Deep Reinforcement Learning for Pairs Trading

Ted Hwang, Samuel Norris, Hang Su, Zhaoming Wu, Yiding Zhao*

I. Introduction and Motivation

Reinforcement learning (RL) [1] differs from traditional supervised machine learning in the sense that it not only considers short-term consequences of actions/decisions, but also long-term outcomes. Because of recent advances in deep learning, model-free deep reinforcement learning (DRL) has proven successful in various applications, as with the success of a deep Q-network (DQN) in the Atari game [2].

A common application of RL is stock trading, as the ultimate goal is to make long-term profit while accounting for the fact that current profits are valued more than future ones. We applied DRL in stock markets to train a pairs trading agent with the goal of maximizing long-term income, albeit possibly at the expense of short-term gain. Briefly, pairs trading is an investment strategy that analyzes pairs of stocks that have a common trend and involves making investment decisions when the stock prices diverge, on the assumption that they will converge back to the trend in the near future [3].

The motivation for our work is clear, as a well-performing investment model will be very lucrative for both corporate and individual investors.

II. Problem Definition

Pairs trading is a market-neutral strategy involving the trading a pair of highly correlated stocks in unison by matching a long position with a short position in the pair [4], [5]. The purpose of this work was to build a deep reinforcement learning model that would optimize decisions regarding making, holding on to, and closing investments in a pairs trading strategy, with the goal of maximizing long-term profit.

* In alphabetical order of last name

III. Background and Related Works

There is a long history of utilizing reinforcement learning techniques in algorithmic trading domain, [6], [7], [8] representing some of the first attempts to build a trading systems that optimizes financial objective functions (i.e., profits) via reinforcement learning. [8] adopts Q-learning to approximate discounted future rewards, and extensive simulations have demonstrated its advantages at profit earning compared to supervised models. [7] tackled the problem of trade execution by considering variables pertaining to the investor's strategy and situation as well as the overall market in order to develop state-based strategies, and it resulted in large performance gain in limit order markets.

Recent studies in deep learning have made model-free reinforcement learning using deep models successful in various applications, such as the success of a deep Q-network in the Atari game [2]. There have been studies proposing novel approaches to apply reinforcement learning to algorithmic trading problems, including [9], which uses least-squares temporal difference to perform generalized policy iteration, and [10], which incorporates a convolutional neural network trained with a variant of Q-learning into policy learning.

[11], [12] demonstrated that PG can outperform Q-learning with proper training, especially in cases with stochastic policies and continuous action spaces. [13] proposed a method to solve deep memory partially observable Markov decision problems (POMDPs) using recurrent PGs. They used recurrent neural networks' (RNN) hidden state to track the partially observed state and train a PG model, which is useful to us because a stock market is often a partially observable setting (do not necessarily know others' actions). Deep models are very complex, and explaining agent decisions through visualization is difficult. [14] is an attempt to visualize the RNN model for discrete

input, although it would be difficult to apply to continuous stock price input (discretizing will lead to curse of dimensions). Attention mechanisms like [15] are worth exploring.

Other algorithmic trading studies use neural networks together with evolutionary algorithms for feature selection and prediction of time series [16]. [17] uses genetic algorithms to predict bankruptcy, which may be useful in other applications in our project. [18] proposed a genetic algorithm based model to tackle the dynamic characteristics of stock data and generate a robust model for pairs trading. [19] employed RL in pairs trading to predict parameters-update time window, trading window, trading threshold, and stop-loss after identifying pairs using cointegration test.

We also consulted the following general sources in our research, primarily to better understand pairs trading [20], [21], [22].

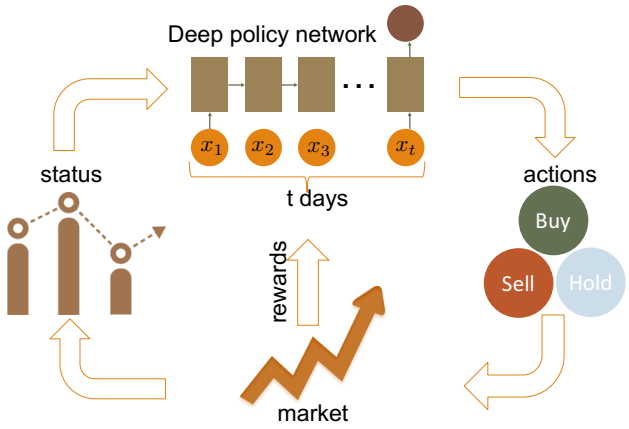

Fig. 1. **The action-reward cycle**. An action (buy, sell, hold) generated from policy network by observing the market status will be executed in market. Market will return the corresponding immediate reward (i.e., the amount of money gained or lost) after the action. The RNN-based deep policy network will be updated according to status, actions, and rewards.

IV. Proposed RL Approach

In this project, we developed a trading agent to maximize long-term profits in the stock market. Our trading agent took in features related to market conditions [3], the relationship of the stocks in a pair, and the agent's current financial position. We used PG, a reinforcement learning approach that directly outputs an action based on the current state, to iteratively train the agent. Over time, the agent learned how to optimally make investment decisions - to buy, to sell or to hold for the two stocks in the pair. We visualized the portfolio value of our trained model in an interactive manner[1].

A. Intuition

Most of the state-of-the-art attempts to train a trading agent via reinforcement learning adopt Q-learning to approximate discounted future rewards. While Q-learning proves successful in many applications with discrete actions, its ability to handle continuous action spaces (i.e., to decide the volume of shares to buy or sell in the market) with stochastic policies is extremely limited. We used PG, a policy-based reinforcement learning framework with the potential to output an action in a continuous space, to train our trading agent, as it affords us the opportunity to maximize the expected rewards.

[1]Live demo at `https://busterbaram.shinyapps.io/project/`

B. Algorithm

In this section, we briefly summarize what RL and PG are. We then describe recurrent RL and how it can be applied in a stock market setting.

The RL model consists of a few components: possible states of the environment $\mathcal{S}$, actions $\mathcal{A}$ an agent can take, a reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a policy function π for the agent. At each time stamp t, the agent takes action a_t, the state of environment will change to s_{t+1} from s_t, and the agent will receive reward r_{t+1}. In a stock market setting, the environment is all the market participants and their status. Clearly, a trading agent will have no way to know the s_t since it only knows what it holds and cannot know what others hold. Possible actions in a stock market setting are *buy*, *sell* and *hold*. The reward can be daily return. The ultimate goal is to maximize long-term accumulated returns $\sum_{k=0}^{\inf} \gamma^k r_k$ where γ is a discount factor (present value is preferred to equivalent future value). Since the environment is partially observed, the policy function, which determines what to do, is stochastic. For example, output of π maybe *buy* with probability 0.8, *sell* 0.1 and *hold* 0.1. We will need to sample an action according to such probability.

PG directly learns π through gradient ascent. Let Θ denote parameter of π to learn and $H =< o_0, a_0, o_1, a_1, \dots o_t >$ denote the history of observed states (tends to be different from s). The goal is to maximize $J = \int_H p(H|\Theta)R(H)dH$, where $R(H)$ is the long-term accumulated reward and $p(H|\Theta)$ is the probability of H conditioned on θ.

Since the state of the environment (e.g., stock prices) is a time series, we use recurrent neural network (RNN) to fit the policy function π. The input to RNN is a series of historical features such as observations, market status, and market indicators, which are called features. The output of policy function is a probability distribution of actions.

Inspired by [23], we use the following features:

- Spread (difference between) of prices of the stocks in the pair
- S&P500 index as a proxy for the state of the market and economy
- VIX index for market volatility
- Effective Federal Funds Rate for its general influence on investments
- Value of the agent's portfolio

Stock prices and all other features are implemented as percentage change from their value at the start of the time period in question.

Instead of back propagating through time, we use the policy log gradient in Eq 5 to update Θ. Figure IV depicts the action-reward cycle in our model. It starts with an RNN-based policy network generating a probability distribution of actions, from which an action (to buy, sell, or hold) is sampled. Then, the action is executed in the market, which leads to an immediate reward, that is, the amount of money lost or gained on the next trading day. The policy network will be updated periodically according to the accumulated history H.

We run the update in a batch manner. For example, given $H =< o_0, a_0, r_0, o_1, a_1, r_1, \dots o_n >$, we update the parameters Θ in the policy network using gradient ascent on J by fetching the last m batches of (observation, action, reward) tuple lists, each of size k, i.e., $B_i = \{(o_{n-i*k+j}, a_{n-i*k+j}, r_{n-i*k+j})\}_{j=0}^{k}$. All m batches will be exploited to conduct gradient ascent via first-order optimization algorithms such as stochastic gradient ascent or AdaGrad. The rewards in each batch a are normalized for a larger rate of convergence. The gradient is calculated as

$$\nabla_\Theta J = \nabla_\Theta \int_H p(H|\Theta)R(H)dH \tag{1}$$

$$= \int_H \nabla_\Theta p(H|\Theta)R(H)dH \tag{2}$$

$$= \int_H \frac{p(H|\Theta)}{p(H|\Theta)} \nabla_\Theta p(H|\Theta)R(H)dH \tag{3}$$

$$= \int_H R(H)\nabla_\Theta \log p(H|\Theta)dH \tag{4}$$

$$\approx \frac{1}{m}\sum_{i=1}^{m} R(H_i)\nabla_\Theta \log p(H_i|\Theta) \tag{5}$$

This equation dictates that we need to run a trading agent m times and get $H_1, H_2, \dots, H_m$ to calculate gradient, which is the average of $\log p(H|\Theta)$. In practice, we can also update the model in a stochastic manner (i.e., directly update Θ with proper step-size using just one history episode H_i). The gradient $R(H_i)\nabla_\Theta \log p(H_i|\Theta)$ can be further broken down into $\sum_{i=0}^{T} r_i \nabla_\Theta \log \pi(a_i|\Theta)$ (see [13] for more detailed derivations). In this way, our model is updated in an online-learning fashion; that is, data comes in a sequential order and the model keeps evolving as new data arrives.

V. Experiment

In this section, we quantitatively analyze the performance of our model by showing experiment results and describe how we visualize the results in an interactive web interface.

A. Setup

The RNN-based policy gradient trading agent was implemented in Python with Theano [24] and Lasagne, a wrapper of Theano. The specific RNN models we implemented inclued bi-directional RNNs, long short-term memory networks (LSTM), and bi-directional LSTM. The length of an episode was set to 200; that is, every decision made by the agent was based on the market status of the past 200 trading days. The number of hidden layer units was set to 20. Proper feature normalization was conducted to avoid gradient

vanishing or explosion. The market simulator is also implemented in Python. Since it is extremely hard to tune deep learning models without GPUs, we experimented on a powerful work station with Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz CPU, 256GB memory and 4 NVIDIA GeForce GTX TITAN X GPU cards.

We selected annual returns as the evaluation metric, defined as

$$\frac{PV(y) - PV(y-1)}{PV(y-1)} - 1$$

, where $PV(y)$ is the portfolio value at the end of year y.

We compared our method, deep reinforcement learning (**DRL**), with two baselines

1) **Standard Investment (SI)**: the performance of a portfolio by investing all cash to S&P500 and holding that position.
2) **Random agent (RAND)**: an agent making random decisions of buying, selling, and holding.

All methods start with $100,000 cash. We assume that there are no transaction costs and do not restrict leverage ratio (i.e. we can take infinite loans). For experimental purposes, we assume agents can purchase stock using adjusted close price on a given date.

B. Data

Based on work by [3] that builds on [25], we decided to analyze stocks from the S&P 500 in the utilities sector. Utilities stocks have performed well in this past work on pairs trading, and the utilities industry should, by nature, be stable [3]. We pull our data from Yahoo! Finance[2], the Federal Reserve[3], and (for the list of S&P 500 Utilities stocks) Wikipedia[4]. We collected daily adjusted closing price data for 4 December 2001 through 30 November 2016. All pairs of stocks from the utilities sector (that had data going back far enough) were put through a Python script we wrote to identify those pairs that are most cointegrated. The pair, SCANA Corporation (NYSE: **SCG**) and WEC Energy Group Inc (NYSE: **WEC**), were most cointegrated over the approximately 15 year period of the entire dataset, and they were selected for use in our reinforcement learning algorithm.

[2]https://finance.yahoo.com/

[3]https://fred.stlouisfed.org/series/DFF

[4]https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

C. Results

The model was trained and tested on our data. Table I shows the starting portfolio value (SV), ending portfolio value (EV) and annual return rate (RR) over time for the proposed method and both baselines. Although our data start from 2001, the agent starts making decision in 2002, as it needs data from a look back period of 200 trading days.

Since RAND consistently performed worse than the standard index investment, we will focus more on the comparison between SI and DRL. The DRL agent lost money in the first two years (2002 and 2003) as the model is, by nature, weak at the start and requires time to tune its parameters to fit the market. Starting from the year 2004, DRL had a positive return rate, though it failed to beat the SI baseline for some years. An interesting phenomenon we can observe from the table is that in 2008, the year of financial crisis, DRL achieved a very high return rate 23.09%, while the market index S&P 500 based trading strategy leads to a huge loss of -36.24%. In 2007 however, the DRL agent lost a considerable amount (-15.43%). Financial crisis can lead to a structural shift in the market, which may mean that DRL has to re-learn from experience. This could be the reason that, for the subsequent several years, DRL failed to beat SI. Finally in 2015 and 2016, the DRL agent accumulated a lot experience and dramatically outperformed both baselines. Especially in the year 2015, both SI and RAND return rates are very low (1.29% and 2.34%, respectively), while the DRL agent achieves a 20.02% return. Similarly, in 2016, DRL yielded a much higher return than both baselines.

The staring value of $100,000 in 2002 and ending value of $\$345,030$ in 2016 for SI means by purchasing $100,000 worth stocks of S&P 500 in 2002, one can make $\$345,030$ at the end of November 2016. It's worth pointing out that the composition of the S&P 500 is not fixed. Thus the SI baseline is biased toward high performing stocks ("better" stocks replace weaker ones)[26].

Even with SI being biased, DRL achieved comparable final ending value by 2016.

D. Interactive visualization

We developed a quantitative analysis web platform using R Shiny to visualize the agent's portfolio value and investment actions over time. It is based on previous work by a group member, but it includes a new visualization tailored to our work. The bottom line chart shows portfolio value and includes vertical marks to denote investment actions. The chart is fully interactive. Existing visualizations on the dashboard have been linked to our data, so both new and existing visualization provide an immersive and informative environment for the user. This visualization can help users check validity of our model. Additionally, it may be very beneficial to those investors who may not fully understand the intricacies of our machine learning approach. A live demo of the visualization (with results from an earlier version of our experiments) is available at online.

VI. Conclusions and Discussion

In this work, we

- designed and implemented a DRL-based agent to make pairs trading decisions;
- utilized a model-free reinforcement learning approach called policy gradient (PG), which we believe has not yet been applied to pairs trading and, due to its ability to handle continuous action spaces, may outperform more limited Q-learning algorithms that are currently in use;
- fitted the policy function with RNN and variations thereof;
- fetched approximately 15 years of stock price data from Yahoo Finance;
- selected the pair (SCG and WEC) for experiments via a cointegration test; and
- trained and tested our agent on the stock price data and calculated annual return rates to validate the effectiveness of the proposed method.

Our work, as a first attempt, demonstrates that besides Q-learning based reinforcement learning, policy-based approaches can also be utilized to build trading agents. Although we chose PG because of its ability to handle continuous action spaces, at this point we only managed to use discrete outputs from the deep policy network, which limits our model to some extent. Moreover, policy-based RL models typically converge to a local rather than global optima, so there is high variance in evaluating the policy action. Future work could include replacing the output function, softmax, in the deep policy network with an appropriate function that outputs continuous actions. We could also reduce variance by using techniques such as Actor-critic[27].

Contribution Distribution

All team members contributed a similar amount of effort in this work.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Binh Do and Robert Faff. Does simple pairs trading still work? *Financial Analysts Journal*, 66(4):83–95, 2010.

[4] Robert J Elliott, John Van Der Hoek*, and William P Malcolm. Pairs trading. *Quantitative Finance*, 5(3):271–276, 2005.

[5] Advanced Financial Data Analysis-Patrick Mc Sharry Assignment Module VII. Efficient pair selection for Pair-Trading strategies.

[6] John Moody, Matthew Saffell, Yuansong Liao, and Lizhong Wu. Reinforcement learning for trading systems and portfolios: Immediate vs future rewards. In *Decision Technologies for Computational Finance*, pages 129–140. Springer, 1998.

[7] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*, pages 673–680. ACM, 2006.

[8] John E Moody and Matthew Saffell. Reinforcement learning for trading. In M J Kearns, S A Solla, and D A Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 917–923. MIT Press, 1999.

[9] James Cumming, Dalal Alrajeh, and Luke Dickens. An investigation into the use of reinforcement learning techniques within the algorithmic trading domain. 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*, 2016.

TABLE I

MODEL PERFORMANCE OVERTIME COMPARE TO BASELINES. HIGHEST RETURN RATE IN EACH YEAR IS IN BOLD.

	SI			DRL			RAND		
Year	SV	EV	RR(%)	SV	EV	RR(%)	SV	EV	RR(%)
2002	100,000	105,114	**5.11**	100,000	99,043	-0.96	100,000	99,141	-0.86
2003	108,497	134,737	**24.18**	98,967	98,706	-0.26	99,141	99,285	0.14
2004	134,676	149,151	**10.75**	98,698	99,354	0.66	99,268	97,971	-1.31
2005	148,447	156,352	**5.32**	99,392	101,701	2.32	98,041	99,304	1.29
2006	159,102	181,126	13.84	101,165	115,418	**14.09**	99,036	101,645	2.63
2007	180,807	190,448	**5.33**	111,243	94,080	-15.43	101,049	99,156	-1.87
2008	188,780	120,372	-36.24	96,185	118,396	**23.09**	99,293	100,416	1.13
2009	124,001	152,093	22.65	110,301	137,117	**24.31**	100,465	100,187	-0.28
2010	154,672	174,992	**13.14**	137,689	149,134	8.31	100,184	99,601	-0.58
2011	176,801	178,308	0.85	146,592	156,456	**6.73**	99,708	99,514	-0.19
2012	181,150	206,820	**14.17**	158,597	163,097	2.84	99,367	98,718	-0.65
2013	212,121	273,639	**29.00**	160,080	180,703	12.88	98,867	98,537	-0.33
2014	271,017	310,481	**14.56**	182,041	182,134	0.05	98,371	101,486	3.17
2015	310,315	314,313	1.29	182,049	218,500	**20.02**	101,444	103,814	2.34
2016	309,920	345,030	11.33	220,400	317,952	**44.26**	104,079	110,571	6.24

[12] Xin Du, Jinjian Zhai, and Koupin Lv. Algorithm trading using q-learning and recurrent reinforcement learning. *positions*, 1:1.

[13] Daan Wierstra, Alexander Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory pomdps with recurrent policy gradients. In *International Conference on Artificial Neural Networks*, pages 697–706. Springer, 2007.

[14] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[16] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2):125–132, 2000.

[17] Kyung-Shik Shin and Yong-Joo Lee. A genetic algorithm application in bankruptcy prediction modeling. *Expert Systems with Applications*, 23(3):321–328, 2002.

[18] Xiangzhou Zhang, Yong Hu, Kang Xie, Shouyang Wang, EWT Ngai, and Mei Liu. A causal feature selection algorithm for stock prediction modeling. *Neurocomputing*, 142:48–59, 2014.

[19] Saeid Fallahpour, Hasan Hakimian, Khalil Taheri, and Ehsan Ramezanifar. Pairs trading strategy optimization using the reinforcement learning method: A cointegration approach. *Available at SSRN 2624328*, 2015.

[20] Lex Dam. Why i wont teach pair trading to my students, 2016.

[21] Jiva Mitchell, CoryKalan. How to use a pairs trading strategy with etfs, 2016.

[22] statistical arbitrage correlation vs cointegration.

[23] Jean Folger. Choosing indicators to develop a strategy, 2016.

[24] James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, David Warde-Farley, Ian Goodfellow, Arnaud Bergeron, et al. Theano: Deep learning on gpus with python. In *NIPS 2011, BigLearning Workshop, Granada, Spain*. Citeseer, 2011.

[25] Evan Gatev, William N Goetzmann, and K Geert Rouwenhorst. Pairs trading: Performance of a Relative-Value arbitrage rule. *Rev. Financ. Stud.*, 19(3):797–827, 21 September 2006.

[26] Adam Hayes. Top reasons stock indices could be biased, 2016.

[27] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.