Reinforcement Learning vs 1/N and Mean-Variance Optimization In The Portfolio Allocation Problem

Thesis by
Matus Jan Lavko

In Partial Fulfillment of the Requirements for the degree of
Hons BSc Economics and Business Economics

UTRECHT UNIVERSITY
Utrecht, Netherlands

2021
Defended 23rd of June

ABSTRACT

In this paper, I test for the out-of-sample trade performance of state-of-the-art model-free reinforcement learning agents in order to compare them against the performance of an equally weighted portfolio and typical mean-variance optimization benchmarks. Motivated by the recent advances in this field and successful applications of the Reinforcement Learning paradigm in continuous state-action space, I develop a study within different contexts. By dividing the European and U.S. broad market index constituents into factor data sets using a Carhart four factor model, the models face different scenarios defined by these factor environments. Consequently, the RL approach is empirically evaluated based on a number of measures and probabilistic assessment. Training the models only on price data and features constructed from the prices, the performance of the RL approach yields better risk-adjusted as well as probabilistic portfolio as the Mean-Variance specifications, but ties with the equally weighted portfolio. The models are partially able to uncover the nonlinear structure of the SDF and the key drivers of the performance, and suggest that these models are useful in practical application as part of a larger trend of AI and big data applications in finance.

Keywords: Artificial Intelligence, Reinforcement Learning, Machine Learning, Portfolio Theory, Diversification, Financial Machine Learning

ACKNOWLEDGEMENTS

I would like to thank Prof. Thomas Walther for his supervision and helpful guidance throughout the process of conducting this study and writing the paper. In addition, I would like to extend my thanks to Mr. Pecher and Mr. Cegin for their valuable input in the conceptual stage of the work especially in the implementation and training phase. Further, I would like to thank Infinity Capital and Mr. Balint who has provided me the opportunity to intern at his company at the research department, where I had access to the Bloomberg Terminal and other resources that I was fortunate to use. In addition, I would like to thank all that have provided me with valuable feedback. Last but not least I would like to thank my girlfriend, my parents and all of those who are always there for me.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

Advances in computing and engineering bring about intriguing challenges in the realm of nonlinear modeling problems that are unfeasible to solve analytically and require more complex methods. The advent of reinforcement learning and stochastic control has been deployed with great success onto the board games problems, notably TD-Gammon (**Tesau1995**) or later ventures of the IBM DeepMind, where the models achieve a high level of winning rates against other programs as well as human professional players[1] (**Silver2016**). Growing digitization and automation in the asset management industry provides a natural ground for experiments with stochastic control because of suitable mathematical specifications of the problem. Numerous attempts to apply reinforcement learning have been conducted with promising results especially using direct reinforcement or inverse reinforcement methods (**Moody1998**), but until recently problems associated with training brittleness and high level of dimensionality did not allow for deep networks to be utilized during the process as the learning process proved often to be unstable.

In the model-free representation of the environment – in contrast to the model-based – there are two main approaches that allow for the use of deep learning, often combined in parallel. In Q-learning, an experience buffer storing transition states has improved training significantly (**Riedmiller**). In Policy Optimization methods, various parallelism techniques as in **Schulman2017** or other tweaks to the architecture[2] have been presented in order to address the issues described earlier. These algorithms that are on the forefront of research have proven to be useful for benchmark problems and given their promising results this provides a motivation to test them on a domain problem in finance – the Portfolio Allocation problem.

1.1 Motivation

Portfolio construction is an important problem in finance, with an objective to construct an optimally performing portfolio. Traditionally, it has been proposed that

[1]Such as the IBM Deep Blue playing Garry Kasparov, a world champion in 1996-1997. The symbolic win of Deep Blue in 1997 was a landmark victory for the computer over a champion player.

[2]Gradient clipping, Monte Carlo sampling or delayed networks are among many proposed ideas as will be described later.

this construction consists in two-steps, with a parametric approach to the estimation of moments and optimizing over a grid of feasible portfolios (**Markowitz1952**). It has been established that due to high dimensionality and short sample histories, the estimation is unstable (**Merton1980**), and various methods have been proposed to stabilize this process such as robust specifications with "uncertainty structures" (**Goldfarb2003**), Bayesian approaches (**Aguilar2000**) or incorporating equilibrium returns and views about asset returns (**Black1992**). These have proven to have ad hoc advantages but often fail due to estimation errors as described in (**Demiguel2005**).

The problem of nonlinear effects and interaction effects present in the structure of the cross-section of the returns result in ambiguity and a "zoo" of factors attempting to explain the difference in risk premia (**Harvey2016**). My work is hence tangent to the problem of asset pricing in the field of machine learning applications in the domain space. Recovering the Stochastic Discount Factor[3] for asset pricing models using nonlinear effects (**Gu2020**) or regularization approaches (**Kozak2020**), and further review of deep learning using an array of firm-specific and macro features under a no-arbitrage machine learning problem specification (**Chen2019**) show that the SDF is well approximated by linear factor models, although there are robustness advantages to using machine learning methods.

Although my goal is not retrieving the SDF explicitly, an application of this approach can be used for actively managing exposures and risks, found implicitly by the model and incorporated into the decision making within the RL framework with various constraints. This can provide benefits to algorithmic trading strategies complementing traditional tools. A similar effort in **Cong2020RL** based on a complex model-based multi-armed bandit RL system strives to make RL more interpretable for investing through considering logits as scoring metrics for allocation. In my paper, a different approach is taken based on a coarser price history used for training and no use of firm-specific or macro variables in order to conduct a benchmark study similar to the method described in **Demiguel2005**.

1.2 Problem Statement

Consider an environment $\mathcal{E}$ consisting of N securities that are contingent on a partially observed continuous state space[4] $\mathcal{S}$ of which $\mathbf{w}_{n,t+1} \subset \mathcal{S}$ are the weights

[3]SDF is defined as the transformation of a portfolio lying on the capital market line such that the expected excess return is zero. For more general definition see **danthine2014intermediate**.

[4]Assuming that a particular observation of state $\mathbf{s}_t$ does not describe the state of the environment as modelled by a multivariate stochastic process.

of a portfolio that constitute actions $\mathbf{a}_t$ with state $\mathbf{s}_t := \mathbf{p}_t$, where $\mathbf{p}_t$ is the respective price vector at time t.

Definition 1. *A portfolio in time t is a unique set of actions $\mathbf{a}_{t-1}$ and is determined by a reward $r(\boldsymbol{a}_t, \boldsymbol{s}_t)$ that we wish to maximize in $[t_0, T)$, where for each t a trade can take place such that:*

$$\mathbf{a}_{t-1} = \boldsymbol{w}_t = \begin{pmatrix} w_{1,t} \\ \vdots \\ w_{N,t} \end{pmatrix}, \quad \sum_{i=1}^{N} w_{i,t} = 1 \quad, w_{i,t} \in \mathbb{R}^+ \tag{1.1}$$

Corollary 1. *Given a portfolio $\mathbf{w}_t$, state $\mathbf{s}_t$ we postulate a simple arithmetic return in the form of $r_t = \frac{s_t}{s_{t-1}} - 1$, and derive the simple portfolio return and variance for a vector valued random variable:*

$$\boldsymbol{\mu} := \mathbb{E}[\mathbf{r}] \approx \mathbb{E}\begin{pmatrix} \mathbf{r}_1 \\ \vdots \\ \mathbf{r}_N \end{pmatrix} = \begin{pmatrix} \frac{1}{T}\sum_{t=1}^{T} r_{1,t} \\ \vdots \\ \frac{1}{T}\sum_{t=1}^{T} r_{N,t} \end{pmatrix} \tag{1.2}$$

$$\sigma^2 := \mathbb{E}[(\mathbf{r} - \mathbb{E}[\mathbf{r}])^2] \quad, \mathrm{Var}[\mathbf{r}_i] = \frac{1}{T-1}\sum_{i=1}^{T}(r_t - \mu_r)^2, \tag{1.3}$$

obtaining the return and volatility of a portfolio as:

$$\mu_p = \mathbf{w}^T \boldsymbol{\mu} \quad, \sigma_p^2 = \mathbf{w}^T \boldsymbol{\Sigma} \mathbf{w},$$

where $\boldsymbol{\Sigma}$ is an element-wise expansion of the left expression in **??**.

Having defined these estimated quantities[5], we would like to maximize the reward function across the discrete grid $t_0, \dots, T$ with distances of length h. We choose this reward to be the Sharpe Ratio as in **??**:

$$\mathbf{w} = \underset{\mathbf{w} \in \mathscr{G}}{\operatorname{argmax}} \sqrt{h} \frac{\mathbb{E}[\mathbf{r_{t,t+h}}]}{\sqrt{\mathrm{Var}[\mathbf{r}_{t,t+h}]}}, \tag{1.4}$$

where we satisfy the set of affine constraints $\mathscr{G}$ as defined in **meucci2009risk**.

[5]Note that there are many options one can take in order to pursue less variance in the estimates as outlined in **Elton**.

This problem is typically attacked with the use of quadratic programming, but we can exploit this formulation and rewrite the problem without the loss of generality such that it is broadly applicable to the reinforcement learning setting:

$$\max_{\Theta} \sum_{t=1}^{T} \mathbb{E}[\gamma r(\mathbf{a}_t, \mathbf{s}_t)], \tag{1.5}$$

where γ stands for a time-discounting factor and Θ the parametric expression of the nonlinear models used to estimate the value or policy functions depending on the architecture of the model used.

Chapter 2

BACKGROUND

2.1 Portfolio Allocation As An Optimization Problem

In a seminal work of **Markowitz1952** the problem of asset allocation is defined such that given a smooth utility function of an investor $U(w)$, the investor wishes to solve the following problem:

$$\begin{aligned} \min \quad & \mathbf{w}^T \Sigma \mathbf{w} \\ \text{s.t.} \quad & \mathbf{w}^T \boldsymbol{\mu} = \rho_p,\ \rho_p \in \mathbb{R}^+ \\ & \mathbf{w}^T \mathbf{1} = 1, \end{aligned} \tag{2.1}$$

for a target return ρ_p. It appears that this problem has a closed form solution under the constraint that Σ is a full-rank non-singular matrix, where we can form a Lagrangian[1]:

$$\mathcal{L}(\mathbf{w}, \lambda, \delta) = \frac{1}{2}\mathbf{w}^T \Sigma \mathbf{w} - \lambda(\mathbf{w}^T \boldsymbol{\mu} - \rho_p) - \delta(\boldsymbol{\mu}\mathbf{1} - 1).$$

with first order conditions:

$$\begin{aligned} \Sigma \mathbf{w} - \lambda \boldsymbol{\mu} - \delta \mathbf{1} &= 0 \\ \boldsymbol{\mu}\mathbf{1} &= \rho_p \\ \mathbf{w}^T \mathbf{1} &= 1 \end{aligned}$$

or equivalently:

$$\mathbf{w} = \lambda \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \delta \boldsymbol{\Sigma}^{-1} \mathbf{1}.$$

Using constraints from **??**, we can reparametrize the problem (**Bianchi2019**) as:

$$\begin{aligned} \boldsymbol{\mu}^T \mathbf{w} &= \lambda \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \delta \boldsymbol{\mu}^T \boldsymbol{\Sigma}^{-1} \mathbf{1} \\ \mathbf{1}^T \boldsymbol{\mu} &= \lambda \mathbf{1}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu} + \delta \mathbf{1}^T \boldsymbol{\Sigma}^{-1} \mathbf{1}, \end{aligned}$$

[1] note that we divide by 2 for convenience.

and after some algebra we arrive at a solution:

$$\mathbf{w}^* = \rho_p \underbrace{\frac{1}{G}(\mathbf{1}^T\boldsymbol{\Sigma}^{-1}\mathbf{1}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} - \mathbf{1}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\boldsymbol{\Sigma}^{-1}\mathbf{1})}_{\underset{n\times 1}{\vec{v}}} + \underbrace{\frac{1}{G}(\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\boldsymbol{\Sigma}^{-1}\mathbf{1} - \mathbf{1}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})}_{\underset{n\times 1}{\vec{s}}}, \tag{2.2}$$

where $G = (\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})(\mathbf{1}^T\boldsymbol{\Sigma}^{-1}\mathbf{1}) - (\mathbf{1}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu})^2$. This solution is of great significance to the argument since $\vec{v}$ and $\vec{s}$ are not based on changing a free parameter. Two important propositions follow:

Proposition 2. *Vectors $\vec{v}$ and $\vec{s}$ from* **??** *span the entire set of feasible mean-variance portfolios as their linear combination.*

Proof. Let q be a minimum variance portfolio resulting from the solution of **??**. If there exists an α such that $\alpha\vec{v} + (1-\alpha)(\vec{v}+\vec{s})$ the weights of q are well defined. Consider an $\alpha = 1-\rho_p$, then it follows that $(1-\rho_p)\vec{s} + \rho_p(\vec{s}+\vec{s}) = \vec{s} + \vec{v}\rho_p$ □

In order to define the mean variance frontier we only need to establish the second proposition, and will do so without a proof as it follows easily from the existence of α.

Proposition 3. *We can generate the set of feasible portfolios as affine combinations of any two distinct portfolios.*

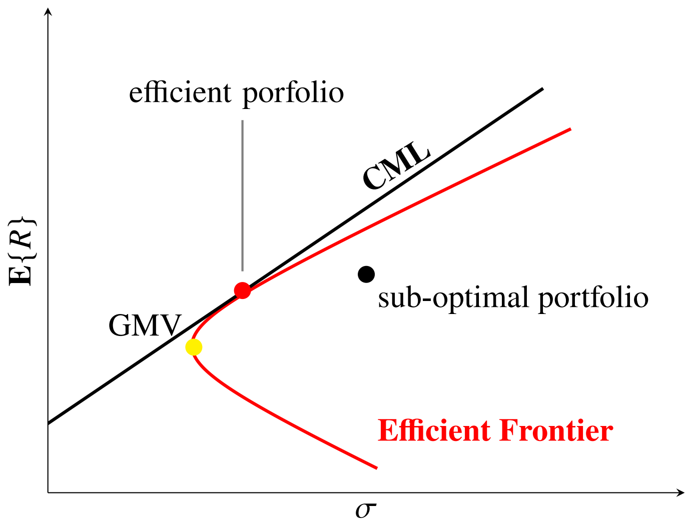


Figure 2.1: Mean-Variance Frontier

The efficient set of portfolios can be seen in Figure **??**, however only portfolios above the global minimum variance portfolio such as the efficient portfolio – defined by the intersection of the Capital Market Line[2] and the frontier – are interesting to our application since they give the maximum return for the least amount of risk.

Naturally, a question of more complex optimization objectives arises which yields different solutions and characteristic portfolios, and I refer the reader to appendix **??** for their descriptions as these specifications constitute the benchmarks in this study. These are all versions of quadratic programs with convex constrains due to the fact that they cannot be easily reduced to a set of linear equations and solved as we have demonstrated with **??**. One of the constraints is for example enforcing an element-wise constraint $\mathbf{w} \geq 0$ known as the long-only constraint. Commonly, a gradient-based method is used for this type of general conic problem such as the SQLSP solver (**kraft1988software**).

2.2 Reinforcement Learning Paradigm

Learning a pattern from data is traditionally a process that does not feature a feedback[3] from the system as in a typical supervised learning setting. Reinforcement learning tries to address this issue of a dynamic environment by internalizing a framework of states, actions and values that are endogenous and often feature a lot of noise. This allows for a more holistic view of the system to be developed as some environments tend to have highly nonlinear structures.

On Figure **??** we see a general setting of such a problem, where a sequence of states and rewards is passed on through the agent, and a policy drives a set of actions that are fed back into the environment. This interaction with the environment uses the Markov Decision Process[4] as a framework, establishing the features of an artificial intelligence problem (**Sutton2018**).

2.2.1 Markov Decision Process (MDP)

Definition 2. *A **Markov Decision Process** is a 5-tuple characterizing a dynamic decision problem of* $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ *where;*

[2]This is the line which under the assumption of a Capital Asset Pricing Model contains all possible portfolios. See **Sharpe1964**.

[3]We can define this feedback as a signal being sent back to the model in a circular fashion.

[4]Although we are arguably dealing with a Partially Observed MDP in this work, all of the definitions are applicable without the assurance of convergence in optimality. See chapter 9 in **Poole2017**

- *$\mathcal{S}$ is a countable set of states, where $s_t \in \mathcal{S} \colon \mathbb{P}[s_t \mid s_{t-1}] = \mathbb{P}[s_t \mid \bigcup_{j=1}^{t-1} s_j]$, satisfies the Markov Property.*
- *$\mathcal{A}$ is a countable set of actions.*
- *$\mathcal{P}$ is a transition matrix $\mathcal{P} \colon \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$. A transition probability $s \rightarrow s'$ is thus $\mathcal{P}_{s,s'} = \mathbb{P}[s' \mid s, a] = \sum_{r \in \mathcal{R}} \mathbb{P}[s', r \mid s, a]$.*
- *$\mathcal{R}$ is a reward generating function $\mathcal{R} \colon \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.*
- *γ is a scalar discounting the value of $r(s_t, a_t)$, while the agent maximizes $r(s_{t+1}, a_{t+1})$.*

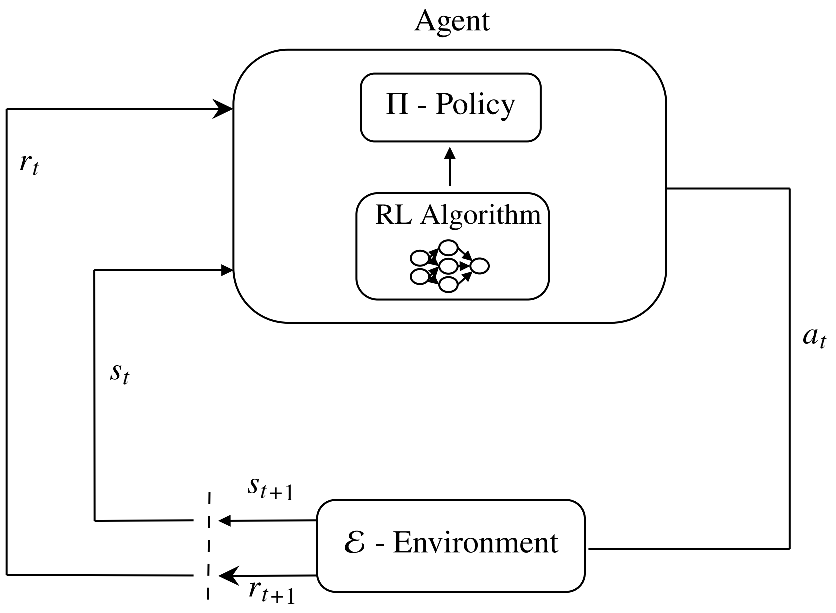


Figure 2.2: Reinforcement learning control flow

An agent typically features a mechanism that determines a policy π, according to a learned experience with the use of a model. This policy is then evaluated using a value function, or a Q-function in order to improve the iteration of the algorithm and learn the dynamics of the environment.

Definition 3. ***Policy*** *π is a mapping $\pi \colon \mathcal{S} \rightarrow \mathcal{A}$. An optimal[5] policy is such that given a value function $V^{\pi}(s)$, there is no π' such that $V^{\pi}(s) \leq V^{\pi'}(s)$,$\forall \pi' \nsim \pi$.*

[5]It can be proved that a stationary MDP always converges in the limiting scenario, see chapter 9 in **Poole2017**.

Definition 4. ***Value Function*** *is a function that determines the expected return from a state s under π.*

$$V^{\pi}(s) := \mathbb{E}_{\pi}[R_t \mid s_t] = \mathbb{E}_{\pi}\Big[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \mid s_t\Big],\ \forall s_t \in \mathcal{S} \tag{2.3}$$

Definition 5. ***Q-function*** *is a function that defines the value of taking action a_t, given a state s_t under a policy π.*

$$Q^{\pi}(s) := \mathbb{E}_{\pi}[R_t \mid s_t, a_t] = \mathbb{E}_{\pi}\Big[\sum_{j=0}^{\infty} \gamma^j r_{t+j+1} \mid s_t, a_t\Big] \tag{2.4}$$

Functions **??** and **??** are interconnected in a sense that they can be defined in terms of each other in a recursive manner (**Poole2017**). This fundamental relationship in reinforcement learning is known as the Bellman Equation and allows for a solution of the MDP through an approach involving a dynamic programming setup, where current and subsequent states share a relation that can be iterated upon to reach an optimal point.

Corollary 4. *From the previous two definitions, we can observe that R_t can be decomposed into $R_t = r_{t+1} + \gamma\Big[\sum_{j=1}^{\infty} \gamma^j r_{t+j+1}\Big]$. This is the current reward and sum of all subsequent rewards discounted accordingly. Hence, we can obtain the Bellman recursive relation for $V^{\pi}(s)$ as:*

$$V^{\pi}(s) = \mathbb{E}_{\pi}\Big[r_{t+1} + \gamma\Big(\sum_{j=1}^{\infty} \gamma^j r_{t+j+1}\Big) \mid s_t\Big]. \tag{2.5}$$

The expectation operator can be further expanded as:

$$\sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S},\, r \in \mathbb{R}} \mathbb{P}(s', r \mid s \cup a)\Big[r + \gamma V^{\pi}(s')\Big].$$

We can think about the expression above as the logic of the reinforcement learning system, where given a state s, all of the possible and attainable states s' return a reward r based on some dynamics given by the probability distribution we are summing upon. Thus, the value function provides a solution to Equation **??** and the ultimate goal becomes to learn $V^{\pi}(s)$, or $Q^{\pi}(a, s)$ [6].

[6]The reasoning we have developed can be applied to the Q-function in a similar fashion depending on the optimization objective of the algorithm.

2.2.2 Solution to the MDP

In order to solve an MDP we need to find a policy π^* that satisfies the relation $\pi^* \geq \pi'$ which is measured by the value functions and their respective sufficient relation for optimality $V^{\pi^*}(s) \geq V^{\pi}(s)$. I will not prove the existence of such policy, but a fact is assumed that there is always a policy that satisfies this inequality and is optimal. This gives rise to the optimization problem we face in the form of:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) \\ Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a). \end{aligned} \tag{2.6}$$

Notice that we can rewrite $Q^*(s,a)$ in terms of $V^*(s)$ as:

$$Q^*(s,a) = \mathbb{E}_{\pi}\Big[r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t, a_t)\Big]. \tag{2.7}$$

Therefore, we can arrive at a solution by maximizing the above expression such that an optimal policy is derived from the optimal value function as described in (**Poole2017**):

$$\begin{aligned} V^*(s) &= \max_{a} Q^*(s,a) \\ \pi^*(s) &= \operatorname*{argmax}_{a} Q^*(s,a) \end{aligned} \tag{2.8}$$

There are two general approaches that we are considering in the model-free setup that provide a numerical way of tackling the problem. Note that the algorithms used in the empirical experiment depend on these techniques to a smaller or larger extent and combine or extend the ideas described in this chapter.

Q-learning

One of the methods combining a dynamic programming approach and Monte Carlo sampling of expected values in the solution of **??** is Q-learning. It uses an off-policy method, a method that tries to improve the policy that is different from the policy used to generate the actions. Estimation of the reward for possible future actions is done without following any sort of greedy update – in contrast to on-policy methods. In order to understand the generic Q-learning algorithm we need to define the notion of a TD error:

Definition 6. ***TD Error*** *is an error resulting from an update in the estimated value of s and the more optimal estimate following from the recurrence relation* $r_{t+1} + \gamma V(s_{t+1})$*:*

$$\delta_t \doteq r_{t+1} + \gamma V(s_{t+1}) - V(s_t). \tag{2.9}$$

This error is interesting because as noted in chapter 6. of **Sutton2018** it results from a guess of another bootstrap. Although this might be a questionable approach it provides us with a way to describe more complex settings without a model of the environment which is advantageous in problems of high-dimensional state and action spaces such as the portfolio allocation problem.

In Q-learning the TD error is reformulated for the Q-function that tries to directly estimate the optimal Q-function $Q^*(s, a)$. It looks at the grid of values – also known as Q-table – chooses the best one and updates the Q-table with an estimate using the TD error. A general algorithm for solving the portfolio allocation problem is outlined in Figure **??**.

Figure 2.3 Q-learning algorithm for the portfolio allocation problem.

```
    Input: investment universe $\mathcal{E}$, price history $\mathbf{P}$, feature matrix $\mathbf{X}$
    Output $Q^*(s, a) \rightarrow w^*$
 1: procedure Q-LEARNING PROCEDURE
 2:     Initialize $Q(s, a)\ \forall\ s \in \mathcal{S}$ with $Q(T, \cdot) = 0,\ \alpha \in [0, 1]$
 3:     repeat
 4:         for $t = 0, \dots, T$ do
 5:             select an action $\mathbf{a'_t}$ based on a policy $a'_t = \max Q(s, a)$.
 6:             take action $\mathbf{a'_t}$ and observe $\{r, s'\}$
 7:             update the Q-table using ??:
 8:             $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_a Q(s', a) - Q(s, a)]$
 9:             if $T \mod h_{retrain} = 0$ then Update network parameters $\tilde{\theta} \leftarrow \hat{\theta}$
10:             end if
11:         end for
12:     until $T > T_{max}$ or terminal.
13: end procedure
```

Until now, we have not described a procedure that would feature a network that estimates the Q-function, however we can see that it is quite intuitive when the

Q-table grows very large and the updates become too costly. It would be as if there would be infinitely many directions to evaluate for each infinitesimal move of a player on the ice trying to reach the safe point.[7]

Policy Gradient

In contrast to learning a value function and an action-value estimate, Policy Gradient looks at learning a policy that is parametrized by a mapping $f(X) \rightarrow \boldsymbol{\theta} \colon \pi \sim \pi_\theta$ often in the form of an Artificial Neural Network or a Recurrent Neural Network **Silver2016**. Value function is thus not involved in the process of selecting an action, although it can still play a role in helping to estimate $\pi(a \mid, s, \boldsymbol{\theta})$. In order to estimate this function one needs to perform a stochastic update with regard to the gradients of an error typically in the form of:

$$\mathcal{J}(\boldsymbol{\theta}) \colon \boldsymbol{\theta_{t+1}} \leftarrow \boldsymbol{\theta}_t + \alpha \nabla \mathcal{J}(\boldsymbol{\theta}_t). \tag{2.10}$$

In the process of solving the optimization objective to learn $\boldsymbol{\theta}$, the gradient of $\mathcal{J}(\boldsymbol{\theta})$ is calculated as:

$$\begin{aligned} \mathcal{J}(\boldsymbol{\pi_\theta}) &= \mathbb{E}_{\pi_\theta}[\mathcal{R}] \\ \mathcal{J}(\boldsymbol{\pi_\theta}) &= \int_{\mathbf{s} \in \mathcal{S}} P^{\pi_\theta}(\mathbf{s})\, ds \int_{a \in \mathcal{A}} \pi_\theta(\mathbf{s}, \mathbf{a}) \mathcal{R}\, da, \quad \text{(by definition of expectation)} \end{aligned} \tag{2.11}$$

now taking the gradient of $\boldsymbol{\theta}$:

$$\begin{aligned} \nabla \mathcal{J}(\boldsymbol{\pi_\theta}) &= \int_{\mathbf{s} \in \mathcal{S}} P^{\pi_\theta}(\mathbf{s})\, ds \int_{a \in \mathcal{A}} \nabla \pi_\theta(\mathbf{s}, \mathbf{a}) \mathcal{R}\, da \qquad (2.12) \\ &= \int_{\mathbf{s} \in \mathcal{S}} P^{\pi_\theta}(\mathbf{s})\, ds \int_{a \in \mathcal{A}} \pi_\theta(\mathbf{s}, \mathbf{a}) \frac{\nabla \pi_\theta(\mathbf{s}, \mathbf{a})}{\pi_\theta(\mathbf{s}, \mathbf{a})} \mathcal{R}\, da \qquad \text{(multiply by 1)} \\ &= \int_{\mathbf{s} \in \mathcal{S}} P^{\pi_\theta}(\mathbf{s})\, ds \int_{a \in \mathcal{A}} \pi_\theta(\mathbf{s}, \mathbf{a}) \nabla \ln[\pi_\theta(\mathbf{s}, \mathbf{a})] \mathcal{R}\, da. \quad (\text{by } \nabla \ln x = \frac{\nabla x}{x}) \end{aligned}$$

[7]This is a reference to the FrozenLake-v0 gym that features a player trying to reach a point on ice that melts after they cross it.

This important result allows us to derive the policy gradient that we want to optimize as described in **Mandic2019**. We will state the theorem without proof that is treated extensively in **Sutton2018** or **Poole2017**.

Theorem 5. *From the result in* **??**, *extending the one-step MDP to a multi-step MDP, for a suitable continuous differentiable policy $\pi(s, a)$ and the expectation of future rewards at each step $\mathcal{J}(\boldsymbol{\theta})$ we obtain a* ***Policy Gradient*** *of the form:*

$$\nabla \mathcal{J}(\boldsymbol{\pi_\theta}) = \mathbb{E}_{\pi_\theta}\Big[\nabla \ln[\pi_\theta(\mathbf{s}, \mathbf{a})] Q^\pi(\mathbf{s}, \mathbf{a})\Big]. \tag{2.13}$$

There are potential benefits from optimizing the policy gradient instead of the typical value function approach as it has been shown to converge more easily in some cases. For a comprehensive treatise of this topic see **Necchi2016**. A general setup of a Policy Gradient algorithm with an actor and critic network is proposed as follows:

Figure 2.4 Policy Gradient algorithm for the portfolio allocation problem.

```
    Input: investment universe $\mathcal{E}$, price history $\mathbf{P}$, feature matrix $\mathbf{X}$
    Output $\pi^*(s, a) = w^*$
 1: procedure Policy Gradient update
 2:     Initialize cache and parameters $\{C\}$, $\boldsymbol{\theta}_0$, $\alpha \in [0, 1]$
 3:     repeat
 4:         for $t = 0, \dots, T$ do
 5:             observe $\{\mathbf{s}_t, \mathbf{r}_t\}$
 6:             obtain action (by sampling from MDP) or $\pi$
 7:             update actor network parameters $\theta \longleftarrow \theta + \alpha Q(\mathbf{s}, \mathbf{a}) \nabla_\theta \ln \pi(\mathbf{a}, \mathbf{s})$
 8:             cache rewards and gradients in $\{C\}$
 9:             update critic network with TD error (if present)
10:         end for
11:     until convergence.
12: end procedure
```

Note that simpler versions of the algorithm without TD error updates exist such as REINFORCE or other episodic Monte Carlo methods as described in chapter 13. **Sutton2018**, but due to problems with sequential data[8] I am not considering their application for our use case.

2.3 Model-Free Algorithms in Continuous State-Action Space

In this section I will briefly describe the workings of the algorithms used for the benchmark study. All of them can be classified as model-free due to their ability

[8]These algorithms are not well equipped to deal with non-iid sampling of the episodes that we use for learning.

to learn the dynamics of $\mathcal{E}$ without prior model on the transition matrix $\mathcal{P}$. In our use case we can think of it as the algorithm learning the hidden factors explaining the excess returns similar to PCA using a nonlinear base and the RL feedback loop. We choose these algorithms, because of their ability to learn in a setting with limited information in contrast to linear models that often suffer from sampling issues (**Aue2013**) and structural breaks (**Andreou2002**).

2.3.1 Actor Critic (A2C)

Actor Critic or A2C is a class of methods that address a problem with estimating the baseline function $b_t(s_t)$ for $r_{t+1} \mid s_t$ and stabilize the training of the algorithm. During the update on line 7 in algorithm **??** an iterative step $\alpha Q(\mathbf{s}, \mathbf{a}) \nabla_\theta \ln \pi(\mathbf{a}, \mathbf{s})$ is taken in order to estimate $\nabla_\theta \mathbb{E}[R_t \mid s_t, a_t]$. It has been proposed in **Williams1992** that the variance of this estimate can be reduced without introducing any bias by subtracting a baseline function estimated by the value function. Therefore the update becomes:

$$\begin{aligned} \alpha(Q(\mathbf{s}, \mathbf{a}) - b_t(s_t)) \nabla_\theta \ln \pi(\mathbf{a}, \mathbf{s}) \\ b_t(s_t) \approx V^\pi(s) \end{aligned} \tag{2.14}$$

The expression $(Q(\mathbf{s}, \mathbf{a}) - b_t(s_t))$ of Equation **??** is commonly known as the advantage, where the baseline is known as the critic and the policy function as the actor (**Mnih2016**). The idea as illustrated in Figure **??** is to estimate transitions of $\mathcal{P}$ not only at the first stage of the transition but also involving subsequent states.[9]

2.3.2 Deep Deterministic Policy Gradient (DDPG)

Instead of modelling $\pi(\cdot \mid \mathbf{s}, \theta)$ as a probability distribution, a deterministic decision function $\rho\colon \mu^\rho(\mathbf{s} \mid \theta) \to [0, 1]$ is postulated, mapping states to actions in a one-to-one relation. From the result in **??** we can redefine the loss function as:

$$\mathcal{J}(\boldsymbol{\theta}) = \int_{s \in \mathcal{S}} \tilde{\mu}(\mathbf{s}) Q(\mathbf{s}, \mu(s)) \, ds \tag{2.15}$$

[9] For more detailed treatise on this multi-episodic approach and proof of the Policy Gradient theorem in this setting see chapter 13.5-13.6 in **Sutton2018**.

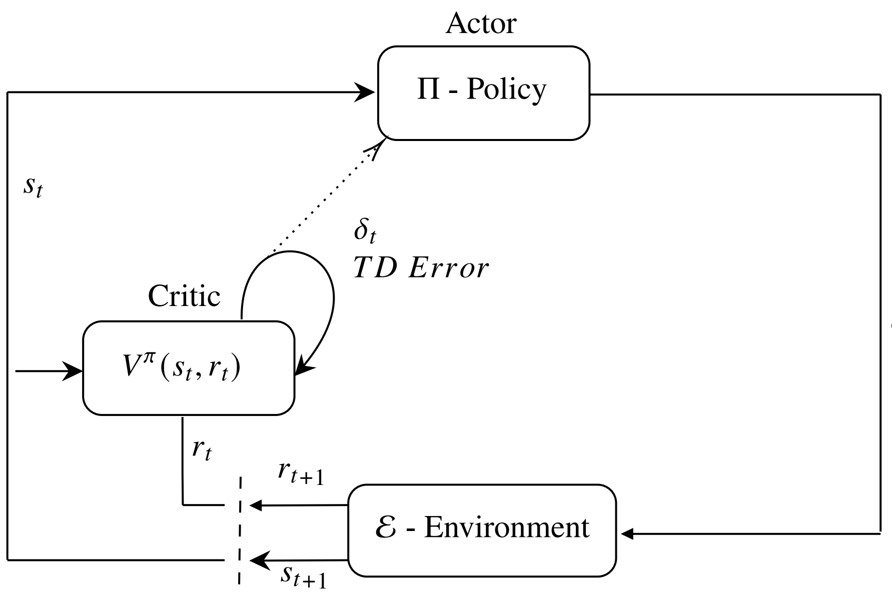


Figure 2.5: On-policy actor critic control flow

where $\tilde{\mu}(\mathbf{s}) = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \mu_0(s)\mu(s', k)\ ds$ is a discounted distribution defined recursively as in **Weng2018**. Hence, a gradient can be taken and an update performed as:

$$\begin{aligned} \nabla \mathcal{J}(\boldsymbol{\theta}) &= \int_{s \in \mathcal{S}} \tilde{\mu}(\mathbf{s}) \nabla_a Q(\mathbf{s}, \mu(s)) \nabla_\theta \mu(s)\ ds \qquad \text{(by chain rule)} \qquad (2.16) \\ \boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha \nabla_a Q(s_t, a_t) \nabla_\theta \mu(\mathbf{s}). \end{aligned}$$

It is apparent that a deterministic policy likely does not introduce enough exploration[10] in the algorithm and DDPG tries to address this by inducing exploration with a noise process[11] $\mathcal{N}$ to the deterministic policy such that:

$$\dot{\mu}(s) = \mu(s) + \mathcal{N}. \qquad (2.17)$$

DDPG is usually learned in mini-batches[12] using an experience replay to obtain an approximately iid sample. It also performs a soft-update in order to stabilize the learning of the actor and critic networks (**Mnih2013**).

[10]In other words that it does not estimate the Q-function well enough due to insufficient noise and converges to a locally optimal specification. See chapter 11.3 in **Poole2017**.

[11]Often in the form of a generic Ornstein-Uhlenbeck process x_t: $dx_t = \theta(\mu - x_t)dt + \sigma dB_t$, where B_t is a standard Brownian motion.

[12]By splitting the learning data and computing gradient on a subset to optimize the learning during gradient descent optimization.

A general outline for the DDPG algorithm as presented in **Lillicrap2016** can be summarized as follows:

Figure 2.6 DDPG algorithm

```
    Initialize actor and critic networks $Q(s, a \mid \theta),\ \mu(s \mid \theta_\mu)$ with random weights
    along with target networks $Q', \mu', \theta_Q, , \theta_\mu$ respectively.
    Initialize experience replay cache $C$
 1: procedure
 2:     for $t = 0, \ldots, S$ do
 3:         Initialize a noise process $\mathcal{N}$, Observe $s_0$
 4:         for $t = 0, \ldots, T$ do
 5:             select action $a_t = \mu(s_t) + \mathcal{N}$
 6:             observe $r_t \mid s_t$ and $s_{t+1}$ and store the array in the replay cache
 7:             sample from the cache to obtain a minibatch $(s_i, a_i, r_i, s_{i+1}) \overset{iid}{\sim} \{C\}$
 8:             update critic by minimizing:
                    $L = \frac{1}{N} \sum_i (r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}) \mid \theta^{Q'})) - Q(s_i, a_i))^2$
 9:             update actor using the sampled policy gradient:
                    $\nabla_{\theta^\mu} \approx \frac{1}{N} \sum_i \nabla_a Q(s, a) \nabla_{\theta^\mu} \mu(s)$
10:             perform a soft-update of weights for:
                    $Q',\ \mu' \colon \theta^{target} \leftarrow \tau\theta^{net} + (1 - \tau)\theta^{target}$
11:         end for
12:     end for
13: end procedure
```

2.3.3 Proximal Policy Optimization (PPO)

Given the difficulty with the TD error approach, sometimes the gradient ascent algorithm results in very large policy updates that can destabilize the optimization process and problems in convergence can arise. To remedy for this, an additional constraint is proposed to stabilize this learning. From Equation **??** and **??** we observe that a typical actor critic maximizes:

$$\mathcal{J}(\theta) = \mathbb{E}[\nabla_\theta \ln \pi(a_t)(Q(s_t, a_t) - b_t(s_t))]. \tag{2.18}$$

The so-called Trust Region methods such as TRPO redefine the surrogate objective function as a ratio of the current policy function function and the old policy function (**Schulman2017**) and introduce a Kullback-Leibler divergence constraint[13] such that the maximization problem is rewritten as:

$$\begin{aligned} \max \quad & \mathbb{E}\Big[\frac{\pi(a_t)}{\pi_{old}(a_t)}(Q(s_t, a_t) - \hat{b}_t(s_t)\Big] \\ \text{s.t.} \quad & \mathbb{E}[D_{KL}(\pi_{old} \parallel \pi)] \leq \eta. \end{aligned} \tag{2.19}$$

PPO is an improvement over TRPO that addresses the relative difficulty of training this sort of specification by introducing a clipped surrogate objective function without the KL constraint in the form of:

$$\begin{aligned} \mathcal{U}^{original} &= r(\theta)\big[Q(s_t, a_t) - \hat{b}_t(s_t)\big] \\ \mathcal{U}^{clipped} &= \text{clip}\big(r(\theta), 1-\epsilon, 1+\epsilon)\big)\big[Q(s_t, a_t) - \hat{b}_t(s_t)\big] \\ \mathcal{J}^{clipped}(\theta) &= \mathbb{E}\Big[\min\big\{\mathcal{U}^{original}, \mathcal{U}^{clipped}\big\}\Big], \end{aligned} \tag{2.20}$$

where $r(\theta) = \frac{\pi(a_t)}{\pi_{old}(a_t)}$. The first term in the expectation in Equation **??** is simply the same expression as in **??**, and the clip function bounds the policy updates within a interval, where ϵ is a hyperparameter. Finally a lower bound of this expression is taken to ensure a conservative update (**Schulman2017**). Optimizing this loss ensures that objective function is more stable and the surrogate objective does not deviate too much when $r(\theta)$ becomes large.

2.3.4 Soft Actor Critic (SAC)

Soft Actor Critic features a model that encourages more exploration in order to ensure more efficient and stable update to π_θ by reformulating the optimization problem in terms of maximum entropy (**Haarnoja2018**). It is an off-policy algorithm[14] as illustrated in **??**, as the agent performs updates based on some sort of a greedy policy

[13]This is a measure of relative entropy between two distributions defined as $D_{KL}(Q \parallel G) = \int_{\mathbb{R}} P(x)\ln(\frac{Q(x)}{G(x)})dx$ for distributions defined on the same probability space.

[14]Note that SAC, DDPG and TD3 are off-policy algorithms – although an asynchronous version of A2C can be formulated off-policy. PPO and A2C are on-policy algorithms so they are learning the value function for one policy as they follow it online.

in contrast to on-policy method, where TD error is directly connected to the agent's current policy as in **??**.

Consider the Equation **??**, where we define the expected sum of rewards as the objective we wish to maximize through gradient ascent. In SAC this objective is generalized to:

$$\mathcal{J}(\pi) = \mathbb{E}[r(s_t, a_t) + \phi\mathcal{H}(\pi(\cdot \mid s_t))], \tag{2.21}$$

where ϕ determines the importance of entropy[15] $\mathcal{H}$ against the reward function in the optimization problem by allowing more noise as it gets bigger.

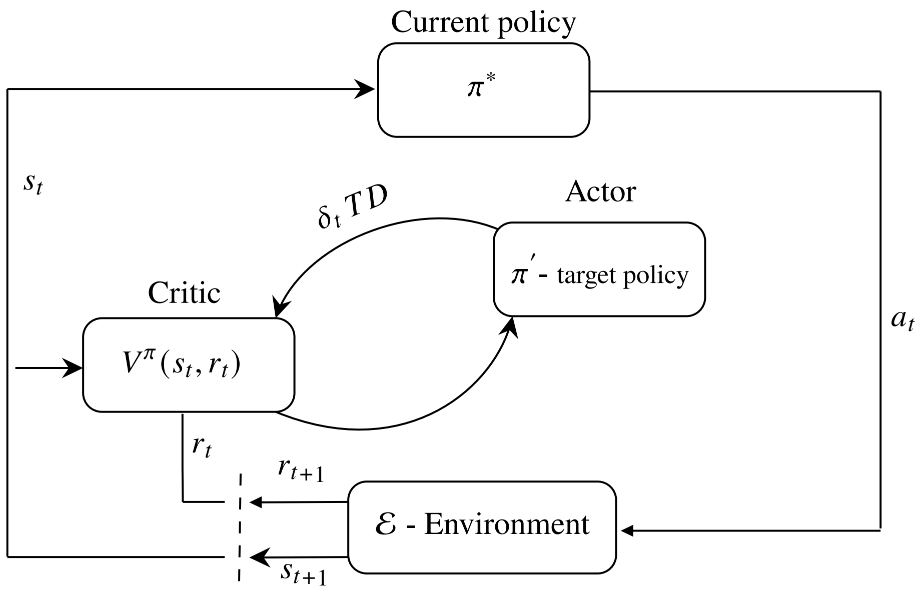


Figure 2.7: Off-policy actor critic control flow

In the soft update, the Bellman recursive relation is defined in order to satisfy the Equation **??** as:

$$\begin{aligned} Q(\mathbf{s}_t, \mathbf{a}_t) &= r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\mathbb{E}[V(\mathbf{s}_{t+1})] \\ V(\mathbf{s}_{t+1}) &= \mathbb{E}_\pi[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \phi \ln \pi(\mathbf{a}_{t+1})]. \end{aligned} \tag{2.22}$$

Similarly to the actor critic setup, two networks are trained corresponding to the value function and the Q-function with the use of stochastic gradients as derived in **Haarnoja2018**:

[15]As defined in (**Goodfellow2016**) to be the negative expectation of the log probability distribution.

$$\mathcal{J}_{V(\mathbf{s_t})}(\theta^V) = \mathbb{E}\Big[\frac{1}{2}\big(V_{\theta^V}(\mathbf{s}_t) - \mathbb{E}[Q(\mathbf{s}_t, \mathbf{a}_t) \ln \pi_{\theta^V}(a_t)]\big)^2\Big]$$
$$\mathcal{J}_{Q(\mathbf{s}_t,\mathbf{a_t})}(\theta^Q) = \mathbb{E}\Big[\frac{1}{2}\big(Q(\mathbf{s}_t, \mathbf{a}_t) - (r(\mathbf{s_t}, \mathbf{a}_t) + \gamma \mathbb{E}_{\mu(\mathbf{s})}[V_{\tilde{\theta}^V}(\mathbf{s}_{t+1})])\big)^2\Big], \tag{2.23}$$

where the outer expectation is taken under states and actions sampled from an experience replay, and the inner expectation in the soft Bellman residual $\mathcal{J}_{Q(\mathbf{s}_t,\mathbf{a_t})}(\theta^Q)$ under the deterministic marginal $\mu(\mathbf{s})$ as in **??**. In addition, a target networks trick is used to stabilize learning by taking updates based on an exponential moving average of the estimated value function[16] as denoted by $V_{\tilde{\theta}^V}$.

Finally, a policy is updated using an optimization step based on the distance between the new policy function and a transformed Q-function as:

$$\pi = \operatorname*{argmin}_{\pi} D_{KL}\Big[\pi(\cdot \mid \mathbf{s}_t) \Big\| \frac{exp(Q^{old}(\mathbf{s}_t, \cdot)}{Z^{old}(\mathbf{s}_t)}\Big], \tag{2.24}$$

where $Z^{old}(\mathbf{s}_t)$ constitutes a partition function that normalizes the distribution.[17]

SAC combines the advantages from DDPG, Actor Critic and maximum entropy frameworks to achieve more consistent performance, outperforming PPO and DDPG on benchmark problems (**Haarnoja2018**). Although its multi-stage optimization process is arguably more complex, it is suggested that the soft policy approach can lead to better asymptotic performance.

2.3.5 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed DDPG algorithm is another modification of the Q-learning algorithm with the use of deep networks for function approximation in a deterministic setting. Because of the large variance in the estimates of the Q-function and the resulting overestimation (**Fujimoto2018**), a couple of tricks are proposed in order to stabilize the learning and restrict the propagation of the error resulting from this bias in the greedy selection of the actions in the Bellman recursion.

TD3 uses two networks for selecting an action and estimating the Q-value, where the target network is used for estimation and the current network for action selection.

[16]This trick has been explored as a way to improve stability in the Deep Q-learning setup, see **Wang2015**.

[17]The notion of partition function and Boltzmann distribution are concepts from information theory, see chapter 3 **mackay2003**

These networks according to **Fujimoto2018** are not necessarily independent due to slow updates and so the decoupling is altered by introducing an upper bound of the less biased estimate by the more biased one as follows:

$$L_{critic} = \frac{1}{N}\sum_i (r_i + \gamma \min_{\omega} Q(s_{i+1}, \mu_1(s_{i+1})) - Q(s_i, a_i))^2, \tag{2.25}$$

where ω is the set of weights learned by the critic networks. Given this loss, the target and current networks are updated using a soft delayed update similar to the SAC in order to stabilize the learning (**Dankwa2019**).

Furthermore, a regularizing method for training the critic network is introduced with the idea of forcing similar actions to have similar values and smooth the value estimate as in **Fujimoto2018**. Hence a clipped noise term is added to the sampled action from an experience replay as described in step 7 of the TD3 algorithm presented in **??**.

Figure 2.8 TD3 algorithm

```
    Initialize actor and critic networks $Q(s, a \mid \theta),\ \mu(s \mid \theta_\mu)$ with random weights
    along with target networks $Q',\ \mu'\ \theta_Q, \theta_\mu$ respectively
    initialize experience replay cache $C$
 1: procedure
 2:     for $t = 0, \dots, T$ do
 3:         initialize a noise process $\mathcal{N}$, Observe $s_0$
 4:         select action $a_t = \mu(s_t) + \mathcal{N}$
 5:         observe $r_t \mid s_t$ and $s_{t+1}$ and store the array in $C$
 6:         sample from the cache to obtain a minibatch $(s_i, a_i, r_i, s_{i+1}) \overset{iid}{\sim} \{C\}$
 7:         smooth the target policy $\tilde{a}_t = \pi(s)_t + \epsilon,\ \epsilon = \text{clip}(\mathcal{N}(0, \sigma), -c, c)$, where
              $c$ is in the neighbourhood of $a_t$
 8:         update the critic nets according to the loss in ?? using $\tilde{a}_t$
 9:         if $t$ mod $h_{retrain}$ then
10:             update actor weights using the sampled policy gradient:

                        $\nabla_{\theta^\mu} \approx \frac{1}{N}\sum_i \nabla_a Q(s, a) \nabla_{\theta^\mu} \mu(s)$
11:             update target actor and critic networks with a soft update
12:         end if
13:     end for
14: end procedure
```

Chapter 3

EMPIRICAL EXPERIMENT

In this chapter I describe an empirical experiment conducted in order to benchmark the performance of the model-free architectures described in the previous chapter against the equiweight and mean-variance approaches to portfolio construction.[1]

3.1 Data

I collect raw daily open high low close price and volume data for all equities in the S&P 500 index and its European counterpart assembled by Bloomberg – Bloomberg 500 index (**Bloomberg**). In addition I collect a sector dataset according to S&P sector classification, assembled from 9 SPDR sector exchange traded funds (**SPDR**) and two iShares ETF's namely the Telecommunications (**TelETF**) and Real Estate (**RealEsETF**). Due to computational constraints and a hypothesis of a change in the market regime, the data set is restricted to start from 15th of October 2008 until 1st of January 2021 for a total of 2974 observations.[2] In order to prevent survivorship bias and ensure completeness of the sample securities that do not have at least 2 years before t_0 are purged, resulting in a dataset of 479 securities listed in Europe and 487 U.S. listed stocks. For estimating the factor model I use factors constructed by Bloomberg, aggregating by monthly return on these factors that are constructed using the methodology in **fama1992cross**. These are the suitably defined risk premia across the section of the assets in my factor model. In addition, a risk-free proxy is used for computing performance metrics and excess returns as the 10-year rate on U.S. Treasury bills.

3.2 Methodology

Training on the whole index is computationally non-feasible for such a large set of securities and is not of interest since I am not considering such large portfolios. To

[1] All of the code without data and reproducible examples can be found at https://github.com/matus-jan-lavko/ReinforcementLearning-vs-EW

[2] Note that this includes the estimation period for the sample covariance matrix which is used as a feature during the training, effectively pushing the training by 252 trading days starting on the 2nd of February 2009

divide the data sets into clusters I take an approach based on a linear time-series factor model that is specified as follows:

$$\mathbf{E}[R_{t,i}] - R_t^f = \alpha_{t,i} + \sum_{j=1}^{n_f} \underbrace{\beta_i(\mathbf{E}[F_{t,j}] - R_t^f)}_{\text{systematic risk}} + \underbrace{\epsilon_{t,i}}_{\text{intrinsic risk}}, \tag{3.1}$$

where it is assumed that intrinsic risks can be hedged such that $\mathbf{E}[\epsilon_{t,i}] = 0$ for a model with j number of factors across i assets such that the no-arbitrage condition holds[3]. For the purpose of the experiment, this rather strict assumption can be relaxed as it is not an exercise in retrieving the SDF, but it is good practice to keep it in mind given the fact that it affects the form of our training data sets.

Following **Carhart1997**, I postulate the existence of a four factor asset pricing model, where a filtration $\mathcal{F}: F_j \in \{MKT_t, SMB_t, HML_t, MOM_t\}$ is the set of risk premia describing the investment universe $\mathcal{E}$. These factors can be described as:

- **MKT**$_t$: Broad market index returns.
- **SMB**$_t$: Small Minus Big, based on a difference between proxy portfolios of small companies and big companies formed on size and book to market ratio or other value indicator.
- **HML**$_t$: High Minus Low, based on a difference between value and a growth portfolios formed in the same way.
- **MOM**$_t$: Momentum, formed as an equal-weighted portfolio of the difference in the best performing securities in terms of lagged simple returns and the worst performing ones.

Assuming that the Generalized Gauss-Markov theorem holds true, we can retrieve $\hat{\beta}_{i,j}$ using a least-squares estimate (**Kempthorne2013**).[4] Given the possibility of substitution effects among the factors I employ a shrinkage method that has the advantages in reducing estimation errors out of sample as in **Kozak2020**. Although in such a low-dimensional model this could reduce performance of the SDF as a pricing kernel, this approach is taken in order to improve the stability of the estimates

[3]This lemma implies an existence of an SDF which is the main object of interest in the asset pricing literature and is the building block of no-arbitrage pricing models (**danthine2014intermediate**).

[4]Note that under the normality assumption on the distribution of residuals it can be proven that this is also the MLE estimator. See chapter 14 in **rice2006mathematical** for proof.

since the parameters will be later used for sorting. On a span of 36 months I estimate a linear model using the L_1 and L_2 penalties[5] and solve the optimization problem in the unconstrained Lagrangian form:

$$(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}) = \underset{a,b}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{y}_i - a\mathbf{1} - \mathbf{F}b)\|_2^2 + \lambda_1 \sum_{j=1}^{n_f} \|b_j\|_1 + \lambda_2 \sum_{j=1}^{n_f} \|b_j\|_2^2$$

After obtaining the estimates, these data sets are constructed using quantile sorting of the coefficients $\hat{\beta}_j$. This β-quantile is defined as:

$$Q_{\boldsymbol{\beta}}(p) := \inf\{\beta \in \mathbb{R} \colon p \le F(x)\},$$

where $F_{\boldsymbol{\beta}}(X) = \mathbb{P}(\beta \le x)$ is the empirical CDF obtained from the factor model.[6] I summarize the resulting sorted data sets in Table **??** together with their corresponding quantiles.

Dataset	Quantile Range	N
Growth 50	$\{Q_{\boldsymbol{\beta}^{\text{HML}}}(0.1)\}$	50
Momentum 50	$\{Q_{\boldsymbol{\beta}^{\text{MOM}}}(0.9)\}$	50
Size 50	$\{Q_{\boldsymbol{\beta}^{\text{SMB}}}(0.1)\}$	50
Growth + Size 100	$\{Q_{\boldsymbol{\beta}^{\text{HML}}}(0.1)\} \cup \{Q_{\boldsymbol{\beta}^{\text{SMB}}}(0.1)\}$	100
Momentum + Size 100	$\{Q_{\boldsymbol{\beta}^{\text{MOM}}}(0.9)\} \cup \{Q_{\boldsymbol{\beta}^{\text{SMB}}}(0.1)\}$	100
Sector ETF	N/A	11

Table 3.1: Beta quantile-sorted data sets

I obtain these sorted portfolios for both the S&P 500 and BE500 indices, resulting in 10 quantile sorted data sets plus the sector ETF data set. This provides for means to test the algorithms on both American and European equities with a single-factor and multi-factor approach, where the idea is to test if the model is able to capture hedging opportunities within different factors – or sectors in the sector ETF data set.

3.2.1 Model Training

Training neural networks is often a challenging task given the specifics of stochastic gradient descent optimization techniques. It is also quite computationally expensive

[5] Where we use the convention of a norm being defined as a notion of distance in the L^p space for a real valued vector $\mathbf{x}$: $\|\mathbf{x}\|_\mathbf{p} := (\sum_\mathbf{j} |\mathbf{x_j}|^\mathbf{p})^{\frac{1}{\mathbf{p}}}$

[6] Note that the definition of the quantile does not require any smoothness or continuity properties to be required of $F(x)$.

which is why I have decided to train the models using Google's cloud service Colab Pro that allowed me to parallelize training on a GPU unit Tesla p100-PCIE-16GB. For the models themselves I have used a high-level package that is built on PyTorch (**Liu2020**) and allowed me to use the Stable Baselines (**stable-baselines**) that provides stable implementations of all state of the art algorithms based on OpenAI infrastructure.

A softer grid search has been employed on all of the models in order to search for optimal hyper-parameters as summarized in appendix **??**. This has been done in order to ensure that a stable configuration of the model is deployed on the data given the difference in the underlying data processes of our data sets.

The grid search as illustrated by **??** initializes an evenly spaced grid of hyper-parameters which differs by the model architecture. Most commonly the learning rate α and the discount factor γ are optimized with model-specific parameters such as batch size or entropy coefficient in SAC. Note that hyperparameter $h_p \in \{\mathcal{H}\}$ is increasing the dimensions of the grid resulting in $O(n^{dim(\mathcal{H})})$ complexity and thus, it is computationally quite expensive to optimize hyperparameters this way as there is $k \times n \times (h_1 \times \ldots \times h_p)$ number of models to train and evaluate.[7] It is possible to initialize the grid randomly, but since I am interested in the magnitude of parameters more than a precise configuration, the even spaced approach is taken. For example optimizing the parameter α typically takes into consideration an array of possible parameters $(\ldots, 1 \times 10^{-4}, 1 \times 10^{-3}, 1 \times 10^{-2})$.

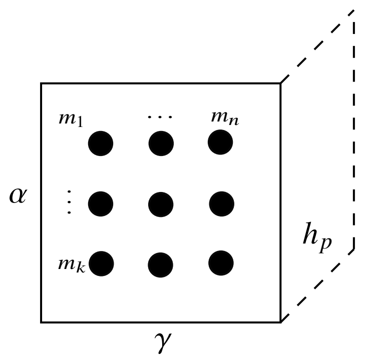
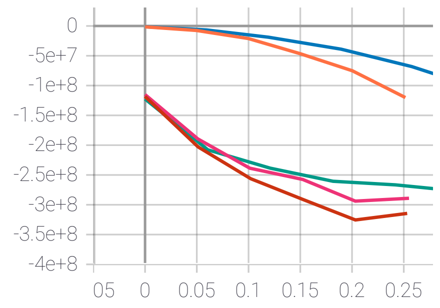
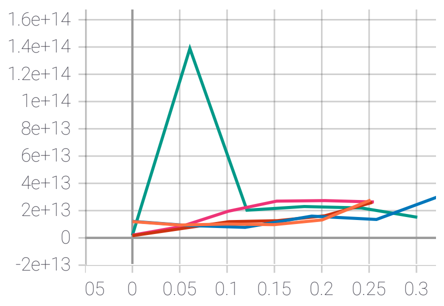


Figure 3.1: Grid search procedure

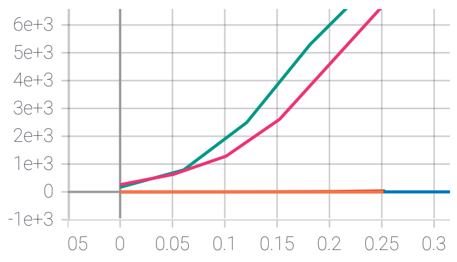
[7]I have trained approximately 4-12 configurations in the grid for each of the 11 datasets – depending on training complexity – of which a full soft search has taken approximately two days for each of the datasets resulting in a total of 24 training days for the search to be conducted.



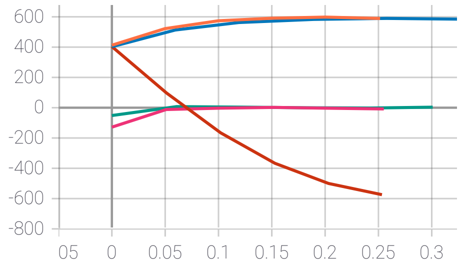
(a) Actor loss



(b) Critic loss



(c) Entropy coefficient



(d) Entropy loss

Figure 3.2: Sample SAC grid search log

Obtaining training statistics for all of the model runs allows for examination of the relevant metrics and selection of the most stable model. The log in Figure **??** presents such a screen for a search over SAC models for the SPX Growth 50 data set. Since it is desirable that the networks learn the Q-function – or other learned function – in a stable fashion, we would like to minimize the critic and actor loss such that the curve is smooth. Critic loss is a standard MSE loss and we want it to converge towards a particular quantity, whereas the actor loss is a negated version of the Critic loss (**lapan2018deep**) and we would like it to smoothly decrease. Similarly we would like the entropy loss – or any other endogenous loss function specific to the model – to be stabilized and decrease in an orderly fashion. From Figure **??** we see that the best candidate is the model drawn in red, as other candidate models such as the one drawn in pink suffer from stability issues. We would therefore pick this model to be our globally optimal model within our soft search space and use the parameters associated with it in the testing. These hyperparameter configurations are presented in appendix **??**.

3.2.2 Rolling Test

Having searched for an optimal model, weights $\boldsymbol{w}^*$ are obtained for each t in the testing set that starts on 1st of Jan 2019 and ends at 1st of Jan 2021. These weights are the weights suggested by the RL agents as optimal for each day, and I will use them for rebalancing the testing portfolios. Note that the model is trained with daily data, but the rebalancing period is set to be larger. I follow the convention of monthly rebalancing (**Demiguel2005**), which is what is reported, but also conduct tests for weekly, quarterly and semi-annual rebalances as robustness checks. The rolling test as illustrated by **??** simply executes trades after a period of h trading days given a vector of optimal weights, repeating this process until T. For the benchmark models, this implies a rolling estimation of necessary parameters $(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$ and solving the optimization problem at each t_n with the result being another set of optimal weights simulating the test portfolio. For the factor constrained approach as described in appendix **??**, a new factor model is estimated on a rolling basis.

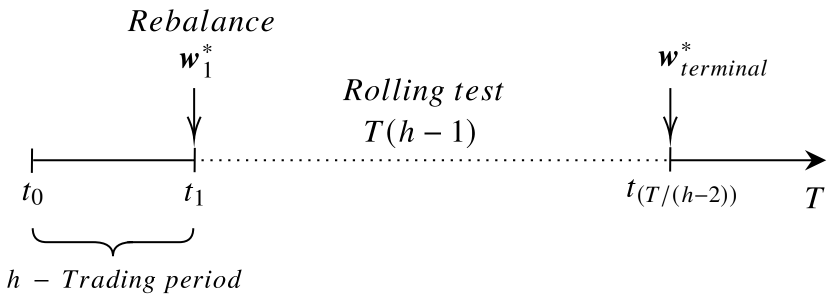


Figure 3.3: Rolling test diagram

Given this procedure we can obtain returns for each of the T/h discrete trading periods as:

$$\mathbf{r}_t = \boldsymbol{w}_{t-1}^* \cdot \mathbf{p}_{[t,t+h)} - \kappa \Delta \boldsymbol{w}_{t-1}^* \cdot \mathbf{p}_{[t,t+h)}, \tag{3.2}$$

where I adjust for transaction costs with $\kappa = 0.001$ that is subtracted as a fixed amount scaling $\Delta\mathbf{w}^*$, which represents the trades that would have taken place. This adjustment is implemented during the training of the RL agents as well, in order to prevent over-trading and reduce the effective number of bets the model is taking. Subsequently, these returns are aggregated geometrically as $\prod_{i=1}^{T}(1+r_i),\ \forall r \in \mathbf{r}_{test}$, and obtain a cumulative performance of the test portfolio. From these returns we can compute the performance and risk measures and other results as presented in the next section.

3.3 Results

Single-Factor

Looking at the results from the S&P 500 single-factor data sets in Table **??** we see that all of the algorithms except A2C have outperformed the equi-weight portfolio on a risk-adjusted basis in the Growth 50 data set, however, none of them were able to beat this benchmark on the Momentum and Size data sets. Comparing the performance to the mean-variance approaches, we see that these portfolios consistently beat the equal risk contribution as well as the minimum variance portfolio in the case of SAC and DDPG. Interestingly, none of the algorithms beat the factor constrained portfolio in terms of performance.

	Growth 50				**Momentum 50**				**Size 50**			
	r^{ann}	SR_α	Max DD	$skew(r)$	r^{ann}	SR_α	Max DD	$skew(r)$	r^{ann}	SR_α	Max DD	$skew(r)$
A2C	37.1%	1.34	-32.8%	-0.35	25.9%	0.95	-35.5%	-0.75	2.54%	0.03	-57.8%	-0.73
DDPG	36.4%	**1.30**	-34.0%	-0.48	24.4%	0.94	-32.5%	-0.80	3.8%	0.05	-59.9%	-0.59
PPO	35.0%	**1.27**	-34.6%	-0.53	26.2%	0.98	-33.5%	-0.77	1.0%	-0.01	-59.6%	-0.81
SAC	37.6%	**1.40**	-32.3%	-0.52	26.6%	1.00	-32.6%	-0.77	4.2%	0.07	-57.3%	-0.69
TD3	38.3%	**1.41**	-32.%	-0.37	26.0%	0.98	-33.8%	-0.66	2.3%	0.02	-57.9%	-0.48
EW	36.8%	1.36	-33.1%	-0.45	27.4%	1.04	-33.5%	-0.76	5.2%	0.09	-57.5%	-0.54
MV	25.2%	1.13	-22.3%	-0.06	22.4%	0.97	-28.6%	-1.04	3.1%	0.05	-41.7%	-0.86
ER	27.8%	1.19	-26.7%	-0.59	19.3%	0.84	-29.6%	-0.94	-2.3%	-0.14	-44.5%	-0.94
FC	53.1%	1.67	-28.4%	-0.45	43.8%	1.46	-31.3%	-0.61	38.9%	1.04	-41.5%	0.10
MSR					46.2%	1.70	-30.4%	-0.91				

Table 3.2: S&P 500 Single Factor Datasets Result

In terms of skewness, the RL agents are good at reducing left-tail risk uniformly in the growth and momentum factor, but struggle with the Size data set. Although all of them beat the mean-variance benchmarks in the size factor, negative skewness remains larger than the equi-weight portfolio. There are no large deviations in terms of drawdowns from the equi-weight portfolio, although generally the RL models deflate the drawdowns slighly. This is true particularly for the SAC that is the only model that is able to decrease drawdown across all of the three factor data sets.

We can see from Figure **??** that the 5% VaR is very similar for the equi-weight benchmark and the RL models, where only the SAC and TD3 perform better in the momentum and growth factors, but the problems with the Size data set remains. In terms of the mean-variance approaches, these both feature a lower VaR than RL for the size factor. Minimum variance and equal risk parity also outperform on the growth and momentum factors, but we see that all of the RL models outperform the factor constrained specification in the momentum factor. For A2C, SAC an TD3,

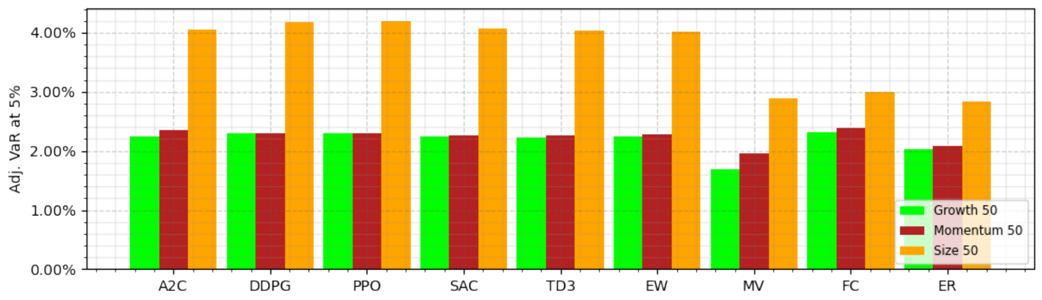


Figure 3.4: S&P 500 Cornish-Fisher Adjusted Value at Risk

better performance is noted for the Growth 50 data set with a slightly lower VaR than the benchmark.

In the European equities, the RL algorithms seemed to be more successful at beating the equi-weight in the performance measures. All of them reported a slightly better performance on the Growth 50 dataset with SAC beating the EW Sharpe by 11%. In the momentum and growth factors TD3 was less successful, but SAC and DDPG still managed to cross the line. All of the algorithms outperformed the minimum variance, but fell short when compared to the linear factor constrained portfolio. An interesting anomaly happens with the momentum factor, where the equal parity outperforms not only the RL models but also all of the mean-variance benchmarks and the MSR which often results in a portfolio that is largely overfit and very concentrated[8].

	Growth 50				**Momentum 50**				**Size 50**			
	r^{ann}	SR_α	Max DD	$skew(r)$	r^{ann}	SR_α	Max DD	$skew(r)$	r^{ann}	SR_α	Max DD	$skew(r)$
A2C	17.6%	**0.90**	-26.6%	-1.95	5.3%	0.11	-51.0%	-0.31	-4.9%	**-0.21**	-46.6%	-0.49
DDPG	18.2%	**0.92**	-27.2%	-1.88	7.8%	**0.19**	-51.0%	-0.37	-5.1%	**-0.21**	-46.8%	-0.47
PPO	17.5%	**0.85**	-27.6%	-2.03	6.6%	**0.15**	-51.4%	-0.33	-5.9%	-0.24	-47.4%	-0.40
SAC	19.5%	**0.97**	-27.1%	-1.78	9.5%	**0.25**	-51.1%	-0.51	-5.2%	**-0.22**	-45.2%	-0.41
TD3	18.0%	**0.87**	-27.7%	-1.99	4.8%	0.09	-51.5%	-0.27	-6.2%	-0.25	-46.6%	-0.63
EW	17.0%	0.84	-27.4%	-1.96	5.1%	0.11	-51.3%	-0.25	-5.6%	-0.23	-46.6%	-0.58
MV	8.2%	0.44	-21.7%	-1.37	-2.4%	-0.16	-44.1%	-1.07	-8.2%	-0.47	-35.1%	-1.28
ER	6.1%	0.30	-21.8%	-1.56	0.1%	0.96	-41.2%	-0.72	13.3%	-0.49	-36.5%	-1.59
FC	42.2%	1.68	-28.9%	-0.92	24.8%	0.93	-38.5%	-0.97	15.1%	0.44	-45.7%	-0.4
MSR	42.24%	1.95	-23.5%	-0.44	25.1%	-0.06	-38.5%	-1.03	-8.6%	0.39	-45.7%	-0.39

Table 3.3: Bloomberg 500 Single Factor Datasets Result

Assessing the distributional properties of the result, RL seems to increase the negative skew in the growth dataset, where all of the models outperform the benchmark, suggesting that they are taking more risky bets. In the momentum and size dataset, however, they reduce negative skewness and beat the mean-variance benchmark as

[8]Since this is a result of a greedy search over the efficient frontier it often results in a small number of positions being very overweight due to unstable moments estimates.

well as the EW benchmark in the size factor. Although the drawdown remains to be around the same level as the EW benchmark, this shows that there could be benefits from reducing the left-tail risk. In contrast to the U.S. equities, RL algorithms performed better in terms of risk measures what is presented in Figure **??** as the difference between the VaR of RL algorithms and the FC mean-variance optimization. In the growth factor, all of the RL algorithms perform better than the factor model. In the size factor, this result is not as pronounced, although notably the SAC algorithm outperforms the FC and the EW. Momentum factor shows that the mean-variance outperforms on VaR measure, although SAC beats the EW again by a small margin.

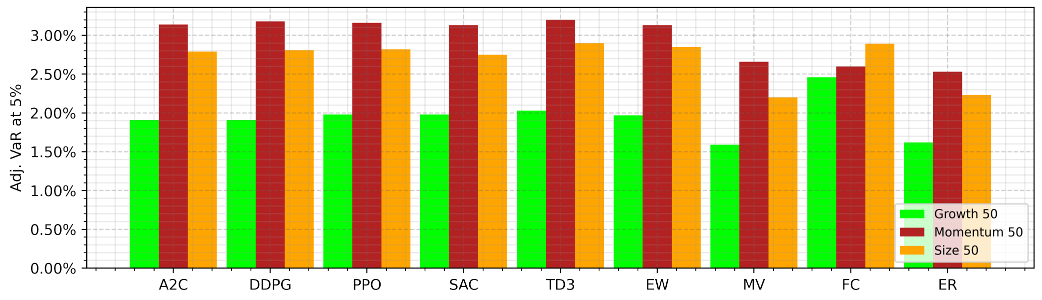


Figure 3.5: Bloomberg 500 Cornish-Fisher Adjusted Value at Risk

Multi-Factor

In the two factor data sets, where I combine growth and momentum factors with the size factor in order to learn a hedging strategy, RL performs better on the U.S securities than the European-listed stocks. Hedging growth with size, SAC and PPO stands out with better SR_α. In terms of risk, SAC is able to reduce the drawdown slighly, albeit skewness remains to be about the same level. These two models also beat MV, but are unable to beat the linear factor as well as the equal parity portfolios. Notably the ER portfolio is able to reduce the DD of the test strategy by more than 13% from the EW benchmark.

In the momentum combined with size, TD3 seems to do a good job at reducing the left-tail risk as well as outperforming the EW benchmark. Similarly, SAC seems to perform just as good as EW and PPO and DDPG also perform considerably well with DDPG increasing the SR_α by 21%, while still reducing drawdowns slightly. With regards to the mean-variance benchmarks, DDPG outperforms MV and ER but falls short of the linear model again.

	Growth + Size 100				Momentum + Size 100			
	r^{ann}	SR_α	Max DD	$skew(r^{ann})$	r^{ann}	SR_α	Max DD	$skew(r^{ann})$
A2C	19.8%	0.55	-45.9%	-0.75	17.6%	0.51	-44.5%	-0.95
DDPG	20.1%	0.58	-43.6%	-0.86	20.6%	**0.63**	-42.1%	-0.89
PPO	22.0%	**0.63**	-44.2%	-0.83	18.8%	**0.55**	-44.5%	-0.87
SAC	21.8%	**0.66**	-42.5%	-0.82	17.4%	0.51	-43.6%	-0.88
TD3	21.1%	0.61	-45.0%	-0.80	18.6%	**0.54**	-43.3%	-0.73
EW	21.3%	0.62	-43.9%	-0.83	17.9%	0.52	-43.9%	-0.88
MV	16.1%	0.58	-32.6%	-0.56	13.1%	0.41	-39.0%	-1.00
ER	16.3%	0.68	-29.4%	-0.77	14.4%	0.60	-29.8%	-1.05
FC	23.0%	0.70	-41.2%	-0.52	21.6%	0.65	-40.4%	-0.76
MSR					20.6%	0.62	-40.4%	-0.77

Table 3.4: S&P 500 Multi-Factor Datasets Result

Looking at the results tested on the European stocks, the RL agents perform rather poorly with only TD3 and A2C outperforming the EW benchmark on growth and size and momentum and size in that order. However, it does so with problems to learn a good hedging strategy, as we see in the case of TD3 resulting in inflated negative skew. These results show that EW portfolio is superior to mean-variance approaches too as only the factor model beats the EW in both of the cases. In contrast to this, the RL approach works better for all of the models except for PPO – where after inspecting the trading pattern, transaction costs probably cut from the performance considerably. Both the MV and ER approaches perform worse with ER, failing quite hard in the momentum size dataset. In general, the RL approach underperforms but not by a wide margin, and seems to have quite stable results not deviating from the EW portfolio to a large extent.

	Growth + Size 100				Momentum + Size 100			
	r^{ann}	SR_α	Max DD	$skew(r^{ann})$	r^{ann}	SR_α	MAX DD	$skew(r^{ann})$
A2C	4.2%	0.11	-38.5%	-1.20	1.2%	**-0.01**	-47.1%	-0.50
DDPG	4.2%	0.11	-37.0%	-1.26	-0.9%	-0.08	-47.1%	-0.38
PPO	2.0%	0.02	-39.5%	-1.64	-1.0%	-0.08	-47.3%	-0.60
SAC	4.6%	0.12	-37.8%	-1.25	-0.9%	-0.08	-48.8%	-0.58
TD3	5.7%	**0.17**	-37.3%	-1.38	-0.3%	-0.06	-47.4%	-0.41
EW	5.0%	0.14	-37.4%	-1.27	0.9%	-0.02	-47.2%	-0.48
MV	1.2%	-0.01	-33.5%	-1.31	-3.9%	-0.2	-42.9%	-0.78
ER	2.6%	0.07	-23.7%	-1.64	-6.4%	-0.39	-36.0%	-1.48
FC	13.6%	0.46	-39.0%	-1.48	3.4%	0.06	-46.9%	-0.67
MSR	11.5%	0.38	-38.6%	-1.49	3.2%	0.06	-46.9%	-0.67

Table 3.5: Bloomberg 500 Multi-Factor Datasets Result

In the sector ETF data set I test for the ability for the models to hedge across the sectors, as there is a different and broader set of exposures for the model to learn. The results show that the simple A2C has been the most successful to learn these paterns, with all of the other models having a relatively good performance as well. DDPG and SAC have been able to reduce the DD and negative skewness the most, suggesting that they were able to capture some of the pricing structure of these complex[9] ETF's. Minimum Variance and Equal Risk deliver lower risk-adjusted returns, but again we see that the factor constrained strategy is superior. Achieving a Sharpe close to the MSR portfolio, it is able to reduce the skew by almost a half while pushing the drawdowns down by 5%.

	Sector ETF			
	r^{ann}	SR_α	Max DD	$skew(r)$
A2C	16.5%	**0.58**	-36.1%	-0.78
DDPG	12.9%	0.44	-36.3%	-0.66
PPO	12.5%	0.42	-37.6%	-0.92
SAC	13.1%	0.46	-35.3%	-0.73
TD3	14.8%	**0.50**	-37.0%	-0.77
EW	13.7%	0.47	-36.9%	-0.77
MV	8.6%	0.33	-27.7%	-0.41
ER	8.1%	0.26	-34.4%	-0.70
FC	25.1%	0.87	-31.2%	-0.39
MSR	25.1%	0.88	-31.6%	-0.68

Table 3.6: Sector ETF Dataset Result

3.4 Inference and Robustness

In order to look at the out of sample stability of the results, we would like to run many trials on different samples and ideally obtain some sort of a bootstrap result, or arrive at a more robust result. This is due to the the results being representative of just one instance of a sequence of random variables of prices. Because of the complexity of training and optimizing RL models, however, such a Monte Carlo experiment is hard to conduct. Hence, in addition to the monthly rolling test, a weekly, quarterly and semi-annual windows have been tested and checked for large divergences that would suggest a high turnover of the strategy rendering the model useless for practical purposes. Referring to Figure **??**, we can see that most of the rankings of the algorithms in terms of performance propagate through these windows. Notably, the PPO algorithm seems to deviate more across the windows with more unstable profile due to over-trading as confirmed by the animations I

[9]Complex from the standpoint of an asset pricing equation as there are many risk premia defining an ETF.

produced for the rebalancing process in the tests. This has been the case with all of the data sets[10], which is interesting due to the PPO' construction and conservative updates mechanism.

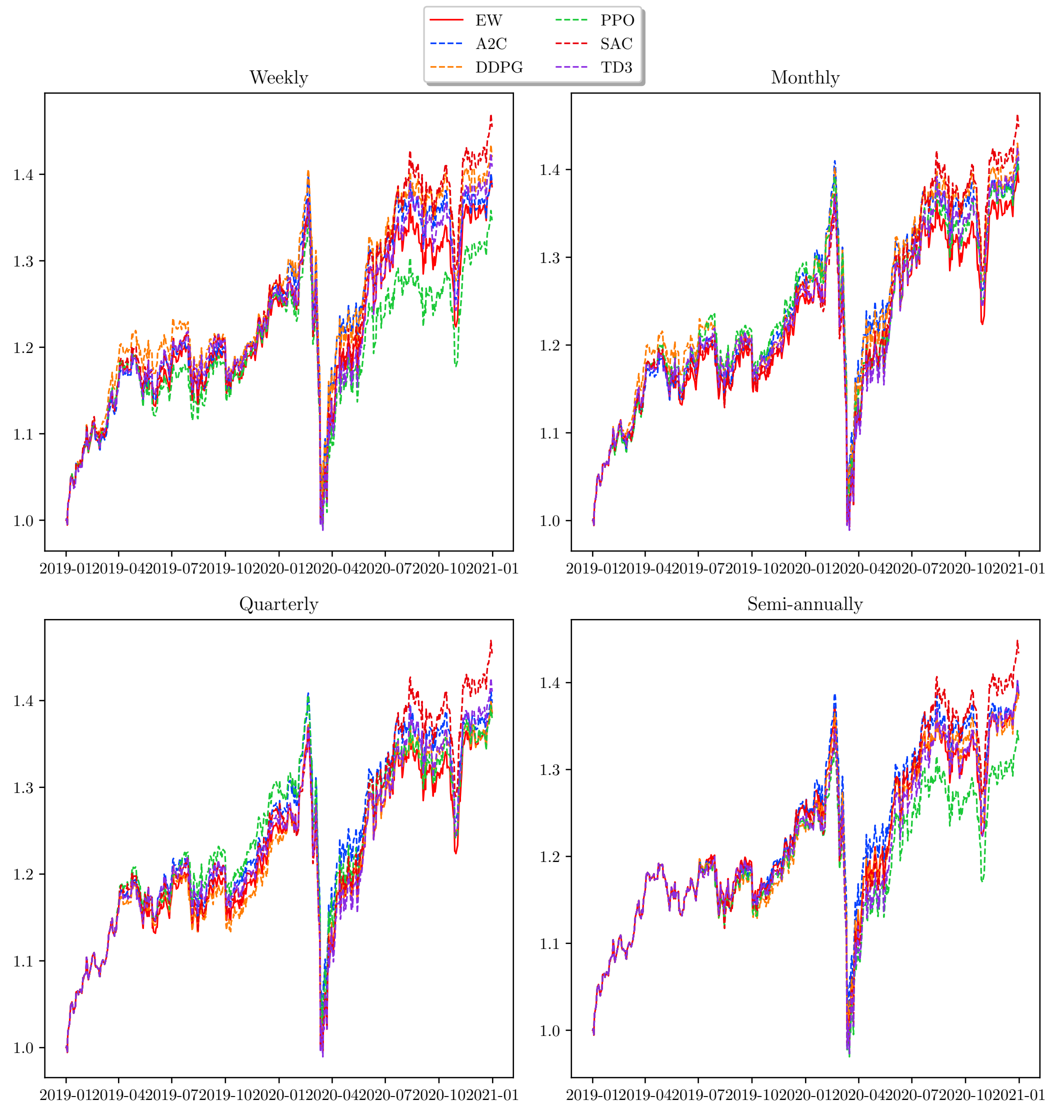


Figure 3.6: BE500 Growth 50 Rolling Tests Cumulative Return

Another issue, that is often times overlooked when such rolling tests are conducted is the large variance of the Sharpe Ratio estimator. It is important to realize that all of the reported parameters such as the sample statistical moments are estimators and they have the same distributional properties of a parameter. Therefore, we need a probabilistic approach to assess the relative performance of the models. Given the

[10]Even if the ranking changed slightly, a clear winner and a clear loser algorithm has emerged. This suggests using a voting ensemble of the models.

non-normal features of the observed distribution of returns we need a way to account for the third and fourth moment as they can lead to a large inflation of the sample Sharpe Ratio due to the fact that any observed Sharpe Ratio can be expressed as a combination of two gaussians with different parameters (**Harvey2016**). This fact can be used to prove that the parameter follows a normal distribution and therefore we can form a test statistic in the form of:

$$\hat{\text{PSR}}(SR_\alpha) = z\left\{\frac{(\hat{SR} - SR_\alpha)\sqrt{T-1}}{\sqrt{1 - \hat{\gamma}_3\hat{SR} + \frac{\hat{\gamma}_4 - 1}{4}SR_\alpha}}\right\}, \tag{3.3}$$

where $\hat{SR}$ is the estimate, SR_α is the reference SR, $\hat{\gamma}_3$, $\hat{\gamma}_4$ are the skewness and kurtosis estimates and $z\{\cdot\}$ is the Standard Normal cdf. We call this the Probabilistic Sharpe Ratio (**Harvey2016**), and essentially we can interpret it as the probability that the observed Sharpe ratio is higher than the reference. This statistic is a function of T as well, so it requires larger returns in order to establish statistical significance.

Using Equation **??**, we can construct a hypothesis test of the form:

$$\begin{aligned} H_0 &: \quad SR_{test} = \omega \\ H_1 &: \quad SR_{test} > \omega, \end{aligned}$$

with a rejection threshold of $1 - P(\omega \leq \hat{SR}_{test}) > \alpha$, for a statistical significance level α. This framework can be used for deflating a reported Sharpe ratio, but I am rather interested in comparing the PSR on a relative level. For example from left Figure in **??** we can see that setting $\omega = 0$, the so-called skill-less level, all of the algorithms together with the benchmark pass the 95% significance level and with this level of confidence we can conclude that the population $SR_{test} > 0$. In addition, we can see that based on this probabilistic assessment, SAC and TD3 have a larger probability of rejecting the null, which is desirable. The Figure on the right suggests that if we run this test on an array of different possible SR, the majority of the algorithms fall short of $\alpha = 10\%$ somewhere between 0 and 0.05. Such an inverted sigmoid is present in all of the data sets, albeit with different cutoffs for statistical significance.

To compare the models, I decided to use a relative percentage marginal for a given test where $\omega = 0$. Note that we have established that the function mapping the

$P(SR_{test} > \omega|H_0) \rightarrow SR^*$ does not yield a linear relationship, and therefore it is hard to compare the magnitude of the marginals across the data sets as these functions feature different domain sets. It is, however, a good proxy for seeing how the models compare against the benchmark.

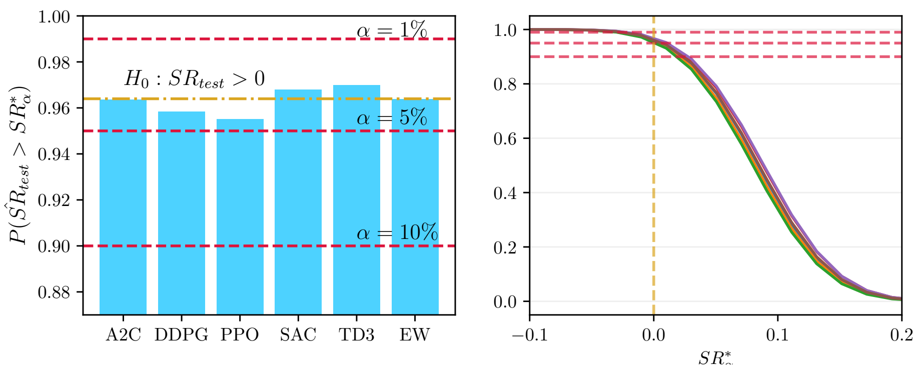


Figure 3.7: PSR test on SPX Growth 50

Dataset	A2C	DDPG	PPO	SAC	TD3
SPX Growth 50	-0.02	-0.55	-0.88	**0.40**	**0.59**
SPX Momentum 50	-1.82	-2.01	-1.20	**0.02**	-1.10
SPX Size 50	-3.42	-2.28	-5.71	-1.12	-3.99
SPX Growth + Size 100	-2.66	-1.52	**0.37**	**1.43**	-0.36
SPX Momentum + Size 100	-0.45	**4.28**	**1.23**	-0.42	**0.88**
BE500 Growth 50	**1.49**	**1.97**	**0.19**	**3.18**	**0.74**
BE500 Momentum 50	**0.01**	**4.46**	**2.24**	**7.69**	-1.13
BE500 Size 50	**1.10**	**1.10**	-0.53	**0.56**	-1.09
BE500 Growth + Size 100	-1.67	-1.67	-6.78	-1.11	**1.65**
BE500 Momentum + Size 100	**0.57**	**3.43**	-3.44	-3.42	-2.29
Sector ETF	**4.54**	-1.31	-2.29	-0.43	**1.3**

Table 3.7: PSR pct. marginal over EW for $H_0 : SR_{test} > 0$

The results show in Figure **??** suggest, that the algorithms perform better on the growth and momentum factors as evident from the positive marginals. In the case of the European equities, most of the algorithms except for TD3 in the momentum data set outperformed the EW benchmark. Generally, they seemed to struggle more with the U.S stocks. The only algorithm that performed relatively well on U.S as well as European single factor data was the SAC. Notably, for the multi-factor datasets the simpler A2C and DDPG have performed better than the more complex architectures such as PPO, SAC and TD3. This suggests that different architectures have advantages in learning in different systems.

Chapter 4

CONCLUSIONS

In this paper, I evaluate and benchmark state of the art Model-Free Reinforcement Learning model architectures against the Occam's Razor approach of equi-weight portfolio and the traditional mean-variance benchmarks. Evaluating the performance of these models on the European and American equities within a range of different contexts such as the single and multi-factor data sets, a complementary approach to portfolio management is proposed. This is done without the need to estimate the problematic parameters such as expected returns or the introduction of specific constraints to the SDF. This has been achieved using a sparse information set consisting only of price data and features transformed thereof.

The main conclusions of my work can be summarized in four points. Firstly, I conclude that the model-free architectures have outperformed the Mean-Variance approaches as demonstrated by the results and the probabilistic assessment of the return series. The resulting portfolios were more stable as a consequence of the models successfully capturing a part of the nonlinear structure in the SDF. This ability to capture the mispricings has not been, however, as successful as the inclusion of risk-premia in the construction of the conditional mean-variance portfolio. The second conclusion ipso facto implies that the factor model has outperformed the RL approach in agreement with the findings of **Chen2019**. Third, I note that the performance against the $\frac{1}{N}$ portfolio is largely tied and it is inconclusive to say that these algorithms outperform this benchmark in this setting. Although notably in some contexts such as the European Growth and Momentum data sets it has performed very well, in others, such as the Size data sets in both U.S. and EU equities, it fell short. The most consistent model architecture has turned out to be the SAC which hints at the inclusion of entropy within the model-free setting to be an important part of learning such nonlinear structures in a limited data setting as it introduces a lot of exploration into the framework. Fourth, I conclude that there are potential hedging benefits in using the RL approach as I have demonstrated that the models are successful at reducing left-tail risks out of the sample. This is certainly subject to the construction of a more complex decision system that would potentially make use of an ensemble model as different architectures performed well in different subsets of the data.

This paper is complementary to the current literature applying Machine Learning and Deep Learning in the Reinforcement Learning context and demonstrates the usefulness of such methods. Questions about interpretability as well as the use of coarser data frequency are subject to further research. Similarly, different asset classes and exposures should be tested to draw more general conclusions about the applicability of this approach. Since this is a rapidly developing field, an increasing number of papers considering similar applications are expected to be written in financial applications. New models from the field of computer science can be further considered for the Portfolio Allocation problem. As such, this paper provides a comprehensive review of the performance of the current state-of-the-art algorithms and provides a study similar to **Demiguel2005** in the context of Reinforcement Learning methods. Its use is of direct interest to portfolio managers as an alternative approach to construct and support investment strategies with an actionable output that consists of optimal weights similar to the Mean-Variance approach.

Appendix A

FEATURES

Simple Moving Average

Simple Moving Average is defined as a moving window of unweighted average prices over some interval d such that:

$$\text{SMA}_d(p_t) = \frac{1}{d} \sum_{i=n-d+1}^{n} p_i.$$

We can calculate SMA by moving this kernel along the time-series to obtain a smoothed version of the prices. In the training procedure, windows of 30 and 60 days were used.

Moving Average Convergence Divergence

MACD is also a trend feature that is constructed using two Exponentially Weighted Moving Averages:

$$\text{MACD}_{d^s,d^l} = \text{EMA}_{short} - \text{EMA}_{long}$$

This EMA smoothing of the prices is similar in logic to SMA, but the kernel applied here on the time series window is not uniform but exponential. This means that more recent observations are given more weight in the averaging, creating an auto regressive filter of the prices. We can define EMA recursively as:

$$\text{EMA}_d t = \left\{ \begin{array}{ll} p_t, & t = 1 \\ \alpha p_t + (1-\alpha) \cdot p_{t-1}, & t > 1 \end{array} \right\},$$

where $\alpha \in [0,1],\ \alpha \approx \frac{2}{d+1}$ as derived from the power series expansion of the EMA definition. For the long EMA, the window of 26 days is used whereas for the short window I use 13.

Bollinger Bands

Bollinger Bands are a combination of a SMA and an upper Bollinger band and a lower Bollinger band. A Bollinger band is a transformed SMA by adding an estimate of a short standard deviation as a way to capture signals outside of 'normal' behavior (**BBHayes**). We express it as:

$$\mathrm{BB}_d = \mathrm{SMA}_d \pm K\hat{\sigma}_d$$

where we estimate σ_d as $\hat{\sigma} = \frac{1}{d-1}\sqrt{\sum_{i=1}^{d}(p_i - \bar{p})^2}$. We use $d = 20$ for the estimation window and $K = 2$ as price moves outside of this band are considered unusual and can pose a positive or a negative signal. We directly input the upper and lower band as two features for the training of the reinforcement learning model

Relative Strength Index

RSI is a momentum feature that tries to capture oscillations within the price that could suggest a directional shift. It's bounded from 0 to 100 where a low value suggests a bullish signal and a high value a bearish signal within the selected timeframe (**RSIChen**). We can calculate it as follows:

$$\text{Relative Strength}_d = \frac{\bar{\mathrm{U}}}{\bar{\mathrm{D}}},$$

where commonly $\bar{\mathrm{U}}, \bar{\mathrm{D}}$ are estimated using an EMA_d of U or D, and

$$\mathrm{p}_d = \begin{cases} u_t, u_t \in \mathrm{U} & \forall p_t^{close} > p_{t-1}^{close} \\ d_t, d_t \in \mathrm{D} & \forall p_t^{close} < p_{t-1}^{close} \end{cases}$$

Then we just index the Relative Strength from 0 to a 100 as:

$$\mathrm{RSI}_d = \frac{100 - 100}{1 + RS}.$$

Commonly a value $d = 14$ is used to estimate the RSI.

Commodity Channel Index

CCI is also a momentum feature typically used to establish longer term trends. We use it with a configuration of 22 days, or a full trading month to complement RSI and capture longer term trends. We define it as follows:

$$\mathrm{CCI}_d = \frac{\hat{p}_t - \mathrm{SMA}_d(\hat{p}_t)}{0.015\frac{1}{n}\sum_{i=1}^{n}|p_t - \mathrm{SMA}_d(p_t)|},$$

where $\hat{p}_t = \frac{p_t^{high}+p_t^{low}+p_t^{close}}{3}$.

Directional Movement Index

Directional Movement Index, often used in an averaged form over a period of time is a momentum feature that tries to gauge the strength of a formed trend. We can define it using these quantities as in (**DXHayes**) first defining a directional movement as:

$$\begin{aligned}\mathrm{DM}_t^+ &= p_t^{high} -_t^{high} \\ \mathrm{DM}_t^- &= p_t^{low} -_t^{low} .\end{aligned}$$

We also need the Average True Range, a non parametric definition of volatility of a security (**ATRHayes**) defined as:

$$\mathrm{ATR}_t = \frac{1}{d}\sum_{t=1}^{d}\max[(p_t^{high} = p_t^{low}), \mathrm{abs}(p_t^{high} - P_t^{close}), \mathrm{abs}(p_t^{low} - p_t^{low})].$$

Then consider a smoothing function of the arbitrary form:

$$\gamma_d(x_t) = \sum_{t=1}^{d} x_t - SMA_d(x_t) - x_t.$$

Hence we can define the directional indicators and directional movement index as:

$$\begin{aligned}DI_t^{\pm} &= \frac{\gamma_d(DM_t^{\pm})}{ATR_t} \\ DMIX_t &= \frac{|DM_t^+ - DM_t^-|}{|DM_t^+ + DM_t^-|} \times 100.\end{aligned}$$

In this paper, $d = 30$ is used for capturing medium-term dynamics within the trend for the components of $DMIX_t$.

Covariance Matrix

For each state s_t I also calculate a sample covariance matrix $\hat{\mathbf{\Sigma}}$ that is used as a feature to train the associated neural networks. A lookback of one year or 252 trading days is used, and the covariance matrix is estimated as:

$$\hat{\mathbf{\Sigma}} = \frac{1}{N-1}\sum_{t=1}^{N}(\mathbf{p}_{\cdot t} - \hat{\mathbf{x}})(\mathbf{p}_{\cdot t} - \hat{\mathbf{x}})^T$$
$$\hat{\sigma}_{ij} = \frac{1}{N-1}(\mathbf{x}_{i,t} - \hat{\mathbf{x}}_{\mathbf{i}})(\mathbf{x}_{j,t} - \hat{\mathbf{x}}_{\mathbf{j}})$$

Turbulence Index

I also indirectly condition on the overall level of volatility in the enviroment $\mathcal{E}$ used for. This is done using a turbulence index as defined in (**Liu2020**):

$$TB_t = (\mathbf{r}_t - \boldsymbol{\mu})\hat{\mathbf{\Sigma^{-1}}}(\mathbf{r}_t - \boldsymbol{\mu})^T,$$

where $\mathbf{r}_t$ is a vector of stock returns in $\mathcal{E}$, $\boldsymbol{\mu}$ is a vector of expected returns estimated as a simple average historic returns over a period of 252 trading days, and $\hat{\mathbf{\Sigma}}^{-1}$ is an inverted sample covariance matrix as described in previous section.

Appendix B

MODEL HYPERPARAMETERS

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.01	0.001	0.0001	0.005
γ		0.001		0.1	
batch		512	512	256	512
steps	60K	80K	60K	60K	60K

SPX Growth 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.01	0.001	0.0001	0.01
γ		0.001		0.1	
batch		512	512	256	512
steps	60K	80K	60K	60K	60K

SPX Momentum 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.01	0.01	0.0001	0.005
γ		0.001		0.1	
batch		256	256	256	512
steps	60K	80K	60K	60K	60K

SPX Size 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.01	0.0001	0.005
γ		0.001		0.1	
batch		256	256	256	512
steps	60K	80K	60K	60K	60K

BE 500 Growth 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.01	0.0001	0.005
γ		0.001		0.1	
batch		256	256	128	256
steps	60K	80K	60K	60K	60K

BE500 Momentum 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.0001	0.001	0.005
γ		0.001		0.1	
batch		256	512	256	512
steps	60K	80K	60K	60K	60K

BE 500 Size 50

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.001	0.0001	0.005
γ		0.001		0.1	
batch		256	256	256	256
steps	60K	80K	60K	60K	60K

SPX Growth + Size 100

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.01	0.001	0.001	0.005
γ		0.001		0.1	
batch		512	128	512	512
steps	60K	80K	60K	60K	60K

SPX Momentum + Size 100

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.001	0.0001	0.005
γ		0.001		0.1	
batch		256	256	256	256
steps	60K	80K	60K	60K	60K

BE500 Growth + Size 100

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.01	0.001	0.001	0.005
γ		0.001		0.1	
batch		512	128	512	512
steps	60K	80K	60K	60K	60K

BE500 Momentum + Size 100

	A2C	PPO	DDPG	SAC	TD3
α	0.01	0.001	0.001	0.0001	0.005
γ		0.001		0.1	
batch		512	128	256	512
steps	60K	80K	60K	60K	60K

Sector ETF

Appendix C

MEAN-VARIANCE OPTIMIZATION

Quadratic Risk Utility

Quadratic utility is the most common formulation of the portfolio allocation problem with a non negative parameter λ controlling for the level of risk-aversion determining the efficient frontier of optimal portfolios. In the first stage of solving this problem we solve:

$$\begin{aligned} \min \ & \lambda \mathbf{w}^T \Sigma \mathbf{w} - \boldsymbol{\mu}\mathbf{w}, \ \lambda \in \mathbb{R}^+ \\ \text{s.t.} \ & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{w} \geq 1. \end{aligned}$$

In the second step we choose a satisfaction measure which conforms the definition of a reward function $r(a_t, s_t)$ such as the sample Sharpe Ratio $\frac{\hat{\mu}_{test} - r_f}{\hat{\sigma}_{test}}$ and search over the set of optimal portfolios such that:

$$\mathbf{w}^* = \underset{\lambda}{\operatorname{argmax}} \ (r(a_t, s_t)).$$

We can search for $\mathbf{w}^*$ greedily since λ is a monotonically increasing function and thus we obtain a global maximum.

Global Minimum Variance

The advantage of a GMV portfolio is that there is no need for an estimate of $\boldsymbol{\mu}$ as in the standard mean-variance formulation. We can solve for this portfolio either analytically or using a simple convex program that reduces to the case where the least amount of risk is taken on:

$$\begin{aligned} \min \ & \mathbf{w}^T \Sigma \mathbf{w} \\ \text{s.t.} \ & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{w} \geq 1. \end{aligned}$$

Equal Risk Contribution

Risk contribution portfolios look at the optimization problem from a perspective of marginal volatilities of each of the assets and their contribution to the overall risk of the portfolio. An Equal Risk Contribution portfolio forces each of the assets to contribute an equal amount of risk such that the weights adjust the volatility contribution to be at parity with other assets.

We can define the volatility of a portfolio as $\sigma_p = \sqrt{\mathbf{w}^T \Sigma \mathbf{w}}$ and for each asset $i \in \mathcal{E}$ the marginal risk contribution:

$$\sigma_i = \frac{\partial \sigma_p}{\partial w_i} \Longrightarrow \sigma_p = \sum_{i}^{N} \sigma_i, \qquad \text{(by Euler's theorem)}$$

since σ_p is a homogeneous function of degree one expressible as:

$$\sigma_p(w) = \sum_{i=1}^{N} w_i \frac{\partial \sigma_i}{\partial w_i}$$

For a complete proof see **Maillard2011**. Expressing the above equation in a vector form we obtain the convex objective we can minimize:

$$\begin{aligned} \min \; & \sum_{i=1}^{N} \left\{ \frac{\sigma_p}{N} - w_i \cdot \frac{\sum_{i \in \mathcal{E}} w}{\sigma_p} \right\} \\ \text{s.t.} \; & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{w} \geq 1. \end{aligned}$$

Factor Constrained Portfolio

Given a time-series factor model of the form **??** we can estimate the covariance matrix as:

$$\hat{\Sigma} = \mathbf{F} \Sigma_{\mathbf{F}} \mathbf{F}^T + \hat{\psi},$$

where $\mathbf{F}$ is the vector of factor loadings, $\Sigma_{\mathbf{F}}$ the sample covariance of the factor loadings and $\hat{\psi}$ is the diagonal matrix of the form $\mathrm{diag}(\hat{\sigma}_1, \ldots, \hat{\sigma}_N)$. We can then solve for a factor constrained portfolio that limits the exposure on a particular factor by the vector υ. We use the value of 0.2 or 20% as a limit of the portfolio exposure for a particular factor. Given the fact that we only use long-only portfolio it is hard to decrease this limit as there might not be such a solution that would satisfy this constraint. We express the problem as a maximization of the objective as described in **diamond2016cvxpy**:

$$\begin{aligned} \max \quad & \mu^T \mathbf{w} - \gamma(\upsilon \hat{\Sigma} \upsilon + \mathbf{w}^T \hat{\boldsymbol{\psi}} \mathbf{w}) \\ \text{s.t.} \quad & \mathbf{w}^T \mathbf{1} = 1 \\ & \mathbf{F}^T \mathbf{w} = \upsilon \\ & \mathbf{w} \geq 1. \end{aligned}$$

INDEX

A
action, 7–9, 12, 19
action space, 11
actions, 10
actor, 14, 15, 25
Actor Critic, 19, 27, 30, 31, 34
actor critic, 13, 14, 16, 18
advantage, 14
affine constraints, 3
agent, 7, 8
algorithm, 10, 11, 13, 19
algorithmic trading, 2
algorithms, 1
artificial intelligence, 7
artificial neural network, 12
asset allocation, 5
automation, 1

B
baseline function, 14
batch size, 24
Bellman Equation, 9, 18, 19
benchmark, 2, 7, 21, 26–29, 33
beta quantile, 23
bibliography, 37
Bloomberg 500, 21, 23
britleness, 1

C
Capital Market Line, 7
closed form, 5
computing, 1
conic problem, 7
convex constraints, 7
critic, 14, 15, 20, 25

cross-section, 2, 21
Cumulative Distribution Function, 23

D
Deep Deterministic Policy Gradient, 15, 16, 19, 27–29, 31, 34
deep learning, 1, 2
deep networks, 1, 19
drawdown, 27, 29, 31
dynamic programming, 9, 10

E
efficient portfolio, 7
endogenous, 7, 25
entropy, 17–19, 24, 25, 35
environment, 2, 7
equal parity portfolio, 29, 30
equal risk parity, 27
equal risk parity portfolio, 29
equal weight portfolio, 27–30, 34, 35
equities, 21, 23, 28–30, 35
error, 12
estimate, 3
experience, 1, 15, 19
experience buffer, 20
exploration, 15, 17
exposure, 2, 31

F
factor constrained portfolio, 27–29
factor model, 2, 21–23, 26, 29, 35
feedback, 7, 14
figures, 6, 8, 11, 13, 15, 16, 18, 20, 24–26, 28, 29, 32, 34

G
Garry Kasparov, 1
Gauss-Markov theorem, 22
gradient, 12, 15, 18
gradient descent, 15, 23

grid search, 24

H

hedging, 23, 29, 30

hyperparameter, 17, 24, 25

I

investor, 5

K

Kullback-Leibler divergence, 17

L

learning rate, 24

left-tail risk, 27, 29, 35

loss function, 14

M

machine learning, 2

market regime, 21

Markov Decision Process, 7, 9, 10, 13

Maximum Sharpe Ratio, 28

mean variance frontier, 6

mean-squared error, 25

mean-variance portfolio, 6

mini-batch, 15

minimum variance portfolio, 6, 27–30

model architecture, 4

model-based, 1

model-free, 1, 10, 13, 21, 35

moments, 2, 33

Monte Carlo, 1, 10, 13, 31

multi-armed bandit, 2

multi-factor, 23, 35

N

network, 11

neural network, 23

no-arbitrage, 22

noise, 7, 15, 18, 20
nomenclature, v
nonlinear, 1, 2, 4, 7, 14

O
off-policy, 10, 17
on-policy, 11, 17, 18
optimization, 7
optimization problem, 10

P
parametric, 2
policy, 7–10, 12, 13, 15–17, 19
policy function, 4, 14, 19
policy gradient, 12–14
policy optimization, 1
portfolio, 3, 6, 26, 35
portfolio allocation, 1, 11
portfolio construction, 1
price, 3
Principal Components Analysis, 14
Probabilistic Sharpe Ratio, 33
probability distribution, 9, 14
Proximal Policy Gradient, 29–31, 34
Proximal Policy Optimization, 17, 19

Q
Q-function, 8, 11, 15, 18, 19, 25
Q-learning, 1, 10, 11
Q-table, 11, 12
quadratic program, 7
quadratic programming, 4

R
random variable, 3
rebalancing, 26
recurrent neural network, 12
reinforcement learning, 1, 4, 7, 9, 26–31

reinforcement leraning, 35
return, 3, 5, 26
reward, 3, 7–10
reward function, 3
rewards, 13
risk premium, 21, 22, 35
robustness, 2, 26

S
S&P 500, 21, 23, 27
sample Sharpe Ratio, 33
sector, 21
securities, 2
Sharpe Ratio, 3, 28, 31–33
shrinkage, 22
single-factor, 23, 27, 35
Soft Actor Critic, 17–20, 24, 25, 27–29, 31, 33–35
soft-update, 15
state, 2, 3, 7–9, 14, 19
state space, 11
states, 9
stochastic control, 1
Stochastic Discount Factor, 2, 22, 35
stochastic process, 2
survivorship bias, 21

T
tables, 23, 27, 28, 30, 31, 34, 41, 42
target network, 19, 20
temporal difference error, 11, 16, 18
time-discounting factor, 4, 24
trade, 3
transaction cost, 26, 30
transition matrix, 8, 14
Trust Region Policy Optimization, 17
turnover, 31
Twin Delayed Deep Deterministic Gradient, 27–30, 33, 34

Twin Delayed Deep Deterministic Policy Gradient, 19

U

utility function, 5

V

value, 9

Value at Risk, 27–29

value function, 4, 8–10, 12–14, 18, 19

variance, 3

volatility, 3

W

weights, 2, 20, 25, 26, 36