Model-based Deep Reinforcement Learning for Financial Portfolio Optimization

Pengqian Yu [*1] **Joon Sern Lee** [*1] **Ilya Kulyatin** [1] **Zekun Shi** [1] **Sakyasingha Dasgupta**** [1]

Abstract

Financial portfolio optimization is the process of sequentially allocating wealth to a collection of assets (portfolio) during consecutive trading periods, based on investors' risk-return profile. Automating this process with machine learning remains a challenging problem. Here, we design a deep reinforcement learning (RL) architecture with an autonomous trading agent such that, given a portfolio, weight of assets in the portfolio are updated periodically, based on a global objective. In particular, without relying on a naive application of off the shelf model-free agent, we train our trading agent within a novel model-based RL architecture using an infused prediction module (IPM), and a behavior cloning module (BCM), extending standard actor-critic algorithms. We further design a back-testing and trade execution engine which interact with the RL agent in real time. Using historical *real* financial market data over multiple years, we simulate daily trading with practical constraints, and demonstrate that our proposed model is robust, profitable and risk-sensitive, as compared to baseline trading strategies and model-free RL agents as used in prior work.

1. Introduction

Reinforcement learning (RL) consists of an agent interacting with the environment, in order to learn an optimal policy by trial and error for sequential decision-making problems (Bertsekas, 2005; Sutton & Barto, 2018). The past decade has witnessed the tremendous success of deep reinforcement learning (RL) in the fields of gaming, robotics and recommendation systems (Lillicrap et al., 2015; Silver et al., 2016; Mnih et al., 2015; 2016). However, its applications in the financial domain have not been explored as thoroughly.

Dynamic portfolio optimization remains one of the most challenging problems in the field of finance (Markowitz, 1959; Haugen & Haugen, 1990). It is a sequential decision-making process of continuously reallocating funds into a number of different financial investment products, with the main aim to maximize return while constraining risk. Classical approaches to this problem include dynamic programming and convex optimization, which require discrete actions and thus suffer from the 'curse of dimensionality' (e.g., (Cover, 1991; Li & Hoi, 2014; Feng et al., 2015)).

There have been efforts made to apply RL techniques to alleviate the dimensionality issue in the portfolio optimization problem (Moody & Saffell, 2001; Dempster & Leemans, 2006; Cumming et al., 2015; Jiang et al., 2017; Deng et al., 2017; Guo et al., 2018; Liang et al., 2018). The main idea is to train an RL agent that is rewarded if its investment decisions increase the logarithmic rate of return and is penalised otherwise. However, these RL algorithms have several drawbacks. In particular, the approaches in (Moody & Saffell, 2001; Dempster & Leemans, 2006; Cumming et al., 2015; Deng et al., 2017) only yield discrete single-asset trading signals. The multi-assets setting was studied in (Guo et al., 2018), however, the authors did not take transaction costs into consideration, thus limiting their practical usage. In recent study (Jiang et al., 2017; Liang et al., 2018), transaction costs were considered but it did not address the challenge of having insufficient data in financial markets for the training of robust machine learning algorithms. Moreover, the methods proposed in (Jiang et al., 2017; Liang et al., 2018) directly apply a model-free RL algorithm that is sample inefficient and also doesn't account for the stability and risk issues caused by non-stationary financial market environment. In this paper, we propose a novel model-based RL approach, that takes into account practical trading restrictions such as transaction costs and order executions, to stably train an autonomous agent whose investment decisions are risk-averse yet profitable.

We highlight our main contributions to realize a model-based RL algorithm for our problem setting. Our first contribution is an infused prediction module (IPM), which incorporates the prediction of expected future observations into state-of-the-art RL algorithms. Our idea is inspired by some

[*]Equal contribution [1]Neuri PTE LTD, One George Street, #22-01, Singapore 049145. **Correspondence to: Sakyasingha Dasgupta <sakya@neuri.ai>.

RWSDM Workshop, ICML 2019

attempts to merge prediction methods with RL. For example, RL has been successful in predicting the behavior of simple gaming environments (Oh et al., 2015). In addition, prediction based models have also been shown to improve the performance of RL agents in distributing energy over a smart power grid (Marinescu et al., 2017). In this paper, we explore two prediction models; a nonlinear dynamic Boltzmann machine (Dasgupta & Osogami, 2017) and a variant of parallel WaveNet (van den Oord et al., 2018). These models make use of historical prices of all assets in the portfolio to predict the future price movements of each asset, in a codependent manner. These predictions are then treated as additional features that can be used by the RL agent to improve its performance. Our experimental results show that using IPM provides significant performance improvements over baseline RL algorithms in terms of Sharpe ratio (Sharpe, 1966), Sortino ratio (Sortino & Price, 1994), maximum drawdown (MDD, see (Chekhlov et al., 2005)), value-at-risk (VaR, see (Artzner et al., 1999)) and conditional value-at-risk (CVaR, see (Rockafellar et al., 2000)).

Our second contribution is a behavior cloning module (BCM), which provides one-step greedy expert demonstration to the RL agent. Our idea comes from the imitation learning paradigm (also called learning from demonstrations), with its most common form being behavior cloning, which learns a policy through supervision provided by expert state-action pairs. In particular, the agent receives examples of behavior from an expert and attempts to solve a task by mimicking the expert's behavior, e.g., (Bain & Sommut, 1999; Abbeel & Ng, 2004; Ross et al., 2011; Kimura et al., 2018). In RL, an agent attempts to maximize expected reward through interaction with the environment. Our proposed BCM combines aspects of conventional RL algorithms and supervised learning to solve complex tasks. This technique is similar in spirit to the work in (Nair et al., 2018). The difference is that we create the expert behavior based on a one-step greedy strategy by solving an optimization problem that maximizes immediate rewards in the current time step. Additionally, we only update the actor with respect to its auxiliary behavior cloning loss in an actor-critic algorithm setting. We demonstrate that BCM can prevent large changes in portfolio weights and thus keep the volatility low, while also increasing returns in some cases.

To the best of our knowledge, this is the first work that leverages the deep RL state-of-art, and further extends it to a model-based setting for real-world application in financial portfolio management. Here we demonstrate our approach extending the standard the off-policy actor-critic algorithm - deep deterministic policy gradients (DDPG) (Lillicrap et al., 2015)). Additionally in the appendix, we also provide algorithms for differential risk sensitive model-based deep RL for portfolio optimization. For the rest of the main paper, our discussion will be centered around how our model-based architecture can improve the performance of the off-policy DDPG algorithm.

2. Preliminaries and Problem Setup

In this section, we briefly review the literature of deep reinforcement learning and introduce the mathematical formulation of the dynamic portfolio optimization problem.

A Markov Decision Process (MDP) is defined as a 6-tuple $\langle T, \gamma, \mathcal{S}, 3\mathcal{A}, P, r\rangle$. Here, T is the (possibly infinite) decision horizon; $\gamma \in (0, 1]$ is the discount factor; $\mathcal{S} = \bigcup_t \mathcal{S}_t$ is the state space and $\mathcal{A} = \bigcup_t \mathcal{A}_t$ is the action space, both assumed to be finite dimensional and continuous; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition kernel and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. Policy is a mapping $\mu : \mathcal{S} \to \mathcal{A}$, specifying the action to choose in a particular state. At each time step $t \in \{1, \ldots, T\}$, the agent in state $s_t \in \mathcal{S}_t$ takes an action $a_t = \mu(s_t) \in \mathcal{A}_t$, receives the reward r_t and transits to the next state s_{t+1} according to P. The agent's objective is to maximize its expected return given the start distribution, $J^\mu \triangleq \mathbb{E}_{s_t\sim P, a_t\sim\mu}[\sum_{t=1}^{T} \gamma^{t-1} r_t]$. The state-action value function, or the Q value function, is defined as $Q^\mu(s_t, a_t) \triangleq \mathbb{E}_{s_{i>t}\sim P, a_{i>t}\sim\mu}[\sum_{i=t}^{T} \gamma^{(i-t-1)} r_i | s_t, a_t]$.

Deep deterministic policy gradient (DDPG) algorithm (Lillicrap et al., 2015) is an off-policy model-free reinforcement learning algorithm for continuous control which utilize large function approximators such as deep neural networks. DDPG is an actor-critic method, which bridges the gap between policy gradient methods and value function approximation methods for RL. Intuitively, DDPG learns a state-action value function (critic) by minimizing the Bellman error, while simultaneously learning a policy (actor) by directly maximizing the estimated state-action value function with respect to the network parameters.

Financial portfolio management is the process of constant redistribution of available funds to a set of financial assets. Our goal is to create a dynamic portfolio optimization scheme that periodically generates trading decisions and then act on these decisions autonomously. We consider a portfolio of $m + 1$ assets, including m risky assets and 1 risk-free asset (e.g., cash or U.S. treasury bond). We introduce such notation: given a matrix $\mathbf{g}$, we denote the i^{th} row of $\mathbf{g}$ by $\mathbf{g}_{i,:}$, and the j^{th} column by $\mathbf{g}_{:,j}$. We denote the closing, high and low price vectors of trading period t as $\mathbf{p}_t$, $\mathbf{p}_t^{\text{h}}$ and $\mathbf{p}_t^{\text{l}}$ where $p_{i,t}$ is the closing price of the i^{th} asset in the t^{th} period. In this paper, we choose the first asset to be risk-free cash, i.e., $p_{0,t} = p_{0,t}^{\text{h}} = p_{0,t}^{\text{l}} = 1$. We further define the price relative vector of the t^{th} trading period as $\mathbf{u}_t \triangleq \mathbf{p}_t \oslash \mathbf{p}_{t-1} = (1, p_{1,t}/p_{1,t-1}, \ldots, p_{m,t}/p_{m,t-1})^\top$ where $\oslash$ denotes the element-wise division. In addition, we let $h_{i,t} \triangleq (p_{i,t} - p_{i,t-1})/p_{i,t-1}$ denote the percentage change

of closing price at time t for asset i, the space associated with its vector form $\mathbf{h}_{:,t}$ ($\mathbf{h}_{i,:}$) as $\mathcal{H}_{:,t} \subset \mathbb{R}^m$ ($\mathcal{H}_{i,:} \subset \mathbb{R}^{k_1}$) where k_1 is the time embedding of prediction model. We define $\mathbf{w}_{t-1}$ as the portfolio weight vector at the beginning of trading period t where its i^{th} element $w_{i,t-1}$ represents the proportion of asset i in the portfolio after capital reallocation and $\sum_{i=0}^{m} w_{i,t} = 1$ for all t. We initialize our portfolio with $\mathbf{w}_0 = (1, 0, \ldots, 0)^\top$. Due to price movements in the market, at the end of the same period, the weights evolve according to $\mathbf{w}_t^{'} = (\mathbf{u}_t \odot \mathbf{w}_{t-1})/(\mathbf{u}_t \cdot \mathbf{w}_{t-1})$, where $\odot$ is the element-wise multiplication. Our goal at the end of period t is to reallocate portfolio vector from $\mathbf{w}_t^{'}$ to $\mathbf{w}_t$ by selling and buying relevant assets. Paying all commission fees, this reallocation action shrinks the portfolio value by a factor $\bar{c}_t \triangleq c \sum_{i=1}^{m} |w_{i,t}^{'} - w_{i,t}|$ where c is the transaction fees for purchasing and selling. In particular, we let ρ_{t-1} denote the portfolio value at the beginning of period t and $\rho_t^{'}$ at the end. We then have $\rho_t = \bar{c}_t \rho_t^{'}$. The immediate reward is the logarithmic rate of return defined by $r_t \triangleq \ln(\rho_t/\rho_{t-1}) = \ln(\bar{c}_t \rho_t^{'}/\rho_{t-1}) = \ln(\bar{c}_t \mathbf{u}_t \cdot \mathbf{w}_{t-1})$.

We define the normalized close price matrix at time t by $\mathbf{P}_t \triangleq [\mathbf{p}_{t-k_2+1} \oslash \mathbf{p}_t | \mathbf{p}_{t-k_2+2} \oslash \mathbf{p}_t | \cdots | \mathbf{p}_{t-1} \oslash \mathbf{p}_t | \mathbf{1}]$ where $\mathbf{1} \triangleq (1, 1, \ldots, 1)^\top$ and k_2 is the time embedding. The normalized high price matrix $\mathbf{P}_t^{\text{h}}$ is defined by $\mathbf{P}_t^{\text{h}} \triangleq [\mathbf{p}_{t-k_2+1}^{\text{h}} \oslash \mathbf{p}_t | \mathbf{p}_{t-k_2+2}^{\text{h}} \oslash \mathbf{p}_t | \cdots | \mathbf{p}_{t-1}^{\text{h}} \oslash \mathbf{p}_t | \mathbf{p}_t^{\text{h}} \oslash \mathbf{p}_t]$, and low price matrix $\mathbf{P}_t^{\text{l}}$ can be defined similarly. We further define the price tensor as $\mathbf{Y}_t \triangleq [\mathbf{P}_t \ \mathbf{P}_t^{\text{h}} \ \mathbf{P}_t^{\text{l}}]$. Our objective is to design a RL agent that observes the state $s_t \triangleq (\mathbf{Y}_t, \mathbf{w}_{t-1})$ and takes a sequence of actions (portfolio weights) over the time $a_t = \mathbf{w}_t$ such that the final portfolio value $\rho_T = \rho_0 \mathbb{E}_{s_t \sim P, a_t \sim \mu}[\exp(\sum_{t=1}^{T} \gamma^{t-1} r_t)]$ is maximized.

3. System Architecture

In this section, we discuss the detailed design of our proposed RL based automatic trading system.

The portfolio optimization framework referenced in this paper is represented in Figure 1 and is a modular system composed of a data handler (DH), an algorithm engine (AE) and a market simulation environment (MSE). The DH retrieves market data and deals with the required data transformations. It is designed for continuous data ingestion. The AE consists of the infused prediction module (IPM), the behavior cloning module (BCM) and the RL agent. We refer the readers to Algorithm 1 in the supplementary material for further details. The MSE is an online event-driven module that provides feedback of executed trades, which can eventually be used by the AE to compute rewards. In addition, it also executes investment decisions made by the AE. The strategy applied in this study is an asset allocation system that rebalances the available capital between a portfolio of pre-selected assets including cash on a daily

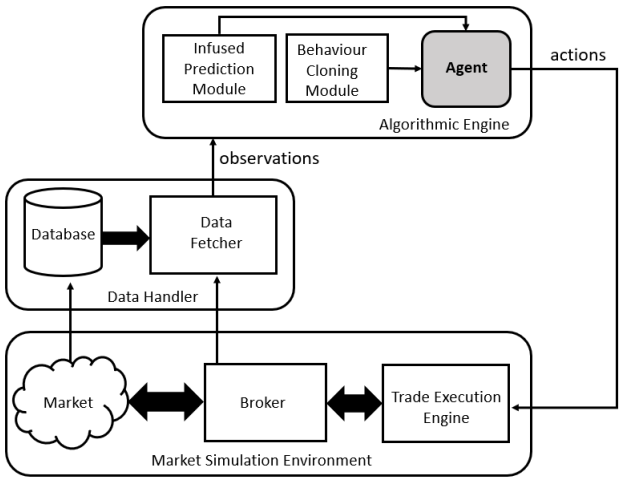


Figure 1. Autonomous portfolio optimization framework.

decision frequency.

The data used in this paper is a mix of U.S. equities[1] on tick level (trade by trade) aggregated to form open-high-low-close (OHLC) bars on an daily frequency.

In the financial portfolio management, a common benchmark strategy is the constantly rebalanced portfolio (CRP) (Cover, 1991), where at each period the portfolio is rebalanced to the initial wealth distribution among the $m + 1$ assets including the cash. Here in additional to baseline model-free DDPG agent, we use CRP as a benchmark. Transaction fees c have been fixed at a conservative level of 20 basis points[2] and, given the use of market orders, an additional 50 basis points of slippage is applied.

We evaluate the performance of the trained RL agent and benchmarks using standard measures as Sharpe and Sortino ratios, value-at-risk (VaR), conditional value-at-risk (CVaR), maximum drawdown (MDD), annualized volatility and annualized returns. Let Y denote a bounded random variable. The Sharpe ratio of Y is defined as $\text{SR} \triangleq \mathbb{E}[Y]/\sqrt{\text{var}[Y]}$. Sharpe ratio, representing the reward per unit of risk, has been recognized not to be always desirable since it is a symmetric measure of risk and, hence, penalizes the low-cost events. Sortino ratio, VaR and CVaR are risk measures which gained popularity for taking into consideration only

[1] We use data from Refinitiv DataScope, with experiments carried out on the following U.S. Equities: Costco Wholesale Corporation, Cisco Systems, Ford Motors, Goldman Sachs, American International Group and Caterpillar. The portfolio selection procedure is outside the scope of this paper. An additional asset is cash (in U.S. dollars), representing the amount of capital not invested in any other asset. Furthermore, a generic market variable (S&P 500 index) is used as additional feature. The data is shared in supplementary files, which will be made available publicly later.

[2] One basis point is equivalent to 0.01%.

the unfavorable part of the return distribution, or, equivalently, unwanted high cost. Sortino ratio is defined similarly to Sharpe ratio, though replacing the standard deviation $\sqrt{\text{var}[Y]}$ with the downside deviation. The VaR_α with level $\alpha \in (0, 1)$ of Y is the $(1-\alpha)$-quantile of Y, and CVaR_α at level α is the expected return of Y in the worst $(1-\alpha)$ of cases.

3.1. Network Architecture

In order to illustrate our proposed off-policy version of dynamic portfolio optimization algorithm, we adapt the actor-critic style DDPG algorithm (Lillicrap et al., 2015). In this setting, at least two networks (one for the actor and one for the critic) are required in the agent as shown in Figure 2. Furthermore, our implementation utilizes both target networks (Mnih et al., 2015) and parameter noise exploration (Plappert et al., 2017), which in itself necessitates two additional networks for the actor. Our agent, comprising of six separate networks (four networks for the actor and two networks for the critic), is described in Algorithm 1 of supplementary material Section A.

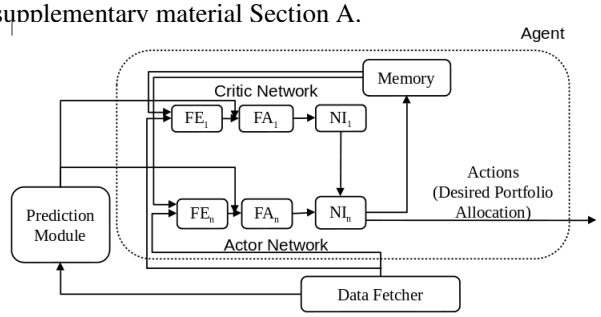


Figure 2. Agent architecture.

We next discuss how the agent is trained and tested. For each episode during training, we select an episode of data by selecting a random trading date that satisfies our desired episode length. At each time step of the episode, market data, i.e., data corresponding to the current time step of interest, fetched via the data fetcher is used to train the IPM which produces predictions of price percentage changes in the next time step. At the same time, the agent's networks are updated by making use of data sampled from the memory (also referred to as the replay buffer) via the prioritized experience replay (Schaul et al., 2015). Once the agent's networks are updated, an action, i.e., desired portfolio allocation in the next time step, can be obtained from the actor network that has been perturbed with the parameter noise. The MSE then executes this noisy action and the agent moves to the next state. The corresponding state, action, rewards, next state and computed one-step greedy action produced by the BCM, are stored in the memory. This process is repeated for each step in the episode and for all episodes. During the testing period, the IPM continues to be updated at each time step while the agent is frozen. Such that, it is no longer trained and actions are obtained from the actual actor network, i.e., the actor network without parameter noise.

As shown in Figure 2, actor and critic networks in the agent consist of feature extraction (FE), feature analysis (FA) and network improvement (NI) components. The FE layers aim to extract features given the current price tensor $\mathbf{Y}_t$. In our experiments, we have pre-selected 8 assets including cash and a time embedding $k_2 = 10$. This essentially means that we have a price tensor of the shape $3 \times 8 \times 10$ if using the channels first convention. The FE layers can be either LSTM-based recurrent neural networks (RNN, see (Hochreiter & Schmidhuber, 1997)) or convolutional neural networks (CNN, see (Krizhevsky et al., 2012)). We find that the former typically yields better performance. Thus, the price tensor is reshaped to a 24×10 tensor prior to being fed to the LSTM-based FE network.

The outputs at each time step of the LSTM-based FE network are concatenated together into a single vector. Next, the previous actions, $\mathbf{w}_{t-1}$, is concatenated to this vector. Finally, it is further concatenated with a one-step predicted price percentage change vector produced by the IPM and a market index performance indicator (i.e., the price ratio of a market index such as the S&P 500 index). The resulting vector is then passed to a series of dense layers (i.e., multilayer perceptrons), which we refer to as the feature analysis (FA) component.

Finally, we have a network improvement (NI) component for each network which specifies how the network is updated. Specifically, NI synchronizes the learning rates between the actor and the critic, which preserves training stability by ensuring the actor is updated at a slower rate than the critic (Bhatnagar et al., 2009). It is also important to note that the actor network's NI component receives gradients from the BCM, which makes use of one-step greedy actions to provide supervised updates to the actor network to reduce portfolio volatility.

3.2. Infused Prediction Module

Here, we implemented and evaluated two different multivariate prediction models, trained in an online manner, differing in their computational complexity. As the most time efficient model, we implemented the nonlinear dynamic Boltzmann machine (Dasgupta & Osogami, 2017) (NDyBM) that predicts the future price of each asset conditioned on the history of all assets in the portfolio. As NDyBM does not require backpropagation through time, it has a parameter update time complexity of $\mathcal{O}(1)$. This makes it very suitable for fast computation in online time-series prediction scenarios. We also implemented another version of IPM using dilated convolution layers inspired by the WaveNet architecture (van den Oord et al., 2018). As there was no significant

difference in predictive performance noticed between the two models, in the rest of the paper we provide results with the computationally efficient NDyBM based IPM module. However, details of our WaveNet inspired architecture can be seen in supplementary material.

We use the state-space augmentation technique, and construct the augmented state-space $\widetilde{\mathcal{S}} \triangleq \mathcal{S}_t \times \mathcal{H}_{:,t+1} \times \mathcal{H}_{:,t+1} \times \mathcal{H}_{:,t+1}$: each state is now a pair $\tilde{s}_t \triangleq (s_t, \mathbf{x}_{t+1})$, where $s_t \in \mathcal{S}_t$ is the original state, and $\mathbf{x}_{t+1} \triangleq (\mathbf{h}_{:,t+1}, \mathbf{h}^{\mathrm{h}}_{:,t+1}, \mathbf{h}^{\mathrm{l}}_{:,t+1}) \in \mathcal{X} \subset \mathbb{R}^{m\times 3}$ where $\mathbf{h}_{:,t+1}, \mathbf{h}^{\mathrm{h}}_{:,t+1}, \mathbf{h}^{\mathrm{l}}_{:,t+1} \in \mathcal{H}_{:,t+1}$ is the predicted future close, high and low asset percentage price change tensor.

The NDyBM can be seen as an unfolded Gaussian Boltzmann machine for an infinite time horizon i.e. $T \to \infty$ history, that generalizes a standard vector auto-regressive model with eligibility traces and nonlinear transformation of historical data (Dasgupta & Osogami, 2017). It represents the conditional probability density of $\mathbf{x}^{[t]}$ given $\mathbf{x}^{[:t-1]}$ as, $p(\mathbf{x}^{[t]}|\mathbf{x}^{[:t-1]}) = \prod_{j=1}^{N} p_j(x_j^{[t]}|\mathbf{x}^{[:t-1]})$[3]. Where, each factor of the right-hand side denotes the conditional probability density of $x_j^{[t]}$ given $\mathbf{x}^{[:t-1]}$ for $j = 1, \ldots, N$. Where, $N = m \times 3$ are the number of units in the NDyBM. Here, $x_j^{[t]}$ is considered to have a Gaussian distribution for each j: $p_j(x_j^{[t]}|\mathbf{x}^{[:t-1]}) = \frac{1}{\sqrt{2\,\pi\,\sigma_j^2}} \exp\Big(-\frac{\left(x_j^{[t]}-\mu_j^{[t]}\right)^2}{2\sigma_j^2}\Big)$.
Here, $\boldsymbol{\mu} \triangleq (\mu_j)_{j=1,\ldots,N}$ is the vector of expected values of the j-th unit at time t given the history up to $t-1$ patterns. It is represented as:
$\boldsymbol{\mu}^{[t]} = \mathbf{b} + \sum_{\delta=1}^{d-1} \mathbf{F}^{[\delta]}\mathbf{x}^{[t-\delta]} + \sum_{k=1}^{K} \mathbf{G}_k\, \boldsymbol{\alpha}_k^{[t-1]}$.
Where, $\mathbf{b} \triangleq (b_j)_{j=1,\ldots,N}$ is a bias vector, $\boldsymbol{\alpha}_k^{[t-1]} \triangleq (\alpha_{j,k}^{[t-1]})_{j=1,\ldots,N}$ are K eligibility trace vectors, d is the time-delay between connections (i, j) and
$\mathbf{F}^{[\delta]} \triangleq (\tilde{f}_{i,j})_{(i,j)\in\{1,\ldots,N\}^2}$ for $0 < \delta < d$, $\mathbf{G}_k \triangleq (g_{i,j,k})_{(i,j)\in\{1,\ldots,N\}^2}$ for $k = 1, \ldots, K$ are $N \times N$ weight matrices. The eligibility trace can be updated recursively as, $\alpha_{i,j,k}^{[t]} = \lambda_k\, \alpha_{i,j,k}^{[t-1]} + x_i^{[t-d_{i,j}+1]}$. Here, λ_k is a fixed decay rate factor defined for each of the k column vectors. Additionally, the bias parameter vector $\mathbf{b}$, is updated at each time using a RNN layer. This RNN layer computes a nonlinear feature map of the past time series. Where in, the output weights from the RNN to the bias layer along with other NDyBM parameters (bias, weight matrices and variance), are updated online using a stochastic gradient method.

Following (Dasgupta & Osogami, 2017)[4], the NDyBM is trained to predict the next time-step close, high and low percentage change for each asset conditioned on the history of all other assets, such that the log-likelihood of each time-series is maximised. As such NDyBM parameters are updated at each step t, following the gradient of the conditional probability density of $\mathbf{x}^{[t]}$: $\nabla_\theta \log p(\mathbf{x}^{[t]}|\mathbf{x}^{[-\infty,t-1]}) = \sum_{i=1}^{N} \nabla_\theta \log p_i(x_i^{[t]}|\mathbf{x}^{[-\infty,t-1]})$.

In the spirit of providing the agent with additional market signals and removing non-Markovian dynamics in the model, along with the prediction, we further augment the state space with a market index performance indicator. The state space now has the form $\tilde{s}_t \triangleq (s_t, \mathbf{x}_{t+1}, I_t)$ where, $I_t \in \mathbb{R}_+$ is the market index performance indicator for time step t.

3.3. Behavior Cloning Module

In finance, some investors may favor a portfolio with lower volatility over the investment horizon. To achieve this, we propose expert behavior cloning , with the primary purpose of reducing portfolio volatility while maintaining reward to risk ratios. One can broadly dichotomize imitation learning into a passive collection of demonstrations (behavioral cloning) versus an active collection of demonstrations. The former setting (Abbeel & Ng, 2004; Ross et al., 2011) assumes that demonstrations are collected a priori and the goal of imitation learning is to find a policy that mimics the demonstrations. The latter setting (Daumé et al., 2009; Sun et al., 2017; Kimura et al., 2018) assumes an interactive expert that provides demonstrations in response to actions taken by the current policy. Our proposed BCM in this definition is an active imitation learning algorithm. In particular, for every step that the agent takes during training, we calculate the one-step greedy action in hindsight. This one-step greedy action is computed by solving an optimization problem, given the next time period's price ratios, current portfolio distribution and transaction costs. The objective function is to maximize returns in the current time step, which is why the computed action is referred to as the one-step greedy action. For time step t, the objective function is as follows:

$$\begin{aligned} \max_{\mathbf{w}_t} \quad & \mathbf{u}_t \cdot \mathbf{w}_t - c \sum_{i=1}^{m} |w_{i,t} - w_{i,t-1}| \\ \text{s.t.} \quad & \sum_{i=0}^{m} w_{i,t} = 1, \quad 0 \le w_{i,t} \le 1, \forall i. \end{aligned} \tag{1}$$

Solving the above optimization problem for $\mathbf{w}_t$ yields an optimal expert greedy action denoted by $\bar{a}_t$. This one-step greedy action is then stored in the replay buffer together with the corresponding (s_t, a_t, r_t, s_{t+1}) pair. In each training iteration of the actor-critic algorithm, a mini-batch of $\{(s_i, a_i, r_i, s_{i+1}, \bar{a}_i)\}_{i=1}^{N}$ is sampled from the replay buffer. Using the states s_i that were sampled from the replay buffer, the actor's corresponding actions are computed and the log-loss between the actor's actions and the one-step greedy actions $\bar{a}_i$ is calculated:

[3]For mathematical convenience, $\mathbf{x}_t$ and $\mathbf{x}^{[t]}$ are used interchangeably.

[4]Details of the learning rule and derivation of the model as per the original paper. Algorithm steps and hyper-parameter settings are provided in supplementary.

Table 1. Deep RL based portfolio optimization agent performance comparison across the different models (last column represents IPM+BCM combined model). Best performance in bold.

	CRP	Model-free Baseline	IPM	BCM	Model-based Combined
Final accnt. value	574859	570482	**586150**	577293	580899
Cummulative return	14.97%	14.11%	**17.23%**	15.46%	16.18%
Annualized return	7.25%	7.14%	**8.64%**	7.77%	8.09%
Annualized volatility	**12.65%**	12.79%	14.14%	12.81%	12.77%
Sharpe ratio	0.57	0.56	0.61	0.61	**0.63**
Sortino ratio	0.80	0.78	0.87	0.85	**0.89**
$\text{VaR}_{0.95}$	1.29%	1.30%	1.41%	1.28%	**1.27%**
$\text{CVaR}_{0.95}$	1.93%	1.93%	2.11%	1.92%	**1.91%**
MDD	13.10%	13.80%	12.60%	12.60%	**12.40%**

$$\bar{L}^{\mu} = -1 \times \frac{\sum_{i=1}^{N} \sum_{j=0}^{m} \bar{a}_{i,j} \log(\mu(s_i))_j + (1 - \bar{a}_{i,j}) \log(1 - (\mu(s_i))_j)}{N(m+1)}. \tag{2}$$

Gradients of the log-loss with respect to the actor network $\nabla_{\theta^{\mu}} \bar{L}^{\mu}$ can then be calculated and used to perturb the weights of the actor slightly. Using this loss directly prevents the learned policy from improving significantly beyond the demonstration policy, as the actor is always tied back to the demonstrations. To achieve this, a fixed factor λ is used to discount the gradients such that the actor network is only slightly perturbed towards the one-step greedy action, thus maintaining the stability of the underlying RL algorithm. Following this, the typical DDPG algorithm (see algorithm 1 in supplementary section A) is executed to train the agent.

4. Experiments

In all experiments[5] the time period spanning from 1 January 2005 to 31 December 2016 (11 years of daily data) was used to train the agent, while the time period between 1 January 2017 to 31 December 2018 (2 years) was used to test the agent out of sample. We initialize our portfolio with $\$500,000$ in cash. We implement a CRP benchmark where funds are equally distributed among all assets, including the cash asset. We also compare the model-free DDPG agent as a baseline (i.e., without infused prediction and behavioral cloning, all other settings remain the same)[6]. It should be noted that, compared to the implementation in prior work (Jiang et al., 2017; Deng et al., 2017; Liang et al., 2018) our baseline agent is situated in a more realistic trading environment that does not assume an immediate trade execution. In particular, our backtesting engine executes market orders at the open of the next OHLC bar, as well as adds slippage to the trading costs. Additional practical constraints are applied such that fractional trades of an asset are not allowed. In addition, in the case of the combined model, at training time the data provided to the agent is augmented with samples from a generative adversarial network (GAN) (Goodfellow et al., 2014; Li et al., 2015) generated HLC (high, low, close) data at a daily frequency. Details of our GAN implementation is provided in the supplementary section E. Augmenting with GAN data in the case of the model-free baseline did not make a significant difference to the performance.

As shown in Table 1, by making use of just IPM, we observe a significant improvement over the baseline in terms of Sharpe (from 0.56 to 0.61) and Sortino (from 0.78 to 0.87) ratios, indicating that we are able to get more returns per unit risk taken. This agent also achieves the highest annualized return overall. However, we note that the volatility of the portfolio is significantly increased (from 12.79% to 14.14%) along with an increase in VaR and CVaR (downside risk) values. As such the IPM by itself provides considerable increase in the return however at the expense of increased risk of the portfolio. One possible reason for increase in risk could be due to the added complexity of IPM module making the agent prone to mistakes when the data comes from the tail end of the distribution or significantly different from training set data distribution.

Introducing the one-step greedy BCM agent considerably reduces the risk of the portfolio as observed from the reduction in annualized volatility, VaR and CVar values. However behavior cloned agent is unable to generalize well as indicated by the reduction in return of the portfolio. To mitigate the drawbacks of the individual modules, we make use of all our contributions as a single model-based framework (i.e. combined model). As shown in Table 1, the combined model is not only better than the performance of the CRP and baseline model free agent, but achieves similar performance as the IPM based model in terms of annualized return, but surpasses all the previous models in terms of both Sharpe (from 0.61 to 0.63) and Sortino ratios (from 0.87 to 0.89). It is worthwhile to note that addition of BCM

[5] Algorithm of our model-based DDPG agent is detailed in supplementary material Section A.

[6] Our baseline is superior to the prior work as (Jiang et al., 2017) due to the addition of prioritized experience replay and parameter noise for better exploration.

has a strong impact in reducing volatility (from 12.84% to 12.77%), MDD (from 12.7% to 12.4%) and the value at risk measures (VaR and CVaR).

We can conclude that IPM by itself considerably improves portfolio management performances in terms of annualized returns along with better Sharpe and Sortino ratio as compared to baseline. This is particularly attractive for investors who aim to maximize their returns per unit risk. However, it is important to note that this may impact the annualized standard deviation or volatility of the performance. BCM as envisioned, helps to reduce portfolio risk as seen by its ability to either reduce volatility or MDD across all runs. It is also interesting to note its ability to improve the risk to return (reward) ratios. Enabling all three modules, our proposed model-based approach can achieve significant performance improvement as compared with benchmark and baseline. The risk sensitivity of the RL agent can be further improved by reward shaping based on risk measures. In this direction, we provide details on an additional risk adjustment module (that can adjust the reward function contingent on risk) with experimental results in supplementary material.

5. Conclusion

In this paper, we proposed a model-based deep reinforcement learning architecture to solve the dynamic portfolio optimization problem. To achieve a profitable and risk-sensitive portfolio, we developed infused prediction and greedy expert behavior cloning modules, and further integrated them into an automatic trading system. The stability and profitability of our proposed model-based RL trading framework were empirically validated on several independent experiments with real market data and practical constraints. In order to provide direct risk sensitive feedback to the RL agent, it is also possible to shape the reward function making use of online measures of Sharpe ratio and downside risk. We formulate such a risk-adjustment mechanism that can work together with our model-based approach. We present the algorithmic details and experimental results with this addition in the supplementary section D. In general our model-based architecture is applicable not only to the financial domain, but also to general reinforcement learning domains which require practical considerations on decision making risk.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1. ACM, 2004.

Arjovsky, M. and Bottou, L. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL http://proceedings.mlr.press/v70/arjovsky17a.html.

Artzner, P., Delbaen, F., Eber, J.-M., and Heath, D. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

Bain, M. and Sommut, C. A framework for behavioural cloning. *Machine intelligence*, 15(15):103, 1999.

Bertsekas, D. P. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 2005.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. Natural actor–critic algorithms. *Automatica*, 45(11): 2471–2482, 2009.

Borovykh, A., Bohte, S., and Oosterlee, C. W. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

Chekhlov, A., Uryasev, S., and Zabarankin, M. Drawdown measure in portfolio optimization. *International Journal of Theoretical and Applied Finance*, 8(01):13–58, 2005.

Cover, T. M. Universal portfolios. *Mathematical finance*, 1 (1):1–29, 1991.

Cumming, J., Alrajeh, D. D., and Dickens, L. *An investigation into the use of reinforcement learning techniques within the algorithmic trading domain*. PhD thesis, 2015.

Dasgupta, S. and Osogami, T. Nonlinear dynamic boltzmann machines for time-series prediction. In *AAAI*, pp. 1833–1839, 2017.

Daumé, H., Langford, J., and Marcu, D. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Dempster, M. A. and Leemans, V. An automated FX trading system using adaptive reinforcement learning. *Expert Systems with Applications*, 30(3):543–552, 2006.

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2017.

Esteban, C., Hyland, S. L., and Rätsch, G. Real-valued (medical) time series generation with recurrent conditional GANs. *arXiv preprint arXiv:1706.02633*, 2017.

Feng, Y., Palomar, D. P., and Rubio, F. Robust optimization of order execution. *IEEE Transactions on Signal Processing*, 63(4):907–920, Feb 2015. ISSN 1053-587X. doi: 10.1109/TSP.2014.2386288.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Gretton, Arthur, Borgwardt, M., K., Rasch, Malte, Schlkopf, Bernhard, and Alexander. A kernel two-sample test, Dec 2012. URL http://jmlr.csail.mit.edu/papers/v13/gretton12a.html.

Guo, Y., Fu, X., Shi, Y., and Liu, M. Robust log-optimal strategy with reinforcement learning. *arXiv preprint arXiv:1805.00205*, 2018.

Haugen, R. A. and Haugen, R. A. Modern investment theory. 1990.

Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaeger, H. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pp. 609–616, 2003.

Jaeger, H. and Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

Jiang, Z., Xu, D., and Liang, J. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1706.10059*, 2017.

Kimura, D., Chaudhury, S., Tachibana, R., and Dasgupta, S. Internal model from observations for reward shaping. *arXiv preprint arXiv:1806.01267*, 2018.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Li, B. and Hoi, S. C. Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 46(3):35, 2014.

Li, Y., Swersky, K., and Zemel, R. Generative moment matching networks. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1718–1727, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/li15.html.

Liang, Z., Chen, H., Zhu, J., Jiang, K., and Li, Y. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Marinescu, A., Dusparic, I., and Clarke, S. Prediction-based multi-agent reinforcement learning in inherently non-stationary environments. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 12(2):9, 2017.

Markowitz, H. M. Portfolio selection: Efficient diversification of investment. 344 p, 1959.

Mittelman, R. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Moody, J. and Saffell, M. Learning to trade via direct reinforcement. *IEEE transactions on neural networks*, 12 (4):875–889, 2001.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Oh, J., Guo, X., Lee, H., Lewis, R. L., and Singh, S. Action-conditional video prediction using deep networks in atari games. In *Advances in neural information processing systems*, pp. 2863–2871, 2015.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Rockafellar, R. T., Uryasev, S., et al. Optimization of conditional value-at-risk. *Journal of risk*, 2:21–42, 2000.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Sharpe, W. F. Mutual fund performance. *The Journal of business*, 39(1):119–138, 1966.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

Sortino, F. A. and Price, L. N. Performance measurement in a downside risk framework. *the Journal of Investing*, 3 (3):59–64, 1994.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

van den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., van den Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., Walters, T., Belov, D., and Hassabis, D. Parallel WaveNet: Fast high-fidelity speech synthesis. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3918–3926, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/oord18a.html.

A. Algorithm

Our proposed model-based off-policy actor-critic style RL architecture is summarized in Algorithm 1.

Algorithm 1 Model-based Deep Reinforcement Learning for Dynamic Portfolio Optimization

Input: Critic $Q(\tilde{s}, a|\theta^Q)$, actor $\mu(\tilde{s}|\theta^\mu)$ and perturbed actor networks $\mu(\tilde{s}|\theta^{\tilde{\mu}})$ with weights θ^Q, θ^μ, $\theta^{\tilde{\mu}}$ and standard deviation of parameter noise σ.
Initialize target networks Q', μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode $= 1, \ldots, M$ **do**
  Receive initial observation state s_1
  for $t = 1, \ldots, T$ **do**
    Predict future price tensor $\mathbf{x}_{t+1}$ with prediction models using s_t and form augmented state $\tilde{s}_t$
    Use perturbed weight $\theta^{\tilde{\mu}}$ to select action $a_t = \mu(\tilde{s}_t|\theta^{\tilde{\mu}})$
    Execute action a_t, observe reward r_t and new state s_{t+1}
    Predict next future price tensor $\mathbf{x}_{t+2}$ using prediction models with inputs s_{t+1} and form the augmented state $\tilde{s}_{t+1}$
    Solve the optimization problem (1) for the expert greedy action $\bar{a}_t$
    Store transition $(\tilde{s}_t, a_t, r_t, \tilde{s}_{t+1}, \bar{a}_t)$ in R
    Sample a minibatch of N transitions, $(\tilde{s}_i, a_i, r_i, \tilde{s}_{i+1}, \bar{a}_i)$, from R via prioritized replay, according to temporal difference error
    Compute $y_i = r_i + \gamma Q'(\tilde{s}_{i+1}, \mu'(\tilde{s}_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update the critic θ^Q by annealing the prioritized replay bias while minimizing the loss:

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - Q(\tilde{s}_i, a_i|\theta^Q))^2$$

    Maintain the ratio between the actual learning rates of the actor and critic
    Update θ^μ using the sampled policy gradient:

$$\frac{1}{N}\sum_i \nabla_a Q(\tilde{s}, a|\theta^Q)|_{\tilde{s}=\tilde{s}_i, a=\mu(\tilde{s}_i|\theta^\mu)} \nabla_{\theta^\mu}\mu(\tilde{s}|\theta^\mu)|_{\tilde{s}=\tilde{s}_i}$$

    Calculate the expert auxiliary loss $\bar{L}$ in (2) and update θ^μ using $\nabla_{\theta^\mu}\bar{L}^\mu$ with factor λ
    Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
    Create adaptive actor weights $\tilde{\mu}'$ from current actor weight θ^μ and current σ: $\theta^{\tilde{\mu}'} \leftarrow \theta^\mu + \mathcal{N}(0, \sigma)$
    Generate adaptive perturbed actions $\tilde{a}'$ for the sampled transition starting states $\tilde{s}_i$: $\tilde{a}' = \mu(\tilde{s}_i|\theta^{\tilde{\mu}'})$. With previously calculated actual actions $a = \mu(\tilde{s}_i|\theta^\mu)$, calculate the mean induced action noise: $d(\theta^\mu, \theta^{\tilde{\mu}'}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_s\left[(a_i - a_i')^2\right]}$
    Update σ: if $d(\theta^\mu, \theta^{\tilde{\mu}'}) \leq \delta$, $\sigma \leftarrow \alpha\sigma$, otherwise $\sigma \leftarrow \sigma/\alpha$
  end for
  Update perturbed actor: $\theta^{\tilde{\mu}} \leftarrow \theta^\mu + \mathcal{N}(0, \sigma)$
end for

Table 2. Hyperparameters for the DDPG actor and critic networks.

	Actor network	Critic network
FE layer type	RNN bidirectional LSTM	RNN bidirectional LSTM
FE layer size	20, 8	20, 8
FA layer type	Dense	Dense
FA layer size	256,128,64,32	256,128,64,32
FA layer activation function	Leaky relu	Leaky relu
Optimizer	Gradient descent optimizer	Adam optimizer
Dropout	0.5	0.5
Learning rate	Synchronized to be 100 times slower than the critic's actual learning rate	10^{-3}
Episode length	650	
Number of episodes	200	
σ for parameter noise	0.01	
Replay buffer size	1000	

B. Hyperparameters for Experiments

We report the hyperparameters for the actor and critic networks in Table 2.

Nonlinear Dynamic Boltzmann machine in the infused prediction module uses the following hyper-parameter settings: delay $d = 3$, decay rates $\lambda = [0.1, 0.2, 0.5, 0.8]$ i.e. $k = 4$, learning rate was 10^{-3}, with standard RMSProp optimizer. The input and output dimensions were fixed at three times the number of assets, corresponding to the high, low and close percentage change values. The RNN layer dimension is fixed at 100 units with a `tanh` nonlinear activation function. A zero mean, 0.01 standard deviation noise was applied to each of the input dimensions at each time step, in order to slightly perturb the inputs to the network. This injected noise was cancelled by applying a standard[7] Savitzky-Golay (savgol) filter with window length 5 and polynomial order 3.

For the variant of WaveNet (van den Oord et al., 2018) in the infused prediction module, we choose the number of dilation levels L to be 6, filter length f to be 2, the number of filters to be 32 and the learning rate to be 10^{-4}. Inputs are scaled with min-max scaler with a window size equal to the receptive field of the network, which is $f^L + \sum_{i=0}^{L-2} f^i = 95$.

For the RNN-GAN, we select the number of training set to be 30,000, noise latent dimension H to be 8, batch size to be 128, time embedding k_1 to be 95, generator RNN hidden units to be 32, discriminator RNN hidden unit to be 32 and a learning rate 10^{-3}. For each episode in Algorithm 1, we generate and append two months of synthetic market data for each asset.

For the BCM, we choose the factor $\lambda = 0.1$ to discount the gradient of the log-loss.

[7] As implemented in scientific computing package Scipy.

C. Details of Infused Prediction Module

C.1. Nonlinear Dynamic Boltzmann Machine Algorithm

In Figure 4 we show the typical unfolded structure of a nonlinear dynamic Boltzmann machine. The RNN layer (of a reservoir computing type architecture) (Jaeger, 2003; Jaeger & Haas, 2004) is used to create nonlinear feature transforms of the historical time-series and update the bias parameter vector as follows: $\mathbf{b}^{[t]} = \mathbf{b}^{[t-1]} + \mathbf{A}^\top \mathbf{\Psi}^{[t]}$

Where, $\mathbf{\Psi}^{[t]}$ is a $M \times 1$ dimensional state vector at time t of a M dimensional RNN. $\mathbf{A}$ is the $M \times N$ dimensional learned output weight matrix that connects the RNN state to the bias vector. The RNN state is updated based on the input time-series $\mathbf{x}^{[t]}$ as follows: $\mathbf{\Psi}^{[t]} = \texttt{tanh}\Big(\mathbf{W}_{rnn}\mathbf{\Psi}^{[t-1]} + \mathbf{W}_{in}\mathbf{x}^{[t]}\Big)$.

Here, $\mathbf{W}_{rnn}$ and $\mathbf{W}_{in}$ are the RNN internal weight matrix and the weight matrix corresponding to the projection of the time-series input to the RNN layer, respectively.

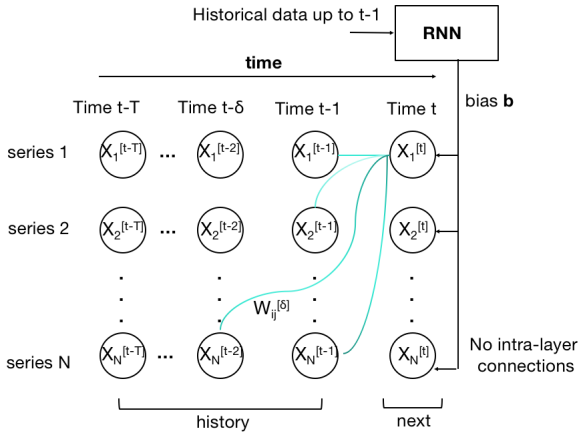
The NDyBM is trained online to predict $\tilde{\mathbf{x}}^{[t+1]}$ based on the estimated $\mu^{[t]}$ for all time observations $t = 1, 2, 3, \ldots$. The parameters are updated such that the log-likelihood $LL(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}} \sum_t \log p(\mathbf{x}^{[t]}|\mathbf{x}^{[-\infty,t-1]})$, of the given financial time-series data $\mathcal{D}$ is maximized. We can derive exact stochastic gradient update rules for each of the parameters using this objective (can be referenced in the original paper). Such a local update mechanism, allows an $\mathcal{O}(1)$ update of NDyBM parameters. As such a single epoch update of NDyBM occurs in sub-seconds $\ll 1s$ as compared to a tens of seconds update of the WaveNet inspired model. Scaling up to large number of assets in the portfolio, in an online learning scenario this can provide significant computational benefits.

Our implementation of the NDyBM based infused prediction module is based on the open-source code available at

https://github.com/ibm-research-tokyo/dybm. Algorithm 3 describes the basics steps.

Algorithm 2 Online asset price change prediction with the NDybM IPM module.

Require: All the weight and bias parameters of NDyBM are initialized to zero. The RNN weights, $\mathbf{W}_{rnn}$ initialized randomly from $\mathcal{N}(0,1)$, $\mathbf{W}_{in}$ initialized randomly from $\mathcal{N}(0,0.1)$. The FIFO queue is initialized with $d-1$ zero vectors. K eligibility traces $\{\boldsymbol{\alpha}_k^{[-1]}\}_{k=1}^K$ are initialized with zero vectors
Input: Close, high and low percentage price change for each asset at each time step
for $t = 0, 1, 2, \ldots$ **do**
  Compute $\mu^{[t]}$ using $\boldsymbol{\mu}^{[t]} = \mathbf{b} + \sum_{\delta=1}^{d-1} \mathbf{F}^{[\delta]}\mathbf{x}^{[t-\delta]} + \sum_{k=1}^{K} \mathbf{G}_k\, \boldsymbol{\alpha}_k^{[t-1]}$ & update the bias vector based on RNN layer
  Predict the expected price change pattern at time t using $\mu^{[t]}$
  Observe the current time series pattern at $\mathbf{x}^{[t]}$
  Update the parameters of NDyBM based on $\nabla \log p(\mathbf{x}^{[t]}|\mathbf{x}^{[-\infty,t-1]})$
  Update FIFO queues and eligibility traces by $\alpha_{i,j,k}^{[t]} = \lambda_k\, \alpha_{i,j,k}^{[t-1]} + x_i^{[t-d_{i,j}+1]}$
  Update RNN layer state vector
end for



$$p(\mathbf{x}^{[t]}|\mathbf{x}^{[t-T,t-1]}) = \prod_{j=1}^{N} p_j(x_j^{[t]}|\mathbf{x}^{[t-T,t-1]})$$

Figure 4. A nonlinear dynamic Boltzmann machine unfolded in time. There are no connections between units i and j within a layer. Each unit is connected to each other unit only in time. The lack of intra-layer connections enables conditional independence as depicted.

C.2. WaveNet inspired multivariate time-series prediction

We use an autoregressive generative model, which is a variant of parallel WaveNet (van den Oord et al., 2018), to learn the temporal pattern of the percentage price change tensor $\mathbf{x}_t \in \mathbb{R}^{m\times 3}$, which is the part of state space that is assumed to be independent of agent's actions. Our network is inspired by previous works on adapting WaveNet to time series prediction (Mittelman, 2015; Borovykh et al., 2017). We denote the i^{th} asset's price tensor at time t as $\mathbf{x}_{i,t} = (p_{i,t}, p_{i,t}^{\text{h}}, p_{i,t}^{\text{l}})$. The joint distribution of price over time $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ is modelled as a factorized product of probabilities conditioned on a past window of size k_1:

$$p(\mathbf{X}) = \prod_{t=1}^{T}\prod_{i=1}^{m} p(\mathbf{x}_{i,t}|\mathbf{x}_{t-k_1}, \ldots, \mathbf{x}_{t-1}, \theta).$$

The model parameter θ is estimated through maximum likelihood estimation (MLE) respective to $p(\mathbf{X})$. The joint probability is factorized both over time and different assets, and the conditional probabilities is modelled as stacks of dilated causal convolutions. Causal convolution ensures that the output does not depend on future data, which can be implemented by front zero padding convolution output in time dimension such that output has the same size in time dimension as input.

Figure 5 shows the tensorboard visualization of a dilated convolution stack of our WaveNet variant. At each time t, the input window $(\mathbf{x}_{t-k_1}, \ldots, \mathbf{x}_{t-1}) \in \mathcal{X}^{k_1} \subset \mathbb{R}^{m\times 3\times k_1}$ first goes through the common F_1 layer, which is a depthwise separable 2d convolution followed by a 1×1 convolution, with time and features (close, high, low) as height and width and assets as channel. Output from F_1 is then feed into different stacks of the same architecture as depicted in Figure 5, one for each different asset. F_l and R_l denotes dilated convolution with filter length f and relu activation at level l, which has dilation factor $d_l = f^{l-1}$. Each R_l takes the concatenation of F_l and R_{l+1} as input. This concatenation is represented by the residual blocks in the diagram. The final M layer in Figure 5 is a 1×1 convolution with 3 filters, one for each of high, low and close, and the output is exactly $\mathbf{x}_{i,t}$. The output from m different stacks is then concatenated to produce the prediction $\mathbf{x}_t$.

The reason for modelling the probabilities as such is twofold. First, it addresses the dependency on historical patterns of the financial market by using a high order autoregressive model to capture long term patterns. Models with recurrent connections can capture long term information since they have internal memory, but they are slow to train. A fully convolutional model can process inputs in parallel, thus resulting in faster training speed compared to recurrent models, but the number of layers needed is linear to k_1. This inefficiency can be solved by using stacked dilated convolution. A dilated convolution with dilation factor d uses filters with $d-1$ zero inserted between its values, which allows it to operate in a coarser scale than a normal convolution with same effective filter length. When stacking dilated

convolution layers with filter length f, if an exponentially increasing dilation factor $d_l = f^{l-1}$ is used in each layer l, the effective receptive fields will be f^L, where L is the total number of layers. Thus large receptive field can be achieved with having logarithmic many layers to k_1, which has much fewer parameters needed compared to a normal fully convolutional model.

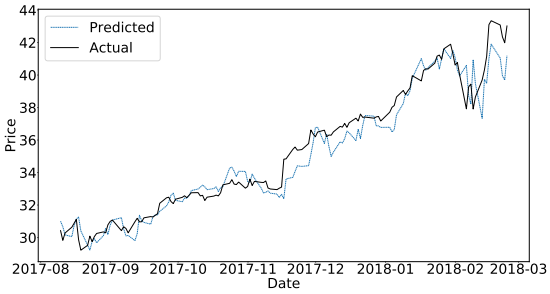


Figure 6. Out of sample actual and predicted closing price movement for asset Cisco Systems.

Secondly, by factoring not only over time but over different assets as well, this model makes parallelization easier and potentially has better interpretability. The prediction of each asset is conditioned on all the asset prices in the past window, which makes the model easier to run in parallel since each stack can be placed on different GPUs.

The serial cross correlation R at lag ℓ for a pair of discrete time series $x(t), y(t)$ is defined as

$$R_{xy}(\ell) = \sum_t x(t)y(t-\ell).$$

Roughly speaking, $R_{xy}(\ell)$ measure the similarity between x and a lagged version of the y. The peak $\alpha = \arg\max_\ell R_{xy}(\ell)$ indicates that $x(t)$ has highest correlation with $y(t+\alpha)$ for all t on average. If $x(t)$ is the predicted time series, and $y(t)$ is the real data, a naive prediction model simply takes the last observation as prediction and would thus have $\alpha = -1$. Sometimes a prediction model will learn this trivial prediction and we call this kind of model trend following. To test whether our model has learned a trend following prediction, we convert the predicted percentage change vector $\mathbf{x}_t$ back to price (see Figure 6) and calculated the serial cross-correlation between the predicted series and the actual. Figure 7 clearly shows that there is no trend following behavior as $\alpha = 0$.

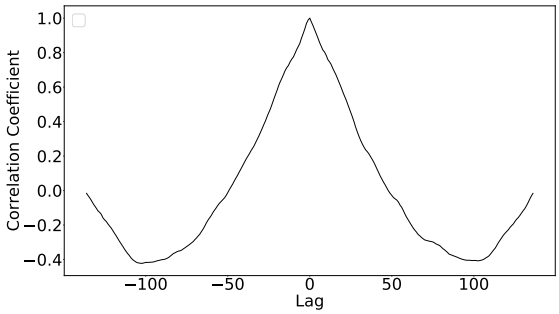


Figure 7. Trend following analysis on predicted closing price movement for asset Cisco Systems.

D. Risk-adjustment Module

In this section, we discuss the risk-adjustment module (RAM) which is aimed to optimize portfolio risk directly. We adapt the framework of recurrent deterministic policy gradient (RDPG, see (Heess et al., 2015)) technique which is a natural extension of DDPG to memory-based control setting with recurrent neural networks, and adjust our risk preference.

Note that the sampled minibatch from the replay buffer in Algorithm 1 is of the form

$$(s_{t_1}, a_{t_1}, r_{t_1}, s_{t_1+1}), \ldots, (s_{t_K}, a_{t_K}, r_{t_K}, s_{t_K+1}).$$

The critic and actor networks are then updated according to these disjointed (non-successive) samples, i.e., $s_{t_k+1} \neq s_{t_{k+1}}, \forall k = 1, \ldots, K-1$. Different from DDPG replay buffer which stores a time-step experience, RDPG replay buffer stores a complete trajectory. In other words, each sample in the DDPG replay buffer is a state-action-reward-next state tuple, while each sample in the RDPG replay buffer is a trajectory starting from a given initial state. When we sample from the RDPG replay buffer, different trajectories are sampled.

Different from the original RDPG (Heess et al., 2015), the update rules of our RDPG are the same as DDPG except that samples from the replay buffer are trajectories instead of disjoint transitions, and that both critic and actor networks in RDPG still take augmented state $\tilde{s}_i$ as inputs. The details of these changes are in Algorithm 3

Having access to trajectory sampling also allows us to apply risk-adjustment to our RL objective. Following (Moody & Saffell, 2001), we consider new risk-aware objectives including differential Sharpe ratio (DSR) and differential downside deviation ratio (DDR). To define the DSR, recall the definition of Sharpe ratio SR and denote its value at time

t by SR_t:

$$\text{SR}_t = \frac{\mathbb{E}[r_t]}{\sqrt{\text{var}[r_t]}}, \tag{3}$$

where r_t is the logarithmic rate of return for period t. The differential Sharpe ratio D_t is then obtained by considering the moving average of the returns and standard deviation of returns in (3), and expanding to first order in the adaptation rate η

$$d_t \triangleq \frac{d\text{SR}_t}{d\eta} = \frac{\omega_{t-1}\Delta\nu_t - 0.5\nu_{t-1}\Delta\omega_t}{(\omega_{t-1} - \nu_{t-1}^2)^{\frac{3}{2}}}, \tag{4}$$

where ν_t and ω_t[8] are exponential moving estimates of the first and second moments of r_t

$$\nu_t = \nu_{t-1} + \eta\Delta\nu_t = \nu_{t-1} + \eta(r_t - \nu_{t-1}),$$
$$\omega_t = \omega_{t-1} + \eta\Delta\omega_t = \omega_{t-1} + \eta(r_t^2 - \omega_{t-1}).$$

The DSR has several attractive properties including facilitating recursive updating, enabling efficient on-line optimization, weighting recent returns more and providing interpretability (Moody & Saffell, 2001).

To define the DDR, we need first to define the downside deviation dd_T as

$$dd_T \triangleq \left(\frac{1}{T}\sum_{t=1}^{T}\min\{r_t, 0\}^2\right)^{\frac{1}{2}},$$

which is the square root of the average of the square of the negative returns. Using the downside deviation as a measure of risk, we can now define the downside deviation ratio (DDR) as

$$\text{DDR}_T \triangleq \frac{\mathbb{E}[r_t]}{dd_T}. \tag{5}$$

The DDR is then defined by considering the exponential moving average of the returns and the squared downside deviation of returns in (6), and by expanding to the first order in the adaption rate η of DDR:

$$d_t \triangleq \frac{d\text{DDR}_t}{d\eta} = \begin{cases} \frac{r_t - 0.5\nu_{t-1}}{dd_{t-1}} & \text{if } r_t > 0, \\ \frac{dd_{t-1}^2(r_t - 0.5\nu_{t-1}) - 0.5\nu_{t-1}r_t^2}{dd_{t-1}^3} & \text{otherwise,} \end{cases} \tag{6}$$

where

$$\nu_t = \nu_{t-1} + \eta(r_t - \nu_{t-1}),$$
$$\text{dd}_t^2 = \text{dd}_{t-1}^2 + \eta(\min\{r_t, 0\}^2 - dd_{t-1}^2).$$

The DDR rewards the presence of large average positive returns and penalizes risky downside returns. The procedure of obtaining the risk-sensitive RDPG agent for dynamic portfolio optimization is summarized in Algorithm 3.

[8]This is different from $\mathbf{w}_t$ in the main paper to denote the portfolio weights.

We perform the experiments with initial values $\nu_0 = \omega_0 = dd_0 = 0$, and add $\varepsilon = 10^{-8}$ to the denominators in (4) and (6) to avoid division by zero. The results of DSR can be found in Table 3 below. We can observe similar phenomenon to before as shown in Table 1, where the rewards were scaled by a fixed number[9]. In particular, the use of IPM drastically improves the Sharpe (from 0.56 to 0.60) and Sortino (from 0.78 to 0.84) ratios. Using the combined model-based approach produces the best results compared to just DSR in terms of Sharpe (from 0.56 to 0.63) and Sortino (from 0.78 to 0.88) ratios.

Results for models with DDR are given in Table 4. We again see that using IPM compared to just DDR improves both Sharpe (from 0.57 to 0.59) and Sortino (from 0.79 to 0.83) ratios. However, we can observe that the annualized volatility is not significantly impacted when comparing IPM with the combined model (from 12.75% to 12.74%). This could be due to the difference in risk adjustment of the rewards.

We show that the use of risk-adjusted rewards such as DSR and DDR does not detrimentally impact performance, moreover, it presents an alternative way of scaling the rewards without the need of heuristically setting a scaling factor.

E. Data Augmentation with Recurrent GAN

The limited historical financial data may prevent us from scaling up the deep RL agent. To mitigate this issue, we provide a mechanism for augmenting the dataset via a recurrent generative adversarial networks (RGAN) framework (Esteban et al., 2017) and generate synthetic financial time series. The RGAN follows the architecture of a regular GAN, with both the generator and the discriminator substituted by recurrent neural networks. Specifically, we generate percentage change of closing price for each asset separately at higher frequency than what the RL agent uses. We then downsample the generated series to obtain the synthetic series of high, low, close (HLC) triplet vector $\mathbf{x}_t$. This approach avoids the non-stationarity in market price dynamics by generating a stationary percentage change series instead, and the generated HLC triplet is guaranteed to maintain their relationship (i.e., generated highs are higher than generated lows).

We assume implicitly that the i^{th} assets' percentage change vector $\mathbf{h}_{i,:}$ follows a distribution: $\mathbf{h}_{i,:} \sim p^i_{\text{data}}(\mathbf{h}_{i,:})$. Let H be the hidden dimension. Our goal is to find a parameterized function G^i_ψ such that given a noise prior $p_z(\mathbf{z}), \mathbf{z} \in \mathbb{R}^{k_1 \times H}$ the generated distribution $p^i_g(G^i_\psi(\mathbf{z}))$ are *empirically similar* to the data distribution $p^i_{\text{data}}(\mathbf{h}_{i,:})$. Formally, given two batches of observations $\{G^i_\psi(\mathbf{z})^{(j)}\}_{j=1}^b, \{\mathbf{h}^{(j)}_{i,:}\}_{j=1}^b$ from

[9]All experiment results shown in Table 1 were generated by scaling the rewards by a factor of 10^3.

Table 3. Performances for different models (all models are risk-adjusted by DSR risk measure. accnt. and ann. are abbreviations for account and annualized respectively).

	DSR	IPM	BCM	Combined model
Final accnt. value	570909	575709	571595	**579444**
Ann. return	7.17%	7.62%	7.24%	**7.96%**
Ann. volatility	12.79%	12.75%	12.81%	**12.68%**
Sharpe ratio	0.56	0.60	0.57	**0.63**
Sortino ratio	0.78	0.84	0.79	**0.88**
$\text{VaR}_{0.95}$	1.30%	1.27%	1.30%	**1.25%**
$\text{CVaR}_{0.95}$	1.93%	1.91%	1.94%	**1.90%**
MDD	13.70%	12.40%	13.70%	**12.20%**

Table 4. Performances for different models (all models are risk-adjusted by DDR risk measure. accnt. and ann. are abbreviations for account and annualized respectively).

	DDR	IPM	BCM	Combined model
Final accnt. value	571790	574791	571737	**578091**
Ann. return	7.26%	7.54%	7.25%	**7.84%**
Ann. volatility	12.81%	12.75%	12.79%	**12.74%**
Sharpe ratio	0.57	0.59	0.57	**0.62**
Sortino ratio	0.79	0.83	0.79	**0.86**
$\text{VaR}_{0.95}$	1.30%	**1.27%**	1.30%	**1.27%**
$\text{CVaR}_{0.95}$	1.94%	**1.91%**	1.93%	**1.91%**
MDD	13.70%	**12.40%**	13.70%	**12.40%**

distributions p_g^i and p_{data}^i where b is the batch size, we want $p_g^i \approx p_{\text{data}}^i$ under certain similarity measure of distributions.

One suitable choice of such similarity measure is the maximum mean discrepancy (MMD) (Gretton et al., 2012). Following (Li et al., 2015), we can show (see supplementary material) that with RBF kernel, minimizing $\widehat{\text{MMD}}_b^2$, which is the biased estimator of squared MMD results in matching all moments between the two distribution p_a, p_b.

As discussed in previous works (Arjovsky et al., 2017; Arjovsky & Bottou, 2017), vanilla GANs suffer from the problem that discriminator becomes perfect when the real and the generated probabilities have disjoint supports (which is often the case under the hypothesis that real-world data lies in low dimensional manifolds). This could lead to generator gradient vanishing, making the training difficult. Furthermore, the generator can suffer from mode collapse issue where it succeeds in tricking the discriminator but the generated samples have low variation.

In our RGAN architecture, we model each asset i separately by a pair of parameterized function D_ϕ^i, G_ψ^i, and we use $\widehat{\text{MMD}}^2$, the unbiased estimator of squared MMD between p_g^i and p_{data}^i, as a regularizer for the generator G_ψ^i, such that the generator not only tries to 'trick' the discriminator into classifying its output as coming from p_{data}^i, but also tries to match p_g^i with p_{data}^i in all moments. This alleviates the aforementioned issues in vanilla GAN: firstly $\widehat{\text{MMD}}^2$ is defined even when distributions have disjoint supports, and secondly gradients provided by $\widehat{\text{MMD}}^2$ are not dependent on the discriminator but only on the real data. Specifically, both the discriminator D_ϕ^i and the generator G_ψ^i is trained with gradient descent method. The discriminator objective is:

$$\max_\phi \frac{1}{b}\sum_{j=1}^{b}\left[\log D_\phi^i(\mathbf{h}_i^{(j)}) + \log\left(1 - D_\phi^i(G_\psi^i(\mathbf{z}^{(j)}))\right)\right],$$

and the generator objective is:

$$\min_\psi \frac{1}{b}\sum_{j=1}^{b}\left[\log\left(1 - D_\phi^i(G_\psi^i(\mathbf{z}^{(j)}))\right)\right] + \zeta\widehat{\text{MMD}}^2$$

given a batch of b samples $\{\mathbf{z}^{(j)}\}_{j=1}^b$ drawn independently from a diagonal Gaussian noise prior $p_z(\mathbf{z})$, and a batch of b samples $\{\mathbf{h}_i^{(j)}\}_{j=1}^b$ drawn from the data distribution p_{data}^i of the i^{th} asset.

In order to select the bandwidth parameter σ in the RBF kernel, we set it to the median pairwise distance between the joint data. Both discriminator and generator networks are LSTMs (Hochreiter & Schmidhuber, 1997).

Although generator directly minimises estimated $\widehat{\text{MMD}}^2$, we go one step further to validate the RGAN by conducting Kolmogorov-Smirnov (KS) test.

Algorithm 3 Risk-adjusted model-based deep reinforcement learning algorithm.

Input: Critic $Q(\tilde{s}, a|\theta^Q)$, actor $\mu(\tilde{s}|\theta^\mu)$ and perturbed actor networks $\mu(\tilde{s}|\theta^{\tilde{\mu}})$ with weights θ^Q, θ^μ, $\theta^{\tilde{\mu}}$, standard deviation of parameter noise σ and risk adaption rate η.
Initialize target networks Q', μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode$= 1, \ldots, M$ **do**
  Receive initial observation state s_1
  Initialize ν_0, ω_0 and dd_0 for DSR and DDR
  for $t = 1 \ldots, T$ **do**
    Predict future price tensor $\mathbf{x}_{t+1}$ using prediction models with inputs s_t and form augmented state $\tilde{s}_t$
    Use perturbed weight $\theta^{\tilde{\mu}}$ to select action $a_t = \mu(\tilde{s}_t|\theta^{\tilde{\mu}})$
    Take action a_t, observe reward r_t and new state s_{t+1}
    Predict next future price tensor $\mathbf{x}_{t+2}$ using prediction models with inputs s_{t+1} and form the augmented state $\tilde{s}_{t+1}$
    Solve the optimization problem (1) for the expert greedy action $\bar{a}_t$
    Compute the DSR or DDR d_t based on (4) or (6)
  end for
  Store trajectory $(\tilde{s}_1, a_1, d_1, \tilde{s}_2, \bar{a}_t, \ldots, \tilde{s}_T, a_T, d_T, \tilde{s}_{T+1}, \bar{a}_T)$ in R
  Sample a minibatch of size K from replay buffer $\{(\tilde{s}_{k_1}, a_{k_1}, d_{k_1}, \tilde{s}_{k_2}, \bar{a}_{k_1}, \ldots, \tilde{s}_{k_T}, a_{k_T}, d_{k_T}, \tilde{s}_{k_{T+1}}, \bar{a}_{k_T})\}_{k=1}^K$.
  Compute $y_{k_t} = d_{k_t} + \gamma Q'(\tilde{s}_{k_{t+1}}, \mu'(\tilde{s}_{k_{t+1}}|\theta^{\mu'})|\theta^{Q'})$ for all $k = 1, \ldots, K$ and $t = 1, \ldots, T$
  Update the critic θ^Q by minimizing the loss:

$$\frac{1}{TK}\sum_{t=1}^{T}\sum_{k=1}^{K}(y_{k_t} - Q(\tilde{s}_{k_t}, a_{k_t}|\theta^Q))^2$$

  Update θ^μ using the sampled policy gradient:

$$\frac{1}{TK}\sum_{t=1}^{T}\sum_{k=1}^{K}\nabla_a Q(\tilde{s}_t, a)|\theta^Q)|_{\tilde{s}_t=\tilde{s}_{k_t}, a=\mu(\tilde{s}_{k_t}|\theta^\mu)} \times \nabla_{\theta^\mu}\mu(\tilde{s}_t|\theta^\mu)|_{\tilde{s}_t=\tilde{s}_{k_t}}$$

  Calculate the expert auxiliary loss $\bar{L}$ in (2) and update θ^μ using $\nabla_{\theta^\mu}\bar{L}^\mu$ with factor λ
  Update the target networks: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$, $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  Create adaptive actor weights $\tilde{\mu}'$ from current actor weight θ^μ and current σ: $\theta^{\tilde{\mu}'} \leftarrow \theta^\mu + \mathcal{N}(0, \sigma)$
  Generate adaptive perturbed actions $\tilde{a}'$ for the sampled transition starting states $\tilde{s}_i$: $\tilde{a}' = \mu(\tilde{s}_i|\theta^{\tilde{\mu}'})$. With previously calculated actual actions $a = \mu(\tilde{s}_i|\theta^\mu)$, calculate the mean induced action noise: $d(\theta^\mu, \theta^{\tilde{\mu}'}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_s\left[(a_i - a_i')^2\right]}$
  Update σ: if $d(\theta^\mu, \theta^{\tilde{\mu}'}) \leq \delta$, $\sigma \leftarrow \alpha\sigma$, otherwise $\sigma \leftarrow \sigma/\alpha$
  Update perturbed actor: $\theta^{\tilde{\mu}} \leftarrow \theta^\mu + \mathcal{N}(0, \sigma)$
end for

E.1. Details on MMD measure and KS-statistics test

Maximum mean discrepancy (MMD) (Gretton et al., 2012) is a pseudometric over Prob($\mathcal{X}$), the space of probability measures on some compact metric set $\mathcal{X}$. Given a family of functions $\mathcal{F}$, MMD is defined as

$$\text{MMD}(\mathcal{F}, p_a, p_b) = \sup_{f\in\mathcal{F}} \mathbb{E}_{x\sim p_a}[f(x)] - \mathbb{E}_{x\sim p_b}[f(x)].$$

When $\mathcal{F}$ is the unit ball in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}$ associated with a *universal* kernel $K : \mathcal{X}\times\mathcal{X}\to\mathbb{R}$, i.e. $\{f\in\mathcal{H} : \|f\|_\infty \leq 1\}$, MMD$(\mathcal{F}, p_a, p_b)$ is not only a pseudometric but a proper metric as well, that is MMD$(\mathcal{F}, p_a, p_b) = 0$ if and only if $p_a = p_b$. One example of a universal kernel is the commonly used Gaussian radial basis function (RBF) kernel $K(x, y) = \exp\left(-\|x-y\|^2/(2\sigma^2)\right)$.

Given samples $\{x_i\}_{i=1}^M$, $\{y_i\}_{i=1}^N$ from p_a, p_b, the square of MMD has the following unbiased estimator:

$$\widehat{\text{MMD}}^2 = \frac{1}{\binom{M}{2}}\sum_{i=1}^{M}\sum_{j\neq i}^{M}K(x_i, x_j) - \frac{2}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}K(x_i, y_j) + \frac{1}{\binom{N}{2}}\sum_{i=1}^{N}\sum_{j\neq i}^{N}K(y_i, y_j).$$

We also have the biased estimator where the empirical estimate of feature space means is used

$$\widehat{\text{MMD}}_b^2 = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M}K(x_i, x_j) - \frac{2}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}K(x_i, y_j) + \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}K(y_i, y_j).$$

Note that the RBF kernel has the following representation

$$\begin{aligned}K(x, y) =& \exp\left(-\frac{\|x\|^2 + \|y\|^2}{2\sigma^2}\right)\exp\left(\frac{\langle x, y\rangle}{\sigma^2}\right)\\ =& C\sum_{n=0}^{\infty}\frac{\langle x, y\rangle^n}{\sigma^{2n}n!}\\ =& C\sum_{n=0}^{\infty}\frac{\langle \phi_n(x), \phi_n(y)\rangle}{\sigma^{2n}n!}\end{aligned}$$

where C is the constant $\exp\left(-||x||^2 + ||y||^2/(2\sigma^2)\right)$, and $\phi_n(\{\cdot)$ is the feature map of polynomial kernel of degree n. Following (Li et al., 2015), we can rewrite the *biased*

estimator $\widehat{\text{MMD}}_b^2$ with ϕ_n:

$$\begin{aligned}\widehat{\text{MMD}}_b^2 =& C\sum_{n=0}^{\infty}\frac{1}{\sigma^{2n}n!}\Bigg[\frac{1}{M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M}\langle\phi_n(x_i),\phi_n(x_j)\rangle \\ &- \frac{2}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}\langle\phi_n(x_i),\phi_n(y_j)\rangle \\ &+ \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\langle\phi_n(y_i),\phi_n(y_j)\rangle\Bigg] \\ =& C\sum_{n=0}^{\infty}\frac{1}{\sigma^{2n}n!}\left\|\frac{1}{M}\sum_{i=1}^{M}\phi_n(x_i)-\frac{1}{N}\sum_{j=1}^{N}\phi_n(y_j)\right\|^2.\end{aligned}$$

Therefore minimizing $\widehat{\text{MMD}}_b^2$ can be seen as matching all moments between the two distribution p_a, p_b.

Although we can do multivariate two-sample test with MMD, it suffers from the curse of dimensionality. Thus instead of looking at the distribution of percentage change vector $\mathbf{h}_i$, we perform two-sample test on the distribution p_h of percentage change series $h_{i,t}$ itself. Specifically, we use the Kolmogorov-Smirnov test, which is a two-sample test method based on $\|\cdot\|_\infty$ norms between the empirical cumulative distribution functions of the two distributions. For each asset of interest, we divide the data set into a training and a validation set, where the training set is used to train the RGAN model. Then we generate a batch of samples with the trained RGAN. For each generated series, we perform the KS test between it and every series in the validation set and calculates the maximum p-value from which. The average of these maximum p-values, denoted by $\bar{p}_{\max}$, among all the generated samples is then used to determine the goodness of fit of the given generated series. The average $\bar{p}_{\max}$ across all assets in our portfolio is 0.11, which shows that we cannot reject the null hypothesis that the generated distribution and the data distribution is the same.

Figure 3. Trading signals (last row is the S&P 500 index).

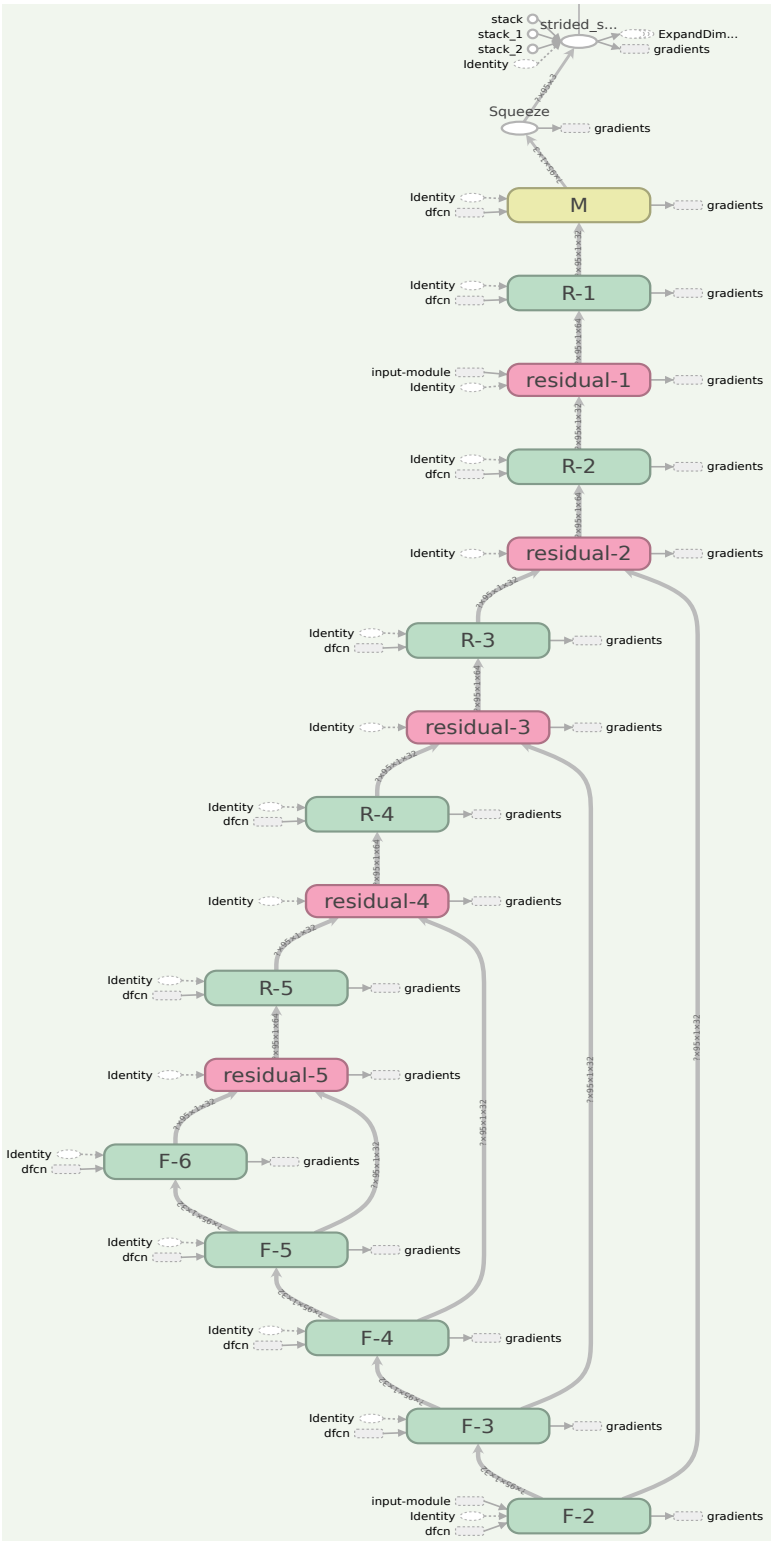


Figure 5. The network structure of WaveNet.