Deep Reinforcement Learning for Pairs Trading Using Actor-critic

Yichen Shen, Yiding Zhao*

Abstract— **Partially observed Markov decision process problem of pairs trading is a challenging aspect in algorithmic trading. In this work, we tackle this by utilizing a deep reinforcement learning algorithm called advantage actor-critic by extending the policy network with a critic network, to incorporate both the stochastic policy gradient and value gradient. We have also used recurrent neural network coupled with long-short term memory to preserve information from time series data of stock market. A memory buffer for experience replay and a target network are also employed to reduce the variance from noisy and correlated environment. Our results demonstrate a success on learning a well-performing lucrative model by directly taking data from public available sources and present possibilities for extensions to other time-sensitive applications.**

I. Introduction

Reinforcement learning (RL) [1] differs from traditional supervised machine learning in the sense that it not only considers short-term consequences of decisions, but also long-term outcomes. A natural 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. Because of recent advances in deep learning (DL), 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 which incorporates a convolutional neural network (CNN) trained with a variant of Q-Learning into policy learning [2].

In stock trading field, pairs trading [3] is a typical market-neutral strategy involving the trading a pair of highly positive correlated stocks in unison by matching a long position with a short position in the pair when the stock prices diverge, based on the assumption that they will converge back in the near future. Pairs trading strategy has explored the statistical arbitrage raised from the short-term fluctuations in the prices of two correlated assets [4]. It utilized the fact that the spread, which is calculated by the discrepancy of a pair of selected stocks, forms a mean-reverting stationary process so any deviations from the mean indicate a possible trading opportunity. The strategy thus makes a long position on the stock which is overpriced, in the mean time sells short another stock which is underpriced. It closes the concluded position by the time they converge and reverts to opposite positions.

The purpose of this work is to build a DRL model that would optimize decisions regarding making, holding on to, and closing investments in a pairs trading setting, with the goal of maximizing long-term profit, albeit possibly at the expense of short-term gain. A well-performing investment model will be very lucrative for both corporate and individual investors.

II. Background and Related Works

There is a long tradition of utilizing RL techniques in algorithmic trading domain, [5], [6], [7] representing some of the first attempts to build a trading systems that optimizes financial objective functions via RL. Extensive simulations have demonstrated its advantages at profit earning compared to supervised models, such as [7] which adopts Q-Learning to approximate discounted future rewards. [6] 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. [8] employed RL in pairs trading to predict parameters-update time window,

* In alphabetical order of last name

trading window, trading threshold, and stop-loss after identifying pairs using cointegration test.

A stock market is often a partially observable process because of the restricted availability of trading data and lack of knowledge to other traders' actions. And it has the complexity of a huge state space and randomness of the policy. There are studies proposing methods to solve deep memory partially observable Markov decision problems (POMDPs). [9] used recurrent neural networks' (RNN) hidden state to track the partially observed state and train a Policy Gradient (PG) model. [10], [11] also demonstrated that PG can outperform Q-Learning with proper training, especially in cases with stochastic policies and continuous action spaces. To apply DRL to algorithmic trading problems, [12] uses least-squares temporal difference (LSTD) to perform generalized policy iteration. However, no previous work has investigated the usage of DRL with respect to pairs trading strategy.

The pairs trading setting in this work is based on the cointegration test results. A brief discussion can be found on [13] for the idea of using cointegration to determine the lockstep characteristics of the pair which will yield a high profit margin. Let $\{y_t\}$ and $\{x_t\}$ be the prices of stock y and x, if for a certain value γ where the series $y_t - \gamma x_t$ is following $I(0)$ stationary, then the two series is said to be cointegrated. This is based on the first term from the right-hand side as in

$$y_t - y_{t-1} = \alpha_y(y_{t-1} - \gamma x_{t-1}) + \epsilon_{y_t} \qquad (1)$$

which states that the cointegration and error-correction are equivalent representations. The ϵ and α are the corresponding white noise and error-correction rate.

III. Proposed DRL Approach

In this work, we developed a trading agent to maximize long-term profits for a pair of most cointegrated stocks. Our trading agent took in features related to market conditions, the relationship of the stocks in a pair, and the agent's current financial position as suggested in [14]. We used actor-critic method, an RL approach that has two components interacting with each other: a policy network that directly outputs a stochastic policy based on the current state, and a value network that evaluate the current policy. Two networks can learn from each other 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.

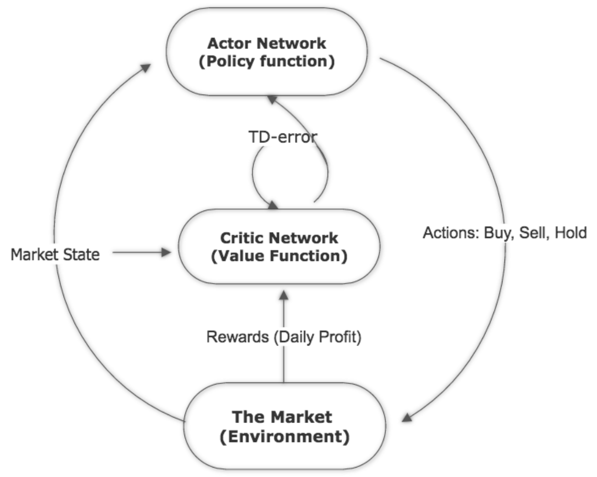

Fig. 1. **The action-state-reward cycle for actor-critic framework**. An action (buy, sell, hold) generated from policy network (actor) by observing market status will be executed in market. Market will return the corresponding immediate reward (i.e., the amount of money gained or lost) after the action. The TD-error calculated from reward will serve as feedback to update value function (critic). The TD-error as Critic's estimation of advantage for a current policy will serve as signal to update the Actor's network.

A. *Intuition*

Most of the state-of-the-art attempts to train a trading agent via RL adopt Q-Learning to approximate discounted future rewards. While Q-Learning proves successful in many applications with discrete actions, its ability to handle stochastic policies is extremely limited. Due to the existence of randomness in financial markets, the optimal trading policy under certain environment can be stochastic. While policy-based approaches can handle stochastic policies, traditional policy gradient methods using Monte-Carlo method have high variances. Thus, we chose actor-critic method, a variation of policy-based RL framework with the potential to output an stochastic optimal policy, to train our trading agent, as it affords us the opportunity to maximize the expected rewards effectively.

B. Algorithm

In this section, we briefly introduce RL and actor-critic. 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 $\mathcal{S}$, action space $\mathcal{A}$, a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ for the agent to choose action and a value function $V : \mathcal{S} \rightarrow \mathbb{R}$ under policy π. At each time stamp t, a trading agent will have to approximate the s_t using public available data, since it cannot have the scope of the entire market. 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). The policy function (actor), 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.

Actor network approximates policy function and learns $\pi(a|s, \theta_p)$ through gradient ascent, where θ_p denote parameter of π. Since the goal is trying to maximize long term reward of each step, which can be expressed as $\mathbb{E}[R_t]$. Standard REINFORCE algorithm introduced by [15] updates θ in the direction of $\nabla_\theta \log \pi(a_t|s_t; \theta) R_t$, which is an unbiased estimator of $\nabla_\theta \mathbb{E}[R_t]$, where R_t can be estimated using Monte-Carlo method. However, updating gradient by direct sampling of R_t usually introduces high variance. It's possible to reduce the variance by subtracting R_t by a baseline term $b_t s_t$[15]. In practice, $b_t s_t$ is commonly approximated by $V^\pi(s_t)$. The gradient to update then becomes

$$\nabla_\theta \log \pi(a_t|s_t; \theta)(R_t - V^\pi(s_t)) \tag{2}$$

When an approximate value function is used as the baseline, $R_t - V^\pi$ can be seen as an estimate of advantage function $A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$. Let r_t be the immediate reward at time t, $Q^\pi(a_t, s_t)$ is just $\mathbb{E}[r_t + \gamma Q^\pi(S_{t+1}, a_{t+1})]$. One unbiased estimator for the advantage function is one-step TD-error $A^\pi(a_t, s_t) = r_t + \gamma V^\pi(s_t) - V^\pi(s_{t+1})$. We substitute this into equation (1) and the subsequent gradient for actor network becomes

$$\nabla_\theta \log \pi(a_t|s_t; \theta) A^\pi(a_t, s_t) \tag{3}$$

$$= \nabla_\theta \log \pi(a_t|s_t; \theta)(r_t + \gamma V^\pi(s_t) - V^\pi(s_{t+1})) \tag{4}$$

Following the architecture of actor-critic, we use a critic network to estimate $V^\pi(s_t)$, which in our case is a separate neural network that has a similar structure as actor network. The objective for critic network is to minimize the mean-squared-error between estimated value of the state and target value of that state under policy π. We estimate the error using one-step TD error. Let θ_v be the parameters of critic network, the objective can be written as $\min_\theta \mathbb{E}[(r_t + \gamma V^\pi(S_{t+1}; \theta_v) - V^\pi(S_t; \theta_v))^2]$. To stabilize the critic network, we use similar approaches in Deep Q-Learning Networks from [2]. Specifically, we use a target network to calculate target value $V^\pi(S_{t+1})$. The online gradient update at each time stamp t for critic network is

$$(r_t + \gamma V^\pi(S_{t+1}; \theta_v^-) - V^\pi(S_t; \theta_v)) \nabla_{\theta_v} V^\pi(S_t; \theta_v) \tag{5}$$

,where θ_v^- is the parameters for target network.

Since the state of the environment (e.g., stock prices) is a time series, we use two recurrent neural networks (RNN) to approximate the policy function (actor) and the value function under the policy (critic). Each state s_t as the input to RNN is a series of historical features such as observations, market status, and market indicators. The output of policy function is a probability distribution of actions.

Inspired by [16], 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. Even with these relatively simply feature, we got encouraging result (See more details in next section).

Our trading algorithm starts with an RNN-based policy network generating a probability distribution of actions, from which an action (e.g. to buy, sell, or hold) is sampled. Then, the action is executed in the market, which leads to an immediate reward r_t, that is, the amount of money lost or gained on the next trading day. The actor network and the critic network will be updated periodically based on their gradient, specified in Eq. (3) (4).

We run the update in a batch manner. Specifically, we used a technique called experience replay[17], where a batch of groups of state, action and reward is randomly sampled from a replay memory. A replay memory saves recent states, actions and rewards up to some threshold. This technique can reduce the correlation of samples from batch and thus stabilize the training process. 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 are normalized for a larger rate of convergence. Since we want our model to be used as in a real portfolio management system, our model is updated in online-learning fashion and goes on forever; that is, data comes in a sequential order and the model keeps evolving as new data arrives. See algorithm (1) below.

Algorithm 1: Actor-Critic Pair-Trading Algorithm

```
1  initialize critic and actor network $V_{\theta_v}(s_t)$ and
     $\pi_\theta(a_t|s_t)$;
2  initialize target network $V_{\theta_v^-}(s_t)$ for critic
     with weights $\theta_v^- \Leftarrow \theta_v$;
3  $n = 0$;
4  initialize replay memory;
5  while $t <$ Today do
6      get new state $s_t$;
7      actor picks an action $a_t \sim \pi_\theta(a_t|s_t)$ and
         get reward $r_t$;
8      append $(s_t, a_t, r_t)$ to memory;
9      if $n$ mod updateCycle = 0 then
10         Sample a mini batch of samples (s, a,
             r) from replay buffer;
11         Compute TD target for each sample
             as $r_t + \gamma V_{\theta_v^-}(s_{t+1})$;
12         Compute critic gradient using Eq.(4);
13         Compute actor gradient using Eq.(3);
14         Update critic using AdaGrad;
15         Update actor using AdaGrad;
16         Update target network $\theta_v^- \Leftarrow \theta_v$;
17     update replay memory to keep most
         recent $Z$ states;
18     $n = n + 1$
```

IV. 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 Actor-critic trading agent was implemented in Python with Theano [18], Lasagne and Keras (both wrappers of Theano). Although their weights are updated independently, Actor network and critic network use the same RNN architecture, except for the last layer, where actor has a softmax output and critic has a linear output. The specific RNN models we implemented include bi-directional RNNs, long short-term memory networks (LSTM), and bi-directional LSTM. The length of an look-back period was set to 50; that is, every decision made by the agent was based on the market status of the past 50 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. We experimented the agent on an Azure GPU instance with NVIDIA Tesla X80 GPU card.

In our experiment, the trading agent will execute actions in a simplified pairs trading setting with a fixed factor of prices ratio γ, under the assumption that the agent is able to make the best decisions over time, even without the knowledge of best combination of two instruments. Let A, B be the pair of stocks we selected in our portfolio, There are three discrete actions that the agent can carry out on each individual day: buy 100 A shares and sell 100 B shares (Buy), sell 100 A shares and buy 100 B shares (Sell) or maintain portfolio structure from the previous day (Hold). The agent can only perform exactly one action per day.

We selected annual returns as the evaluation metric, defined as

$$\frac{\mathbf{PV}(y) - \mathbf{PV}(y-1)}{\mathbf{PV}(y-1)} - 1 \tag{6}$$

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

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

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

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

B. Data

Based on work by [14] that builds on [4], 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 [14]. We pull our data from Yahoo! Finance[1], the Federal Reserve[2], and (for the list of S&P 500 Utilities stocks) Wikipedia[3]. We collected daily adjusted closing price data for January 1, 2002 through December 30, 2016. All pairs of stocks from the utilities sector (that had data going back far enough) were put through a test 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 hence were selected for use in our RL algorithm.

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 near 2002, as it needs data from a look back period of 50 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 didn't beat baseline in the first few years as the model is, by nature, weak at the start and requires time to tune its parameters to fit the market. An interesting phenomenon we can observe from the table is that in 2008, the year of financial crisis, DRL achieved a positive return rate 5.02%, while the market index S&P 500 based trading strategy leads to a huge loss of -36.21%. The DRL agent didn't perform as well as baseline in a few years after 2008. 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 2014, 2015 and 2016, the DRL agent accumulated a lot experience and dramatically out-performed 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 71.07% return. In 2016, DRL yielded an even higher return of 103.58%, outperforming both baselines by a large margin. Overall the results, the DRL agent needs some time to "learn" the best action in different market situations. When DRL has learned how to interact with market dynamics, it can achieve very high performance. The starting value of \$100,000 in 2002 and ending value of $\$461,413$ in 2016 for SI means by purchasing \$100,000 worth stocks of S&P 500 in 2002, one can make $\$259,816$ at the end of November 2016. It's worth pointing out that the composition of the S&P 500 and the γ for cointegration are not fixed. Thus the SI baseline is biased toward high performing stocks ("better" stocks replace weaker ones) as indicated in [19], while the agent attempts to exploit relying on only the fixed γ. Even with the above facts, DRL achieved better final ending value by 2016. Please see Table I for detailed results in each year.

[1]`https://finance.yahoo.com/`

[2]`https://fred.stlouisfed.org/series/DFF`

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

V. Conclusions and Discussion

In this work, we

- designed and implemented a Actor-Critic agent to make pairs trading decisions;
- utilized a stochastic policy-based RL approach called Actor-critic, which we believe has not yet been applied to pairs trading, may outperform more limited PG/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 most cointegrated pair (SCG and WEC) for experiments via a cointegration test;
- trained and tested our agent on the stock price data and calculated total return rates to validate the effectiveness of the proposed method.

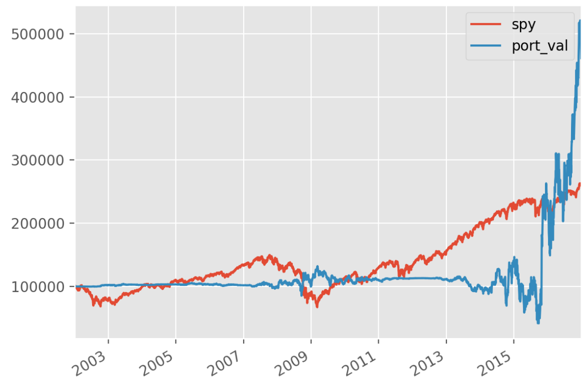

Fig. 2. **The DRI portfolio value against SI portfolio value over time**. Financial crisis can lead to a structural shift in the market, which may mean that DRL has to re-learn from experience. Finally in 2014, 2015 and 2016, the DRL agent accumulated a lot experience and dramatically out-performed both baselines.

Our work demonstrates that besides Q-Learning/Policy Gradient based reinforcement learning, actor-critic approaches can also be utilized to build trading agents. We chose actor-critic framework in our pair-trading algorithm because of its ability to output stochastic policy and it's lower variance than Monte-Carlo based policy-gradient methods. The results show robust performance of RL trading agent using Actor-Critic. However, due to our fairly simplistic set-up, our learned trading strategies is still quite limited. Future work could explore actor-critic approaches with continuous action space as in [20]. Also, future models can utilize intra-day market data to make agent capable of making multiple actions per day.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] Ganapathy Vidyamurthy. *Pairs Trading: quantitative methods and analysis*, volume 217. John Wiley & Sons, 2004.

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

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	77,580	-22.42	100,000	102,078	**2.08**	100,000	99,141	-0.86
2003	800,78	99,444	**24.18**	102,127	102,488	0.35	99,141	99,285	0.14
2004	99,399	110,082	**10.75**	102,530	103,353	0.8	99,268	97,971	-1.31
2005	109,563	115,397	**5.32**	103,276	102,740	-0.52	98,041	99,304	1.29
2006	117,427	133,682	**15.85**	103,020	101,826	-1.16	99,036	101,645	2.63
2007	133,446	140,562	**5.15**	101,790	99,481	-2.27	101,049	99,156	-1.87
2008	139,331	88,842	-36.21	99,944	104,962	**5.02**	99,293	100,416	1.13
2009	91,520	112,254	**22.65**	102,901	107,322	4.30	100,465	100,187	-0.28
2010	114,158	129,155	**13.14**	107,486	110,522	2.82	100,184	99,601	-0.58
2011	130,490	131,602	0.85	110,182	112,859	**2.43**	99,708	99,514	-0.19
2012	133,700	152,646	**15.42**	112,859	111,309	-1.37	99,367	98,718	-0.65
2013	156,558	201,962	**29.00**	111,912	99,856	-10.77	98,867	98,537	-0.33
2014	200,027	229,154	14.56	97,868	135,694	38.65	98,371	101,486	3.17
2015	229,032	231,983	1.29	135,054	231,034	71.07	101,444	103,814	2.34
2016	228,740	259,816	13.59	226,647	461,413	103.58	104,079	110,571	6.24

[15] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

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

[17] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[18] 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.

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

[20] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.