For our first model, we plan to use a Recurrent Neural Network (RNN) to implement our model. RNNs excel at time series analysis (predicting future results based on past data) because they are able to capture temporal dependencies. This method of implementation should be able to provide significant outputs based on our training data. In order to implement the RNN, we can utilize the TensorFlow library. The library has pre-existing architecture for implementing a simple RNN, GRU, and LSTM, so we can alter the type of neural network we’re using in future implementations without significantly altering the code or data. This allows us to get initial results in our first iteration while maintaining flexibility for later iterations.

Useful documentation and existing works:

[TensorFlow RNN Documentation](https://www.tensorflow.org/guide/keras/working_with_rnns)

[Time Series Prediction with TensorFlow](https://www.tensorflow.org/tutorials/structured_data/time_series)

[Time Series Prediction Paper (with some code)](http://www.pjoes.com/pdf-81557-32517?filename=Simulating%20and%20Predicting.pdf)

Our second model used an RNN implemented with TensorFlow, for the same reason we used it for the first model. We used two LSTM layers with batch normalization, making some changes to the model and data input to get more accurate predictions. First, we added an LSTM layer and increased the size of the LSTM layers in our model to better handle our more complex data. We also added batch normalization to stabilize the training process. Then, when handling the data, we normalized the features to prevent a small subset of the features from dominating all the others. These additional layers and normalization improved the training process for the new model, and allowed it to make more advanced predictions rather than the plus or minus 1 goal predictions in the first iteration.