Stock Trading with Recurrent Reinforcement Learning (RRL)

CS229 Application Project

Gabriel Molina, SUID 5055783

I. INTRODUCTION

One relatively new approach to financial trading is to use machine learning algorithms to predict the rise and fall of asset prices before they occur. An optimal trader would buy an asset before the price rises, and sell the asset before its value declines.

For this project, an asset trader will be implemented using recurrent reinforcement learning (RRL). The algorithm and its parameters are from a paper written by Moody and Saffell[1]. It is a gradient ascent algorithm which attempts to maximize a utility function known as Sharpe's ratio. By choosing an optimal parameter w for the trader, we attempt to take advantage of asset price changes. Test examples of the asset trader's operation, both 'real-world' and contrived, are illustrated in the final section.

III. UTILITY FUNCTION: SHARPE'S RATIO

One commonly used metric in financial engineering is Sharpe's ratio. For a time series of investment returns, Sharpe's ratio can be calculated as:

$$S_T = \frac{Average(R_t)}{Standard\ Deviation(R_t)} \quad \text{for interval} \quad t = 1, ..., \mathrm{T}$$

where R_t is the return on investment for trading period t. Intuitively, Sharpe's ratio rewards investment strategies that rely on less volatile trends to make a profit.

IV. TRADER FUNCTION

The trader will attempt to maximize Sharpe's ratio for a given price time series. For this project, the trader function takes the form of a neuron:

$$F_t = \tanh(w^{\mathrm{T}} x_t)$$

where M is the number of time series inputs to the trader, the parameter $w \in \Re^{M+2}$, the input vector $x_t = [1, r_t, ..., r_{t-M}, F_{t-1}]$, and the return $r_t = p_t - p_{t-1}$.

Note that r_t is the difference in value of the asset between the current period t and the previous period. Therefore, r_t is the return on one share of the asset bought at time $t-1$.

Also, the function $F_t \in [-1,1]$ represents the trading position at time t. There are three types of positions that can be held: long, short, or neutral.

A *long* position is when $F_t > 0$. In this case, the trader buys an asset at price p_t and hopes that it appreciates by period $t+1$.

A *short* position is when $F_t < 0$. In this case, the trader sells an asset which it does not own at price p_t, with the expectation to produce the shares at period $t+1$. If the price at $t+1$ is higher, then the trader is forced to buy at the higher $t+1$ price to fulfill the contract. If the price at $t+1$ is lower, then the trader has made a profit.

---

[1] J Moody, M Saffell, Learning to Trade via Direct Reinforcement, *IEEE Transactions on Neural Networks*, Vol 12, No 4, July 2001.

A *neutral* position is when $F_t = 0$. In this case, the outcome at time $t+1$ has no effect on the trader's profits. There will be neither gain nor loss.

Thus, F_t represents holdings at period t. That is, $n_t = \mu \cdot F_t$ shares are bought (long position) or sold (short position), where μ is the maximum possible number of shares per transaction. The return at time t, considering the decision F_{t-1}, is:

$$R_t = \mu \cdot \left(F_{t-1} \cdot r_t - \delta \left| F_t - F_{t-1} \right| \right)$$

where δ is the cost for a transaction at period t. If $F_t = F_{t-1}$ (i.e. no change in our investment this period) then there will be no transaction penalty. Otherwise the penalty is proportional to the difference in shares held.

The first term ($\mu \cdot F_{t-1} \cdot r_t$) is the return resulting from the investment decision from the period $t-1$. For example, if $\mu = 20$ shares, the decision was to buy half the maximum allowed ($F_{t-1} = .5$), and each share increased $r_t = 8$ price units, this term would be 80, the total return profit (ignoring transaction penalties incurred during period t).

V. GRADIENT ASCENT

Maximizing Sharpe's ratio requires a gradient ascent. First, we define our utility function using basic formulas from statistics for mean and variance:

We have

$$S_T = \frac{E[R_t]}{\sqrt{E[R_t^{\,2}] - (E[R_t])^2}} = \frac{A}{\sqrt{B - A^2}} \quad \text{where} \quad A = \frac{1}{T}\sum_{t=1}^{T} R_t \quad \text{and} \quad B = \frac{1}{T}\sum_{t=1}^{T} R_t^{\,2}$$

Then we can take the derivative of S_T using the chain rule:

$$\frac{dS_T}{dw} = \frac{d}{dw}\left\{ \frac{A}{\sqrt{B - A^2}} \right\} = \frac{dS_T}{dA} \cdot \frac{dA}{dw} + \frac{dS_T}{dB} \cdot \frac{dB}{dw}$$

$$= \sum_{t=1}^{T} \left\{ \frac{dS_T}{dA} \cdot \frac{dA}{dR_t} + \frac{dS_T}{dB} \cdot \frac{dB}{dR_t} \right\} \cdot \frac{dR_T}{dw} = \sum_{t=1}^{T} \left\{ \frac{dS_T}{dA}\frac{dA}{dR_t} + \frac{dS_T}{dB}\frac{dB}{dR_t} \right\} \cdot \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{dw} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{dw} \right\}$$

The necessary partial derivatives of the return function are:

$$\frac{dR_t}{dF_t} = \frac{d}{dF_t}\left\{ \mu \cdot \left(F_{t-1} \cdot r_t - \delta \left| F_t - F_{t-1} \right| \right) \right\} = \frac{d}{dF_t}\left\{ -\mu \cdot \delta \cdot \left| F_t - F_{t-1} \right| \right\} = \begin{cases} -\mu \cdot \delta & F_t - F_{t-1} > 0 \\ \mu \cdot \delta & F_t - F_{t-1} < 0 \end{cases}$$

$$= -\mu\delta \cdot \operatorname{sgn}(F_t - F_{t-1})$$

$$\frac{dR_t}{dF_{t-1}} = \frac{d}{dF_{t-1}}\left\{ \mu \cdot \left(F_{t-1} \cdot r_t - \delta \left| F_t - F_{t-1} \right| \right) \right\} = \mu \cdot r_t - \frac{d}{dF_{t-1}}\left\{ -\mu \cdot \delta \cdot \left| F_t - F_{t-1} \right| \right\} = \begin{cases} \mu \cdot \delta & F_t - F_{t-1} > 0 \\ -\mu \cdot \delta & F_t - F_{t-1} < 0 \end{cases}$$

$$= \mu \cdot r_t + \mu\delta \cdot \operatorname{sgn}(F_t - F_{t-1})$$

Then, the partial derivatives dF_t / dw and dF_{t-1} / dw must be calculated:

$$\frac{dF_t}{dw} = \frac{d}{dw}\left\{\tanh(w^T x_t)\right\} = (1-\tanh(w^T x_t)^2)\cdot\frac{d}{dw}\cdot\left\{w^T x_t\right\} = (1-\tanh(w^T x_t)^2)\cdot\left\{x_t + w_{M+2}\frac{dF_{t-1}}{dw}\right\}$$

Note that the derivative dF_t/dw is recurrent and depends on all previous values of dF_t/dw. This means that to train the parameters, we must keep a record of dF_t/dw from the beginning of our time series. Because stock data is in the range of 1000-2000 samples, this slows down the gradient ascent but does not present an insurmountable computational burden. An alternative is to use online learning and to approximate dF_t/dw using only the previous dF_{t-1}/dw term, effectively making the algorithm a stochastic gradient ascent as in Moody & Saffell's paper. However, my chosen approach is to instead use the exact expressions as written above.

Once the dS_T/dw term has been calculated, the weights are updated according to the gradient ascent rule $w_{i+1} = w_i + \rho \cdot dS_T/dw$. The process is repeated for N_e iterations, where N_e is chosen to assure that Sharpe's ratio has converged.

VI. TRAINING

The most successful method in my exploration has been the following algorithm:

1. Train parameters $w \in \Re^{M+2}$ using a historical window of size T
2. Use the optimal policy w to make 'real time' decisions from $t = T+1$ to $t = T + N_{predict}$
3. After $N_{predict}$ predictions are complete, repeat step one.

Intuitively, the stock price has underlying structure that is changing as a function of time. Choosing T large assumes the stock price's structure does not change much during T samples. In the random process example below, T and $N_{predict}$ are large because the structure of the process is constant. If long term trends do not appear to dominate stock behavior, then it makes sense to reduce T, since shorter windows can be a better solution than training on large amounts of past history. For example, data for the years IBM 1980-2006 might not lead to a good strategy for use in Dec. 2006. A more accurate policy would likely result from training with data from 2004-2006.

VII. EXAMPLE

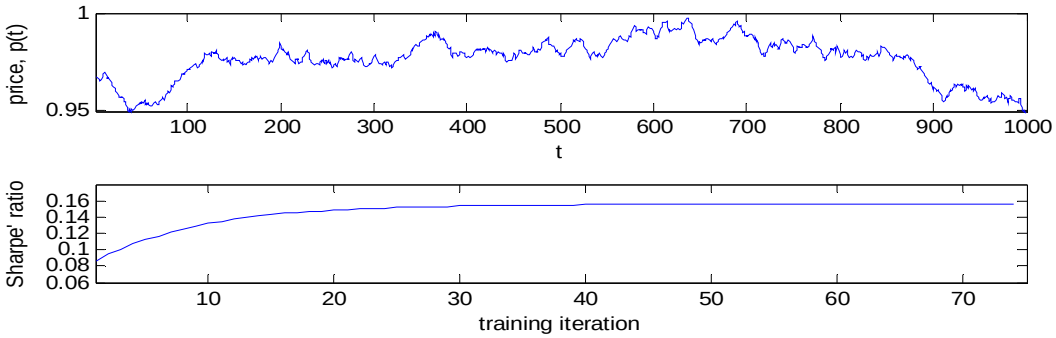

Figure 1. Training results for autoregressive random process. $T = 1000$, $N_e = 75$

The first example of training a policy is executed on an autoregressive random process (randomness by injecting Gaussian noise into coupled equations). In figure 1, the top graph is the generated price series. The bottom graph is Sharpe's ratio on the time series using the parameter w for each iteration of training. So, as training progresses, we find better values of w until we have achieved an optimum Sharpe's ratio for the given data.

Then, we use this optimal w parameter to form a prediction for the next $N_{predict}$ data samples, shown below:

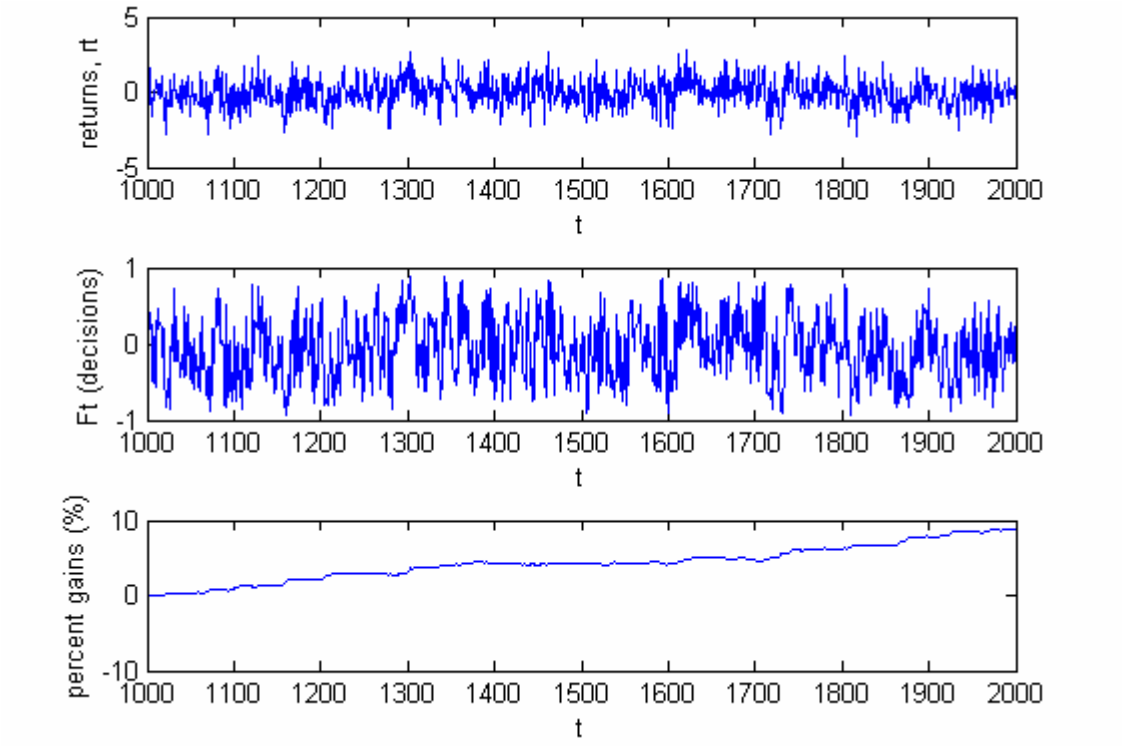

Figure 2. Prediction performance using optimal policy from training. $N_{predict} = 1000$

As is apparent from the above graph, the trader is making decisions based on the w parameter. Of course, w is suboptimal for the time series over this predicted interval, but it does better than a monkey. After 1000 intervals our return would be 10%.

The next experiment, presented in the same format, is to predict real stock data with some precipitous drops (Citigroup):

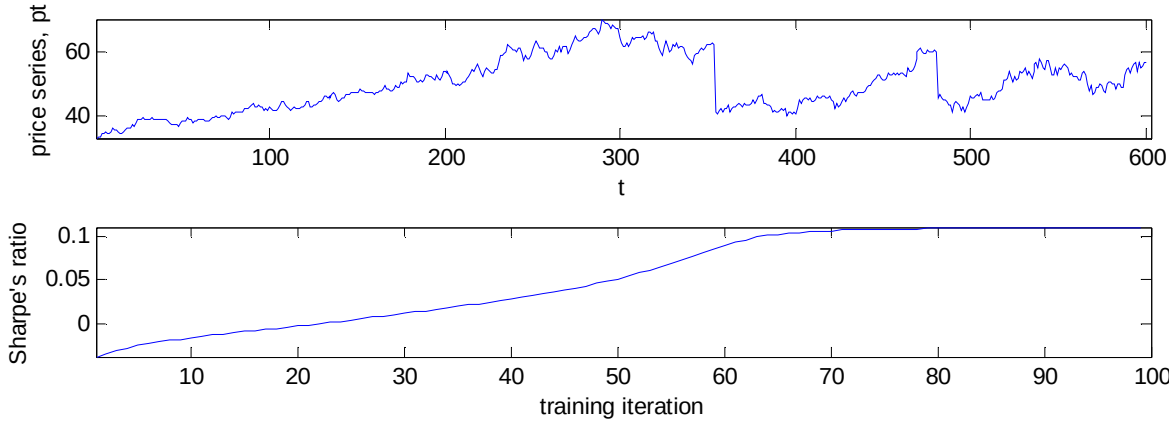

Figure 3. Training w on Citigroup stock data. $T = 600$, $N_e = 100$

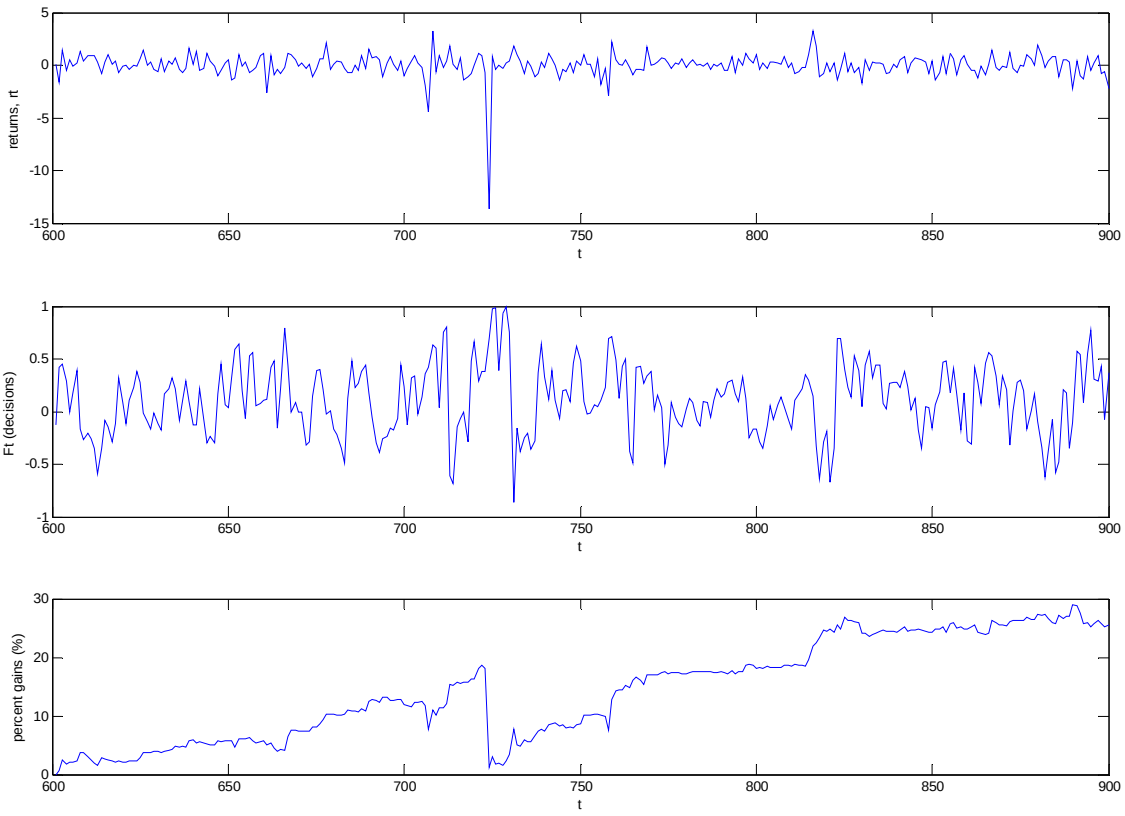

Figure 4. r_t (top), F_t (middle), and percentage profit (cumulative) for Citigroup. Note that although the general policy is good, the precipitous drop in price (downward spike in r_t) wipes out our gains around t = 725.

The recurrent reinforcement learner seems to work best on stocks that are constant on average, yet fluctuate up and down. In such a case, there is less worry about a precipitous drop like in the above example. With a relatively constant mean stock price, the reinforcement learner is free to play the ups and downs.

The recurrent reinforcement learner seems to work, although it is tricky to set up and verify. One important trick is to properly scale the return series data to mean zero and variance one[2], or the neuron cannot separate the resulting data points.

VII. CONCLUSIONS

The primary difficulties with this approach rest in the fact that certain stock events do not exhibit structure. As seen in the second example above, the reinforcement learner does not predict precipitous drops in the stock price and is just as vulnerable as a human. Perhaps it would be more effective if combined with a mechanism to predict such precipitous drops. Other changes to the model might be including stock volumes as features that could help in predicting rises and falls.

Additionally, it would be nice to augment the model to incorporate fixed transaction costs, as well as less frequent transactions. For example, a model could be created that learns from long periods of data, but only periodically makes a decision. This would reflect the case of a casual trader that participates in smaller volume trades with fixed transaction costs. Because it is too expensive for small-time investors to trade every period with fixed transaction costs, a model with a periodic trade strategy would more financially feasible for such users. It would probably be worthwhile to try adapting this model to this sort of periodic trading and see the results.

---

[2] Gold, Carl, FX Trading via Recurrent Reinforcement Learning, *Computational Intelligences for Financial Engineering, 2003. Proceedings. 2003 IEEE International Conference on.* p. 363-370. March 2003. Special thanks to Carl for email advice on algorithm implementation.