Exploring the applicability of Seq2Seq architecture to Intraday Technical trading

Patrick Kelly
pkellyl@stanford.edu

Abstract

The performance of several applications such as machine translation, language modelling, sentiment analysis, chatbots and question answering has advanced substantially in the past 6 years. Main drivers of this progress have been the development of the seq2seq model architecture using RNNs and the subsequent addition complementary architectures such as attention and convolutional neural networks. The core ability of the seq2seq architecture is to develop either a prediction (like a sentiment score) or a predicted sequence (like a semantically and idiomatically correct translation of a text) based on an arbitrary input sequence. This paper explores the application of this ability to predicting the returns of a publicly traded stock price given a sequence of indicators. The notion that stock price behavior is random in the short term and in the best of cases explicable yet not predictable in the medium to long term is widely held. Besides legion theoretical underpinnings and explanations, the fact that if stock price behavior were predictable arbitrage opportunities paired with traders' ambitions would quickly do away with predictability. As economists say, there are no $100 bills lying in the sidewalk, because if there were, someone would have picked them up already. Sadly, we have not found any unattended currency on the path that this project has led us down. We nevertheless believe the contribution of this exercise is to confirm the notion of lack of arbitrage in the face of the appearance of novel technology. We must conclude that if the predictability we sought did in fact exist, then it has been arbitraged-away already by some secretive algorithm absconded in some quietly humming server in some hedge fund's data center. The $100 bill has already been picked up.

1. Introduction – seq2seq model and attention

Machine translation, language modelling, sentiment analysis, chatbots and question answering have all advanced substantially in the past 6 years. Sequence-to-sequence (seq2seq) models have been 'unreasonably effective' (to quote, I believe, Andrej Karpathy) in advancing the performance of these applications. The seq2seq encodes a sequence in an RNN, usually based on the LSTM or GRU model, and then decodes the encoded sequence as per a given objective. The objective may be the generation of a sequence (for instance a series of tokes representing the translation of the input sequence) or a specific value (like a sentiment score).

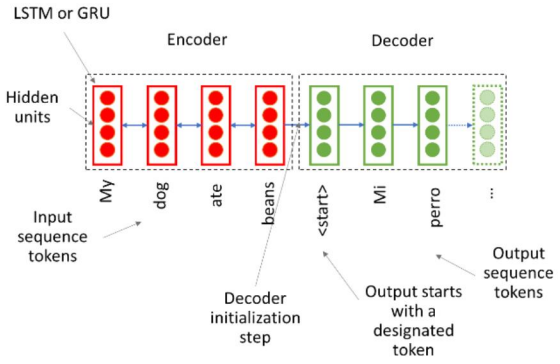

Figure 1 - Basic seq2seq. The seq2seq architecture encodes a sequence of arbitrary length and then decodes into another sequence of arbitrary lengths as per a given learned objective – here an English to Spanish translation is shown. The values of weights of the hidden layer of the encoder represent the input sequence and is passed to the decoder in order to govern the decoding process as per the input. In a typical architecture, the decoded sequence is initialized with encoder weights and a 'start' token. The decoder recurrently generates output tokens until a target length is reached or the decoder generates an 'end' token.

seq2seq performs well when compared to other RNN of fully-connected architectures. This performance has limitations, nevertheless. As the encored receives tokens it only has seen past items in the sequence. It therefore cannot access the full context of the sequence before encoding the next setp. Bidirectional RNNs, which encode in start-to-end and end-to-start simultaneously address this challenge. Another challenge comes with longer sequences, that are harder to encode as dependencies among tokens that are far apart and thus may have a hard time propating through time and influencing the encoder correctly. Lastly, the decoder does not have access to the encoder hidden layer weights at different timesteps. These may be of value in deciding the decoded token sequence.

Attention is and architecture that deals with the two latter issues. Attention in its basic form provides the decoder with a weighted sum of the encoder hidden layer inputs. Attention output is combined in a fully connected layer to decoder output in order to define the next token.

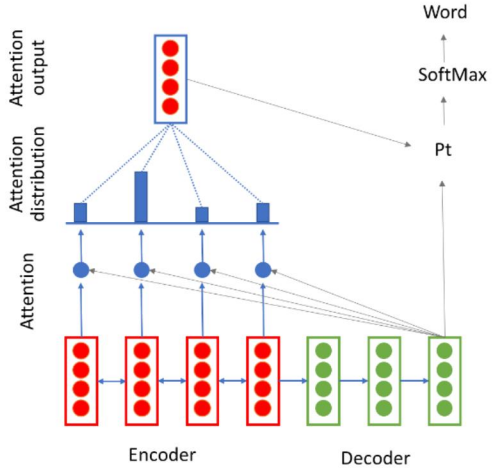

Figure 2 - seq2seq with attention. Attention in its basic form generates a weighted sum of the encoder hidden layers which is dependent on the decoder output. This weighted sum is then combined with decoder output to generate the next token. More complex architectures such as multiplicative and dot product attention can be implemented to gain flexibility at the expense of processing cost. Also, multi-head attention, i.e., several parallel attention structures, can be implemented in order to specialize each head in distinct attention tasks.

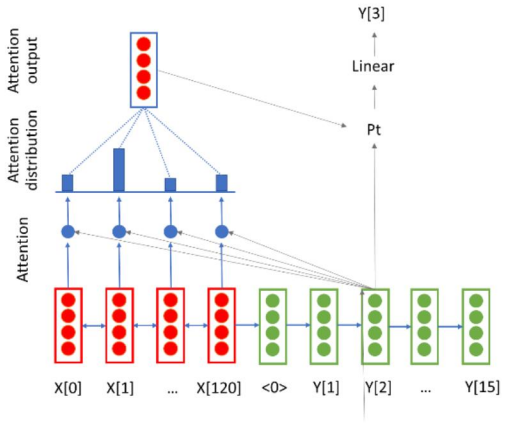

Figure 3 – Architecture for the project. A seq2seq model with a fixed length input and output. The input represents a timeseries of values related to a stock price. The encoder is a multi-layer bidirectional LSTM with dropout. The decoder is initialized with a 0 value and uses a monodirectional LSTM. Decoder length if 2 was used in experiments seeking to predict the return of a stock. Longer fixed lengths were used to try to predict sequences of returns.

2. Data preparation

One year of data for eight stocks was used. Data (> 200 million observations) represented the size and price of each trade for seven stocks (AMZN, JD, SFIX, SHOP, TSLA, TTD) and a market index tracking ETF (SPY). The idea behind selecting stocks from the tech sector of varying levels of liquidity and volatility was that each might capture distinct market dynamics. SPY, which track the S&P 500 was included to capture dynamics of the overall market that may help predict the behavior a out target stock. The target stock for predictions was arbitrarily selected to be TTD. Data was summarized by minute to generate 889 thousand observations that included number of trades, total volume, maximum price, average price, for a given minute. Data was limited to cover 9 am to 5 pm. The final dataset for the experiments was a minutely feature set that contained date, day of week, minute; and for each of the eight tickers returns, volume changes (relative), number of trades, maximum price. 35 features were thus rendered.

The dev and test sets were chosen to reflect the most recent data as well as to show data for each weekday. The test set was thus defines as the last week of data for may, the dev set was defined as the prior week. The dev and test sets contained 2400 observations each (480 minutes per day x 5 days).

3. Design considerations

Framework and hardware: The experiments were implemented on pytorch and run on AWS P3 instances which are equipped with an NVIDIA Tesla V100 GPU.

Data normalization: I decided not to normalize all data. I scaled data so as to make each features data range reasonably similar. Thus, values like max_stock_price, which could reach 200 were divided arbitrarily by 2000 to make them comparable in absolute magnitude with minutely returns that usually oscillate in the +-1.0 e-5 and +-1.0 e-2 range. I did not scale or normalize the returns feature as this is the value to be predicted by the decoder. Scaling and/or normalizing would make it harder for the decoder to learn the right objective or add a descaling/deformalizing step which would add needless complexity.

```
self.encoder = torch.nn.LSTM(x_size, hidden_size, num_layers = n_layers, bias = True, bidirectional = True)
self.decoder = torch.nn.LSTMCell(1 + hidden_size, hidden_size, bias = True)
self.h_projection = torch.nn.Linear(hidden_size * 2, hidden_size, bias = True)
self.c_projection = torch.nn.Linear(hidden_size * 2, hidden_size, bias = True)
self.att_projection = torch.nn.Linear(hidden_size * 2, hidden_size, bias = False)
self.output_projection = torch.nn.Linear(hidden_size, 1, bias = True)
self.combined_output_projection = torch.nn.Linear(hidden_size * 3, hidden_size, bias = True)
self.dropout = torch.nn.Dropout(p=dropout_rate)
```

Figure 4 – Layer definitions for the model

Neural network units: Layer definitions can be seen in figure 4. Again, because the nature of the experiment was to define an absolute quantity or series of quantities, all layers, save the attention_projection layer, were defined to have bias. Leaving bias out would make values for the same observations generate different outcomes depending on the batch and/or training epoch, shifting outputs and making learning impossible (as I found by experience!)

Regularization: Dropout was used with values of 0, 0.1 and 0.3. Given the noisiness of the data, it seems dropout did not have much effect.

Output activation: As the output range was expected to be in the range of minutely returns (+-1.0 e-5 and +-1.0 e-2) tanh was considered an option as it has a relatively linear behavior in that range. Sigmoid was discarded because its range 0-1 precludes predicting negative values. ReLU was also discarded for not being symmetrical about zero. Finally, after several tests, no activation was used in the final output. A linear layer output was used for its symmetry about x = 0 and its linear behavior.

Loss function: Two loss functions were tested: absolute, mean squared error (MSE). The absolute error was considered as the outputs were going to be << 1 and thus squared errors, I surmised, could be very small and perhaps not affect backpropagation much. Finally the MSE error was used as a smaller loss was achieved.

Training: Training was developed using random batches of features (selecting a random day and a random window within each day)

Single performance metric: The hypothetical trading gain that could be realized if the decoded sequence or value was used for trading.

3. Experiments

Experiment 1 - Predict a sequence: This consisted in encoding a sequence of the feature set with the objective of generating the sequence of returns that would follow. The input was the 34 feature set, and the objective was the returns for TTD (one dimension). Encoding windows of 60, 100, and 120 minutes were tested using decode windows of 5, 10, 15 and 20 minutes.

Figure 4 – Experiments. Here we see the two variants of experiment – predicting a sequence and predicting a cumulative return.

Experiment 1 results: After varying window sizes and testing hyperparameters (learning rate, hidden size, number of LSTM layers) it was not possible to generate a meaningful result. Error was > 0.1, and the system essentially predicted a flat line on or about zero value. It seemed to approximate the average of the target value

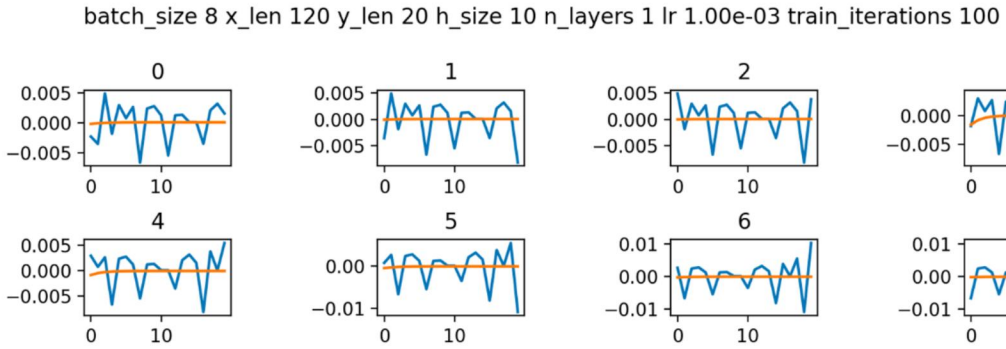

Figure 5 – Sample output

Experiment 1. Figures show almost flat prediction of 20 minute time sequences in 8 samples.

Experiment 2 - Predict a value: This consisted in encoding a sequence as in experiment 1, and trying to predict the cumulative return over a period of 10, 20, and 60 minutes. Results showed that the system predicted positive cumulative returns only 48% of the time. Thus, no trading advantage could be developed from the system as it stands now.

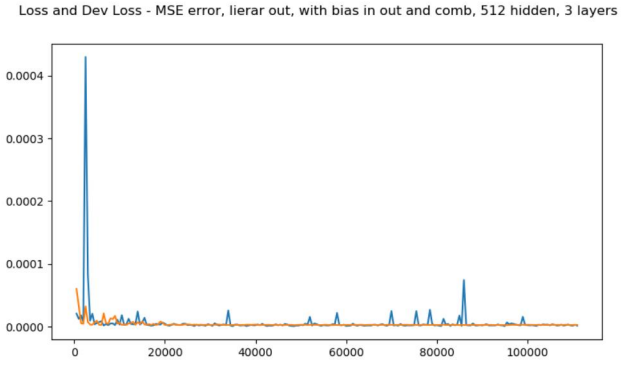

Figure 6 - The train error (blue) and dev error (orange) decreased drastically in the first iterations and then remained relatively static over training epochs. The system trained with a decaying learning rates (halving every 5000 – 8000 batches). Smaller learning rates did not smooth the train error curve. Possibly the fact that the training data is extremely noisy could lead to this behavior. Also, perhaps larger batch sizes should be used to avoid peaks.

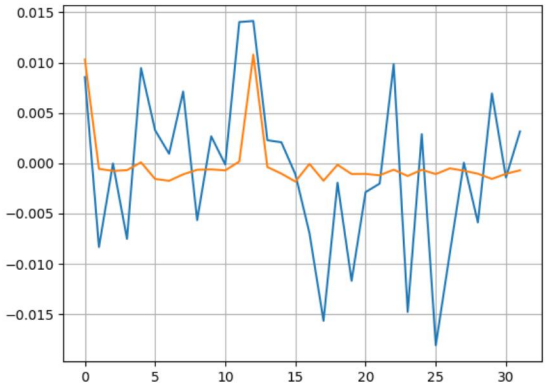

Figure 7 – Prediction vs actual. For 32 separate sequences, the blue line shows the actual cumulative return and the orange line shows the prediction. In this sample we see good performance around minute 0 and minute 13. This was a chance event.

Use

3. Further work

As was mentioned in the introduction, predicting stock returns is a famously complicated aspiration. Nevertheless, it need not be considered impossible. Future tests to better understand the problem could include:

- Using larger training set – maybe 5 - 10 years of data vs just 1
- Developing the prediction on each stock return, not just one
- Use larger batch sized to smooth training errors
- Developing features with a 1 D convolution preprocessor over the feature set
- Performing error analysis to understand performance. The predictive power of the model can vary by day of the week, time of day, and so on