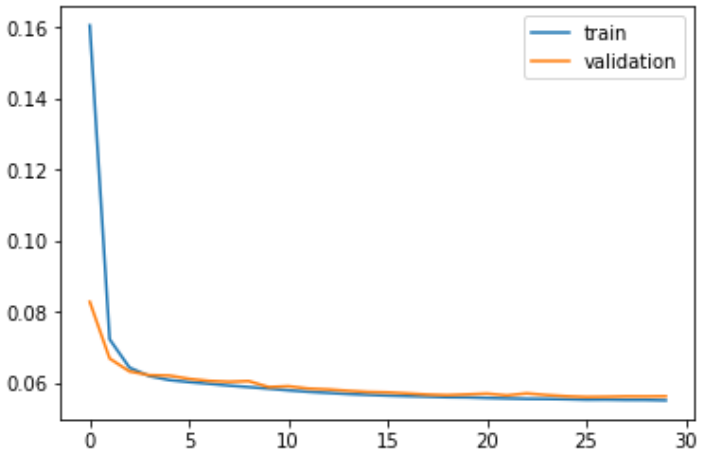
Steven Jordan
CSC 578 - Section 910 Online
06/10/2019

Project Video: https://youtu.be/fubxAUt8kik

Kaggle Username: Steven Jordan |Ranking: 11

Final Competition Model

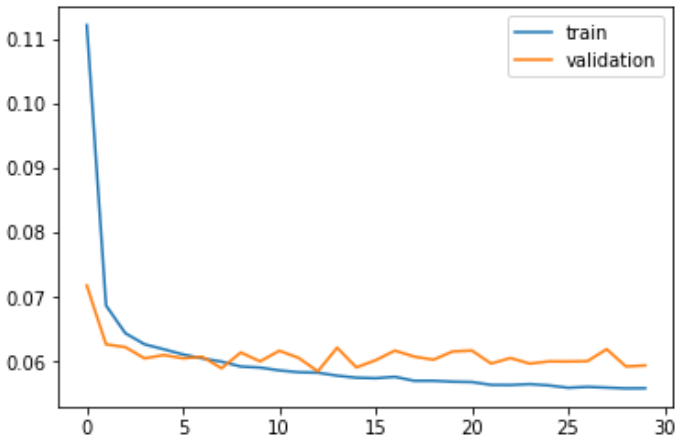
Figure 1. Final Competition Model



Over the course of this final project, I created close to three dozen models and submitted 23 of them to Kaggle. After the final, private scores were announced (that were measured against 70% of the held-out test data), I ranked 11/41 with a Mean Absolute Error of 0.53920. However, to my *extreme* surprise – this best of my RN N models was literally the simplest: the input layer is a single LSTM layer with 10 units and there are no hidden layers. It was tested over 30 epochs, and as one can see in Figure 1, there was very little overfitting, with the final validation loss (0.0562) only slightly greater the training loss (0.0551). There were no changes to the activation function (the default is tanh), and there was no recurrent dropout included as part of the model.

Other Model Attempts

Figure 2. CNN + GRU Combination



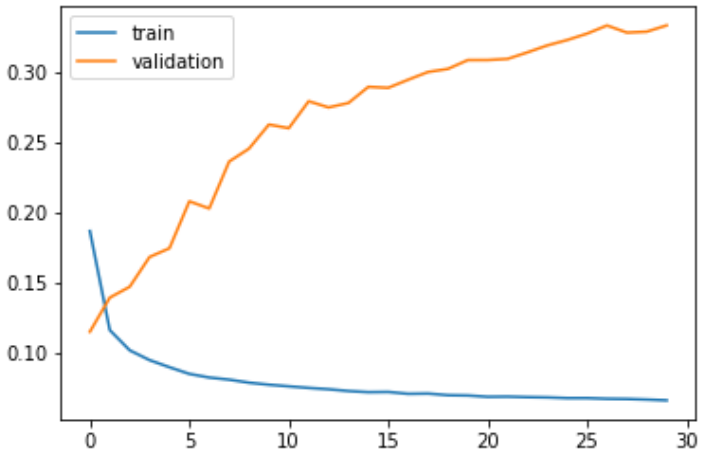
Prior to the final scores being released, my best scoring model in 30% test set (MAE of 0.53676) was another simple model – nearly identical to the best model, but with a hidden LSTM layer with 10 nodes. For both the first and second layers, the activation function was changed to RELU. It's final training and validation loss were 0.0556 and 0.0567 respectively – so not that different from the winning model. I had made numerous unsuccessful attempts to improve upon this model, for example:

A model that adds a CNN layer before the RNN. This model has an Input Convolution 1D layer with 32 filters, a kernel size of 5, and uses the RELU activation function. The second layer was a Gated Recurrent Unit, also using the RELU activation function. It led to similar results in the validation (Figure 2) but a slight drop in the test MAE (it dropped to 0.53937). However, as one can see in Figure 2, that there was some likely overfitting with final training and validation losses of 0.0559 and 0.0594 respectively.

After seeing the overfitting of that model, I figured I would attempt to reduce the overfitting by introducing a recurrent dropout of 0.4 in the GRU layer, and a dropout layer after the CNN layer of 0.4.

However, this combination dramatically reduced the accuracy of the model, and to my surprise, it *dramatically* increased the level of overfitting, as seen in Figure 3. The MAE after submission to Kaggle (I expected it to be poor) was 1.61471.

Figure 3. CNN + GRU Combination + Dropout



Journey of Training and Model Building

I began my model building journey by replicating the model provided in one of the reference resources (Brownlee, 2017) that was provided on the Assignment Page. My first model attempt had an MAE of 2.66666, which did not raise any red flags, since it was my first attempt; it was the same model as my winning model, but with 50 units. Over the course of a week, I continually added to and modified the model by adding more recurrent layers, combining it with CNN layers, experimenting with bidirectional layers, dropout, and statefulness. I submitted about a dozen models to Kaggle, but no matter what I did, the best I could improve my MAE was to 2.55660. I had double and triple checked my training data and scaling – everything seemed correct. However, it wasn't until the final weekend in which I realized that my training data, while had the correct rows, was concatenated in a different way than the provided test_data (without columns labels it was hard to know, since the fourteen characteristics were repeated in the same manner). Once I realized that, I fixed the code, went back through the process at step 1, an LSTM layer with 50 units, and my original model provided a *much-improved* MAE of 0.55515.

I then went through different iterations, once change at a time. I started with changing the number of units in the LSTM layer – an increase in units increased the MAE, and a decrease decreased the MAE – which resulted in my (now) model. I attempted many changes – first individually to see if I could increase my score incrementally. I first stacked two recurrent LSTM layers and produced a better score – as described in the "Other Model Attempts" section of this documentation.

My attempts to improve the model (unsuccessfully) included: Adding dropout, stacking more recurrent layers, using GRU instead of LSTM, introducing bidirectionality, changing the normalization methods to both log and MinMaxScaler (since on Piazza, the professor indicated that the de-scaling values may be incorrect), and testing different activation functions. I did some research on neural networks and activation functions and learned that Google "discovered" an activation function called SWISH (Google, 2017) that supposedly is more effective than RELU. It is not yet built into keras, so I learned to implement it using code found on a Github

Figure 4. Model with Statefulness

```
Train on 42033 samples, validate on 10509 samples
Epoch 1/15
42033/42033 [==============================] - 385s 9ms/step - loss: 0.0796 - val_loss: 0.0705
Epoch 2/15
42033/42033 [==============================] - 389s 9ms/step - loss: 0.0623 - val_loss: 0.0662
Epoch 3/15
42033/42033 [==============================] - 393s 9ms/step - loss: 0.0606 - val_loss: 0.0637
Epoch 4/15
42033/42033 [==============================] - 386s 9ms/step - loss: 0.0597 - val_loss: 0.0593
Epoch 5/15
42033/42033 [==============================] - 384s 9ms/step - loss: 0.0594 - val_loss: 0.0613
Epoch 6/15
42033/42033 [==============================] - 392s 9ms/step - loss: 0.0589 - val_loss: 0.0614
Epoch 7/15
42033/42033 [==============================] - 391s 9ms/step - loss: 0.0588 - val_loss: 0.0589
Epoch 8/15
42033/42033 [==============================] - 376s 9ms/step - loss: 0.0586 - val_loss: 0.0590
Epoch 9/15
42033/42033 [==============================] - 382s 9ms/step - loss: 0.0584 - val_loss: 0.0605
Epoch 10/15
33096/42033 [======================>.......] - ETA: 1:17 - loss: 0.0583
```

page (Hsieh, 2017). I introduced statefulness and a batch input of (3, 24, 14) – and began training a model. However, after about 2 hours of training (and getting through only 10 epochs), it was clear that the validation loss was not improving after the 3rd epoch (Figure 4), so I stopped that training session to use my remaining time more wisely.

Reaction and Reflection of My Result and Competition

I am incredibly surprised by my result in the competition – for two reasons. For one, I am surprised that my simplest model ended up being the most accurate one. I am questioning if that is the point? Is sometimes simpler better? Secondly, I am surprised by my high(ish) ranking in the class. I ranked 11 out of 41, and it was with one of my first models, so I am surprised that more didn't achieve the same or nearly the same score when that model was so simple.

That said, I would love to see what I may be missing, and would highly appreciate if the Professor could send out the winning two or three models to the class after they're graded.

Reaction and Reflection of the Assignment Overall

As noted above, the majority of my struggles and frustration with this assignment was due to my training set being formatted incorrectly. After my realization, and re-reading the instructions, I realize that the structure I used technically didn't violate any instructions, but I should have been better at understanding the intent. My recommendation to the Professor, for the next class, would be to either have the students separate the training and test sets (so if they use a different order of columns than the Professor's, it will have no impact on the final scores), provide the training set with the full 336 columns, or to provide the test set with just the 14 columns.

Other than that, I quite enjoyed this assignment – even though I am very confused and curious as to why my extremely simple model performed so highly. I definitely look forward to working with and studying neural networks more in the future!

References

1. Brownlee, Jason. "Multivariate Time Series Forecasting With Lstms In Keras". Machine Learning Mastery, 2017, https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/.
2. Google. Searching For Activation Functions. 2017, https://arxiv.org/pdf/1710.05941.pdf. Accessed 7 June 2019.
3. Hsieh, Johnny. "Johnny7861532/Activation-Function-Swish-In-Keras". *Github*, 2017, https://github.com/johnny7861532/activation-function-swish-in-Keras.