Documentation for "RNNs, LSTMs, and the Capital Markets"

Motivation

Market prediction isn't just a problem, it is *the* problem. In fact, it is a problem in *n*-dimensions, which makes solving it almost impossible. In the mid 1990s, recurrent neural networks showed promise coping with the market prediction problem, however, they contained over 100 features, which makes them expensive to construct. Despite the difficulty of finding the right feature set, we were curious to know what a network like this might look like implemented using recent technology.

Key Takeaways

- Market prediction is an unsolved problem in n-dimensions and, thus, intrinsically fascinating.
- This is an exploration of the performance of recurrent neural networks and long-short term memory nodes on data from the U.S. capital markets.
- Data were limited to monthly frequency and could easily be expanded to alternate timesteps and augmented with economic or fundamental data.
- The purpose was primarily to understand the coding of the network and not create an engine for trading decisions (don't use this for trading).
- We concluded that simpler models (i.e., fewer hidden layers, shorter lookback periods) produced better results than more complex networks.
- The activation function turned out to have an outsized impact on our results with hyperbolic tangent yielding the worst performance and rectified linear unit performing best.
- Further research on the market prediction problem should focus on the ideal activation function, as well as the optimal amount and type of data.

• Findings

Per our goal of using a high-level API to construct a market model, our mission was accomplished. For the curious, our proxy for the market was SPY and not the actual S&P500, which can't be bought by non-institutional investors as it is too expensive.

That said, having first learned to code neural networks using Numpy, the process was far from easy. We used the Keras interface. While Keras does a masterful job of abstracting away the details of building a fully-connected graph and running small and large amounts of data through it, we caution the reader to leave his/her linear algebra at home. Though matrix operations are going on "under the hood," thinking one's way through Keras with the underlying matrices in mind can lead to madness. That said, anyone interested in building a wildly complex model for market prediction would do well to learn and use Keras.

Our findings were straightforward and can be partitioned in two: financial and computational. Per finance, we used an economy of data and modeled off monthly observations. Ideally, we'd have access to a socket of streaming market data and our model would look completely different. Still, our model did objectively perform well on our test-set data with a best of the bunch .36 MSE. Given the paucity of data used, this is an outstanding result.

We attribute our results to three factors. First, RNNs are highly sophisticated with an easy-to-use dropout to prevent overfitting. Second, we began to approach complexity from the right perspective (i.e., the number of hidden layers) and were quick to course correct when things went south. Lastly, we backed further away from adding complexity by decreasing the lookback period. It should be noted that market prognosticators and capital markets professionals almost constantly compare today's market action to what they believe is a comparable point in the – sometimes distant – past. Our results say this is a bad practice. More importantly, our findings suggest the simpler the model, the better (on financial data).

• Final Code

Below we present our model and its settings. Parts were inspired by MIT.S.191 – Intro. to Deep Learning.

```
# Define function to create LSTM model
def create_model():
   model = Sequential()
   model.add(LSTM(50, input_shape=(1, look_back),
activation='relu'))
   #model.add(Dense(20, activation='tanh'))
   model.add(Dropout(0.2))
   model.add(Dense(1, activation='relu'))
   model.compile(loss='mean_squared_error', optimizer='adam')
   return model


# let's look back one month
look_back = 1

# these are boilerplate values
epochs = 100
batch_size = 1

trainX, trainY = [], []
testX, testY = [], []
for i in range(look_back, len(train)):
   trainX.append(train[i-look_back:i, 0])
   trainY.append(train[i, 0])
for i in range(look_back, len(test)):
   testX.append(test[i-look_back:i, 0])
   testY.append(test[i, 0])

# Fit to LSTM input format
trainX = np.reshape(trainX, (len(trainX), 1, look_back))
testX = np.reshape(testX, (len(testX), 1, look_back))
trainY = np.array(trainY)
testY = np.array(testY)
```

● Final Performance

The model produced the following final output, after 100 training epochs. We've presented an abbreviated version below.

```
Epoch 1/100
112/112 - 1s - loss: 0.5669 - 559ms/epoch - 5ms/step
Epoch 2/100
112/112 - 0s - loss: 0.5610 - 82ms/epoch - 728us/step
Epoch 3/100
112/112 - 0s - loss: 0.5568 - 84ms/epoch - 750us/step
Epoch 4/100
112/112 - 0s - loss: 0.5558 - 84ms/epoch - 749us/step
Epoch 5/100
112/112 - 0s - loss: 0.5537 - 84ms/epoch - 749us/step
Epoch 90/100
112/112 - 0s - loss: 0.5529 - 89ms/epoch - 791us/step
Epoch 91/100
112/112 - 0s - loss: 0.5531 - 86ms/epoch - 767us/step
Epoch 92/100
112/112 - 0s - loss: 0.5522 - 86ms/epoch - 766us/step
Epoch 93/100
112/112 - 0s - loss: 0.5527 - 85ms/epoch - 756us/step
Epoch 94/100
112/112 - 0s - loss: 0.5529 - 86ms/epoch - 772us/step
…

Epoch 95/100
112/112 - 0s - loss: 0.5523 - 86ms/epoch - 772us/step
Epoch 96/100
112/112 - 0s - loss: 0.5524 - 87ms/epoch - 781us/step
Epoch 97/100
112/112 - 0s - loss: 0.5520 - 85ms/epoch - 760us/step
Epoch 98/100
112/112 - 0s - loss: 0.5524 - 86ms/epoch - 764us/step
Epoch 99/100
112/112 - 0s - loss: 0.5522 - 85ms/epoch - 762us/step
Epoch 100/100
112/112 - 0s - loss: 0.5532 - 87ms/epoch - 773us/step
4/4 [==============================] - 0s 1ms/step
2/2 [==============================] - 0s 2ms/step
Train Score: 0.55 MSE
Test Score: 0.36 MSE
```