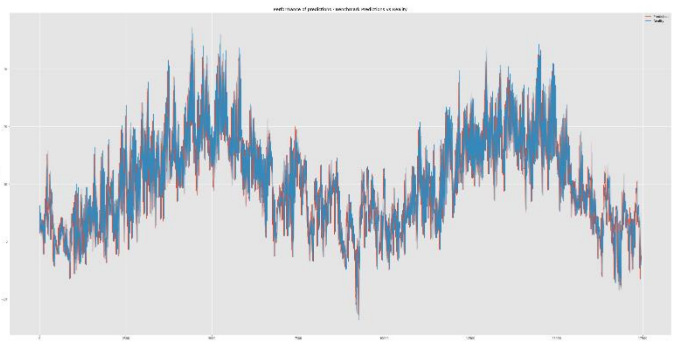
Kaggle Username: Will Chirciu   Public & Private Ranking: 4th

CSC 578 In-Class Final Project

Single Layer

The goal of this project was to find the optimal model to predict the temperature for the next hour given the data for prior hours using recurrent neural networks. The optimal model will seek to minimize the mean absolute error (MAE) on the test data. The journey getting to the final model was a long one. It started off with single-layer Long Short-Term Memory networks (LSTM). LSTMs are a type of recurrent neural network that can learn long-term dependencies to be incorporated in processing 'current' inputs. If we could get away with a simpler model, it would help both in interpretability and in run time. LSTMS were run with unit values of 50 or 100. Units are the dimensionality of the output space. Ideally, we would set a range of values (10 – 200 in increments of 10 for example) to test using a grid search. However, for the sake of time, the previously mentioned units were used. These models were trained across different numbers of epochs and the best results were found with a lower number (at around 50). The best model initially was an LSTM with 100 units training over 50 epochs. It achieved a MAE of 0.57611 on the full test data according to Kaggle.





The graph on the left shows the training process for this basic model. As you can see it minimizes the mae slightly above 0.6 on the first epoch and steadily decreases over the remaining 49 epochs. The process looks to be fairly stable. The graph on the right simply compares the actual temperatures in the test set with the predicted values. Both the predicted and actual values follow a similar pattern.

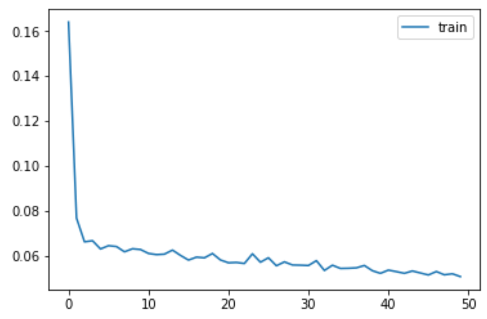
Once a decent LSTM model was found, an attempt was made to improve these results using a Gated Recurrent Unit (GRU) network. A GRU is just an LSTM without an output gate. It has been known to work well on smaller datasets. However, using GRU with similar parameters as LSTM, there was no real improvement in the error as can be shown in the results table.

In a SimpleRNN, the output of a fully connected RNN is fed back to the input. Using this model, there was once again no real improvement. As far as single layers go, LSTM, GRU, and

SimpleRNN all returned similar MAE scores. LSTM gave slightly better results, so it was used as the base model to be improved upon.

Stacked Layers

The next objective was to see if adding more layers would lead to improved results. LSTM was used for each layer due to its decent performance. Because of the increased training time from using multiple layers, GRU and SimpleRNN were not tested in stacked layers. The number of units were chosen arbitrarily. The best MAE obtained was 0.60108. The first layer was an LSTM with 100 units, the second an LSTM with 50 units, and the third an LSTM with 25 units.

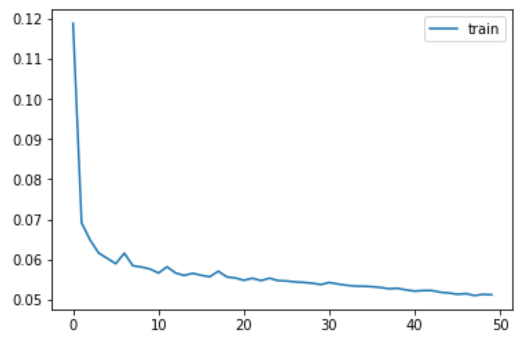


The chart on the left shows the training pattern for this stacked model. In comparison to the single layer model, the training is slightly less stable. We see a lot more spikes across the epochs. The training speed looks to be the same however, as we are still using the tanh activation function.

0.6 MAE could not be broken with stacked layers. Therefore, it was concluded that more complex, deep networks are not the way to go for this dataset.

Bidirectional RNNs

A Bidirectional RNN connects two hidden layers of opposite directions to the same output. This allows the output layer to retrieve information from past and future states simultaneously. This type of RNN is particularly useful when context of the inputs is needed (in handwriting recognition for example). Using a bidirectional LSTM with 50 units, an MAE of 0.58676 was obtained.

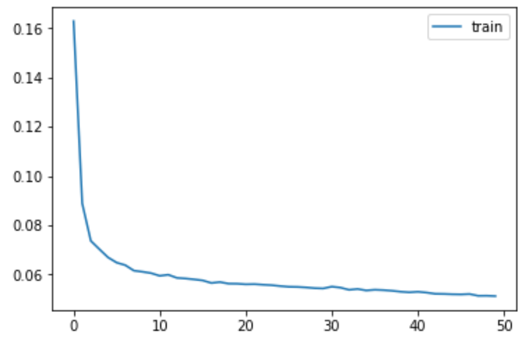


The chart depicting training loss values across successive epochs on the left shows slight instability at the beginning of the training process. However, after around the 20th epoch the curve smooths out and steadily decreases.

There was no significant improvement in the scores using Bidirectional RNNs over single layer basic RNNs to warrant use, especially with the increased run times. It doesn't seem like this data would benefit from this type of network as it doesn't really require context of the inputs as much as handwriting would.

CNN + LSTM

Using convolutional neural nets allows us to extract features for the LSTMs to interpret across time steps. The least amount of time was spent trying to optimize this network layout due to its increased run time and large number of combinations of hyperparameters. Only a few models were run and the only one recorded managed to obtain an MAE of 0.62690. We start with a 1-D convolutional layer with 28 filters of size 4 using the ReLU activation function. Once the features have been extracted, the output feature map is sent to the LSTM layer with 100 units.



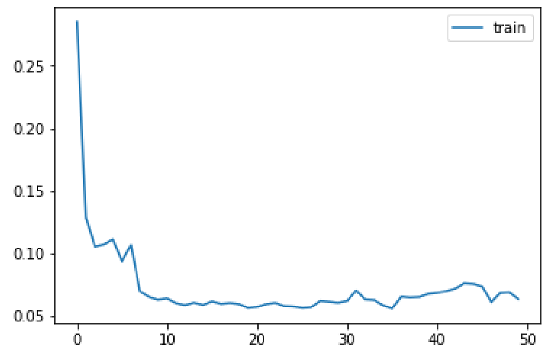
The training process was very stable as can be seen in the chart on the left. However, from the results there is no reason to use CNNs in conjunction with LSTMs for this dataset. I think it is because too much information is lost when the data is run through the convolutional layer. Either that, or the hyperparameters need to be explored a bit more. Because of the increased number of parameters, grid search would be the most feasible option to explore this. Due to time constraints in getting this project completed in time, this option was not pursued.

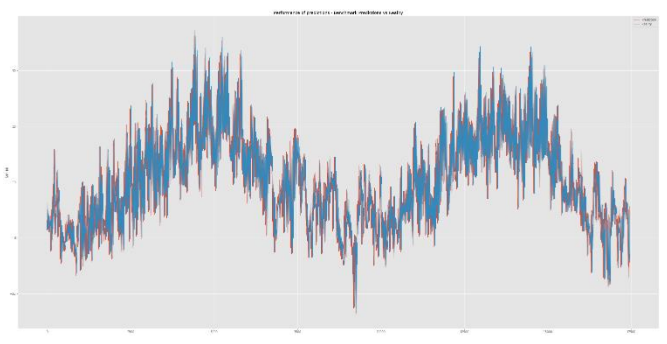
Using Dropout

After running a bunch of these models, the training error was consistently 10 times smaller than the test error. Because of this, using dropout would ideally close the gap as it had in the previous assignment when dealing with convolutional neural networks. However, as can be seen in the results table, network layouts that implemented dropout had MAE values significantly higher than any other model (around 2.0 MAE). After doing a bit of research, several issues might explain this. The first is that with some of these models, dropout was applied right before the last layer. This might not give the network time to correct for errors created by the dropout. Second, the network was probably not given enough time to converge. These models were run with around 50 – 200 epochs. If the model hasn’t converged in that time, it would likely return poorer results. If dropout were to be reimplemented, these are the avenues to be explored.

Final Model: Stateful LSTM

Instead of training the entire network on the full training set, it might be useful to do this in batches, where the weights obtained in each batch are used to update the weights in the next. In the case of LSTMS, a sequence of a batch might be dependent on the sequence of another batch. To create a stateful LSTM, the number of training instances needed to be divisible by the batch size. The optimal model was a single layer stateful LSTM with 50 units and was trained using a batch size of 378 instances. After 50 epochs, a MAE of 0.51115 was achieved. Training these networks in batches led to the biggest improvement in MAE by far.





This came as a bit of a surprise because the training process appeared to be quite unstable. The training loss across successive epochs graph appears to take a very weird shape, more unique then the rest of the network types. The Actual vs Predicted chart however, forms a similar shape as our base model as the MAE is only around 0.06 lower. The only explanation as to why the stateful LSTM worked was that the dependencies between each sequence (378 instances of 24 time-steps) was actually significant.

Overall Comments: I really enjoyed this project. I like experimenting with different parameters to see what works. I must admit that Recurrent Neural Networks are still fairly confusing to me and I felt a lot of the time I'm running these models in a black box since Keras is doing all of the work. I was pleasantly surprised to once again make it to top 5 in the leaderboards. I am eager to learn what other people's models looked like to see if we came to a similar or completely different conclusion. If I had more time, I would play around with CNNs a lot more because I think it has potential. Also, from what I've read, stateful LSTMs are useful in stacked layers so that is another path of potential improvement.

There were two major challenges I had in completing this project. The first was in preparing the training data to be Keras ready. Thankfully the documentation in **[1]** assisted me in this as it was working with a very similar example. The second was in getting the stateful LSTMs to work. I was really confused how to implement the batch size and batch input shape. The documentation in **[2]** showed various ways to use different batch sizes when training and predicting LSTMS. I chose to copy the weights from the training model to the test model. This eventually led to my optimal model.

Ultimately, I think this project had just the right amount of difficulty. We got to be hands-on in building one of the more advanced neural networks and it was encouraging to see people in the class help each other on Piazza. If I had more time, I would probably do more with the exploratory data analysis and see if any feature selection/engineering could be done to improve the models.

Results Table:

Model	Details	Epochs	MAE
0	LSTM(50)	50	0.57997
1	LSTM(50), LSTM(50)	50	0.62664
2 **BASE**	LSTM(100)	50	0.57611
3	LSTM(100)	150	0.62555
4	LSTM(100)	100	0.63917
5	GRU(50)	50	0.62597
6	GRU(100)	50	0.63512
7	GRU(100, dropout = 0.2)	50	1.38017
8	LSTM(50), LSTM(100)	100	0.63779
9	LSTM(50,dropout = 0.3), LSTM(100,dropout = 0.2)	100	2.07177
10	SimpleRNN(50)	100	0.56874
11	SimpleRNN(100)	100	0.57556
12	SimpleRNN(50)	500	0.63706
13	SimpleRNN(50, ReLU)	100	0.60585
14	LSTM(200)	100	0.61714
15	LSTM(100),LSTM(50),LSTM(25)	50	0.60108
16	LSTM(100),LSTM(100),LSTM(50)	50	0.62592
17	Conv1D(28,4,relu),LSTM(100)	50	0.62690
18	Bidirectional LSTM(50)	50	0.58676
19	Bidirectional LSTM(50,dropout=0.3)	50	2.06843
20	LSTM(100,stateful,batch_size = 378)	50	0.55387
21	GRU(100,stateful,batch_size = 378)	50	0.54595
22	GRU(200,stateful,batch_size = 378)	100	0.64507
23	GRU(50,stateful,batch_size=378)	50	0.55234
24 **FINAL**	LSTM(50,stateful,batch_size=378)	50	0.51115
25	LSTM(50,stateful,batch_size=189)	50	0.55433
26	LSTM(50,stateful,batch_size=837)	50	0.58723

References:

[1] “Multivariate Time Series Forecasting with LSTMs in Keras”
URL: https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

[2] “How to use Different Batch Sizes when Training and Predicting with LSTMs”
URL: https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/