Aalto University
School of Business

FINANCIAL PORTFOLIO MANAGEMENT WITH EVOLUTION STRATEGIES-BASED REINFORCEMENT LEARNING

Master's Thesis
Wilhelm Ala-Krekola
Aalto University School of Business
Information and Service Management
Spring 2021

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author Wilhelm Ala-Krekola

Title of thesis Financial Portfolio Management with Evolution Strategies Based Reinforcement Learning

Degree Master of Science in Economics and Business Administration

Degree programme Information and Service Management

Thesis advisor(s) Timo Kuosmanen

Year of approval 2021 **Number of pages** 91 **Language** English

Abstract

In Reinforcement Learning, an algorithmic agent learns to execute actions in environment by interaction and reinforcement. The action points given by environment are called states in which agent can execute some action to maximize its reward function. Reinforcement Learning usually integrate Artificial Neural Networks as function approximators. Neural networks are usually trained with a technique called backpropagation which includes gradient calculation over parameters of the network. An alternative method for tuning the network parameters based on evolutionary computing is Evolution Strategies, which is based on random sampling around the search parameters in order to find better solutions over generations.

Financial Portfolio Management is a process in which an investor tries to allocate capital to some set of investment options to maximize the investment objective. This study implements Evolution Strategies Based Reinforcement Learning agent for financial portfolio management problem to dynamically allocate portfolio weights daily for given set of assets. The study is conducted as an empirical study in two different asset classes; ETFs from Pacific Exchange and stocks from OMX Helsinki Exchange.

This study concludes that the portfolio management agent can execute profitable and competitive policies in both in and out-of-sample period. All versions of the algorithms perform well against the benchmarks beating almost all benchmarks and having at least similar or better performance than the S&P500 following ETF in Pacific Exchange portfolio case. However, the profit maximizing objective function exposes the agent's portfolio to risk meaning higher volatility, Value-at-Risk and Condition Value-at-Risk than benchmarks which might lead to huge losses and fluctuations in portfolio value.

Keywords Reinforcement Learning, Evolution Strategies, Financial Portfolio Management

Tekijä: Wilhelm Ala-Krekola

Työn nimi Financial Portfolio Management with Evolution Strategies Based Reinforcement Learning

Tutkinto Kauppatieteiden Maisteri

Koulutusohjelma Informaatio- ja palvelujohtaminen

Työn ohjaaja(t) Timo Kuosmanen

Hyväksymisvuosi 2021 **Sivumäärä** 91 **Kieli** Englanti

Tiivistelmä

Vahvistusoppimisessa agentti pyrkii ratkaisemaan ongelmaympäristön suorittamalle toimia, joiden perusteella ympäristö palauttaa agentille palkkion tai rangaistuksen riippuen toimenpiteestä ja tästä johtuvasta ympäristön muutoksesta. Agentin tavoite on maksimoida ajan saatossa kertyvä kumulatiivinen palkkio. Vahvistusoppimisessa hyödynnetään usein keinotekoisia neuroverkkoja. Neuroverkkojen parametreja tyypillisesti säädetään takaisinsyöttömenetelmän avulla, jossa neuroverkon parametrejä säädetään gradientti-informaation perusteella. Evolutionäärinen laskenta tarjoaa tähän vaihtoehtoisen tavan muuttaa neuroverkon parametrejä. Tässä menetelmässä parametreihin lisätään kohinaa satunnaisprosessin kautta, jonka jälkeen prosessia toistetaan, kunnes optimi arvo löytyy.

Sijoitussalkun hallinnassa sijoittaja pyrkii jakamaan sijoitettavan varallisuutensa arvopapereihin siten, että sijoitukset maksimoivat sijoittajan sijoitustavoitteet. Tässä tutkielmassa sovelletaan evolutionääriseen laskentaan perustuvaa vahvistusoppimista sijoitussalkun hallinnassa, jossa algoritminen agentti sijoittaa dynaamisesti sille annettua varallisuutta annetuille arvopapereille päivittäisellä sijoitusvälillä. Tutkielma on toteutettu empiirisenä tutkimuksena, jossa algoritmeja sovelletaan kahden tyyppisiin arvopapereihin; pörssinoteerattuihin rahastoihin yhdysvaltalaisessa Pacific Exchange pörssissä ja yksittäisiin osakkeisiin OMX Helsingin pörssissä.

Tämän tutkielman tulokset osoittavat, että vahvistusoppimismenetelmiin perustuvat sijoitussalkun hoitaja agentit pystyvät suorittamaan tuottoisaa strategiaa sekä opetus, että testiaineistossa. Algoritmi pystyy suoriutumaan sijoittamisesta vertailuindeksejä paremmin ja S&P 500 rahaston kohdalla ainakin yhtä hyvin tai paremmin. Tuoton maksimoiva tavoitefunktio kuitenkin altistaa agentin suuremmalle riskille, mikä johtaa korkeampaan volatiliteettiin, Value-at-Risk ja Conditional Value-at-Risk arvoon.

Avainsanat Vahvistusoppiminen, evolutionaarinen laskenta, sijoitussalkun hallinta

Table of Contents

List of Tables

List of Figures

1 Introduction

The evolution of Reinforcement Learning (RL) methods has been remarkable during the 2010's. Although it is not a new field study in Machine Learning (ML) and Artificial Intelligence (AI), the growth of computational capacity has opened new doors for developing intelligent self-learning agents that learn to do tasks by exploring the problem environment via trial and error. The success has also been enabled with use of Artificial Neural Networks (ANN), which are used as a function approximators, mimicking brain like behavior and computation. The ANNs can scale the dimensionality of the state spaces from tabular methods to continuous and multi-variable presentations (Arulkumaran, Deisenroth, Brundage, & Bharath, 2017).

Reinforcement Learning has been able to solve problems in hard environments like Atari games and board games like Go and Chess (Barto, Thomas, & Sutton, 2017). However, there are not that many as famous implementations like gameplaying AlphaGo (nowadays MuZero), solving business related or other problems, as the research has been highly focused on topics such as game-playing or self-driving cars. Dulac-Arnold, Mankowitz & Hester (2019) define the main problems of using RL in real world problems as, training offline with limited data and samples, high-dimensional action and state spaces, safety constraints that should not be violated, partially-observability and non-stationarity, unspecified reward functions, systems who desires explainable policies, real-time inference and large and unknown delays of system actuators, sensors and rewards.

Financial portfolio management on the other hand, is a famous topic in field finance. One of the most famous methods developed to handle financial portfolio is a mean-variance portfolio by Harry Markowitz (1952). Financial Portfolio Management is a process of selecting and allocating assets in considered portfolio. There are two main types of portfolio management: active and passive. In active portfolio management the asset allocation i.e., weights of the portfolio are managed frequently over trading horizon, whereas in passive the allocation happens only once in the trading period (Fabozzi & Markowitz, 2011). A lot of different techniques have been developed to manage portfolios and allocate assets. Also, the machine learning methods such as artificial neural networks have been used to develop trading and portfolio management methods.

Also, RL methods have been used to build trading and portfolio management methods. Some of the studies like Moody & Saffell (1999), Gold (2003), Huang (2018) and Zhang,

Zohren & Roberts (2020) have been focusing on trading one asset and some others on focusing multiple assets at once like Jiang & Liang (2017). So-called policy gradient methods have been then used especially in financial portfolio management studies related to reinforcement learning. However, training portfolio management agent is hard. Using large neural networks as function approximators can be computationally expensive as they require a lot of analytical computation in backpropagation phase when their weights are tuned against loss function. One other thing that makes learning hard in financial portfolio management is longer episodes i.e., investment horizons can be long. Jiang, Xu & Liang (2017) developed portfolio management agent which incorporated Policy Gradient (PG) reinforcement learning to successfully manage portfolio which was based on Poloniex cryptocurrencies. They were able to make profitable strategy using self-learning RL-agent. They also experienced that long investment horizon caused problems for agent as the signal was losing its significance if the training data was not near the actual trading period (Jiang, Xu, & Liang, 2017).

To address the expensive and sometimes hard objective functions, Evolution Strategies (ES) optimization methods have been developed. Evolution strategies optimization methods learn the objective by using so-called fitness function and randomized search around the parameter space. These methods were first introduced in early 60's but was then developed further in 70's by Rechenbach (1978). They have later been applied to train neural networks and reinforcement learning agents as so-called direct policy search method. One of the most famous evolution strategies-based reinforcement learning implementation was developed by researchers at OpenAI by Salimans et al. (2017). One benefit of using evolution strategies with neural networks is that it can be used to learn the network weights without gradient based backpropagation step.

This study presents a financial portfolio management agent that tries to maximize the mean logarithmic return over the trading period. The agent will use artificial neural network as a function approximator which is tuned by using evolution strategies-based methods.

1.1 Aims of the Study

This study builds a portfolio management agent that is trained by using evolution strategies-based optimization methods. The purpose of the agent is to maximize the mean logarithmic return to maximize the cumulative returns. Built agents are then compared to each other and to benchmarks which are the relative indices of the market that the portfolio agent is in. The

aim is to find out whether it is possible to learn some profitable behavior to manage the chosen mix of assets in the portfolio.

By building the agent this study tries to answer the following research questions:

1. Can the Evolution Strategies Based Reinforcement Learning Agent learn profitable portfolio management policy?

2. Can the portfolio management agents beat the related benchmarks measured by the performance of the objective function i.e., average logarithmic daily return?

The study is conducted as an empirical study for two different types of portfolios and markets. One portfolio consisting of Exchange Traded Funds ETFs and the other portfolio consisting of stocks from the Finnish Stock Exchange. The trading agent is trained by using two different types of evolution strategies algorithms, one developed by researchers at OpenAI in study by Salimans et al (2017) and the other is Parameter-Exploring Policy Gradient developed by Sehnke et al. (2010).

1.2 Structure of the Thesis

This thesis consists of following parts: algorithmic theory, theory on portfolio management, empirical study setup, results and discussion. The first part gives a definition to both reinforcement learning and evolutions strategies. In this first part the fundamentals and basic algorithms of RL are introduced. Also, this section introduces evolutionary optimization method Evolution Strategies and the fundamental algorithms related to ES. The connection between ES and RL is then defined. Second part gives definitions for financial portfolio management and related metrics and introduces some of the studies that have been conducted by using reinforcement learning in trading and financial portfolio management.

The third part defines the setup for the empirical study. It defines the problem environment and how the RL agent interacts with environment and what are its observations. Also, it introduces the assets and markets from which are selected to be part of the study. Last the training and evaluation of the performance is given.

The last two parts present the results and discussion, respectively. The results are based on out-of-sample period, which the agent have not seen beforehand. With these results, this thesis tries to answer the research questions. The results also compare the agent's

performance to selected benchmarks. The study ends up in discussion which presents the take outs of the studies, some ideas for improvement and ideas for further research.

2 Theory

This chapter gives definitions for both algorithmic methods of Reinforcement Learning (RL) and Evolution Strategies (ES). First the theoretical framework for RL is defined. Next, the ES optimization method is given and later tied to the concept of RL.

2.1 Reinforcement Learning

The problem of reinforcement learning is at the same time a problem, a class of solutions and field of study that research these problems (Barto & Sutton, 2018). The two key components of the RL are agent and environment. The agent has some set of logic, decision rules and boundaries on which it tries to solve the environment, i.e., maximize the objective function given the problem. The environment defines the dynamics and rules for solving the problem. For example: the cleaning robot must clean the house, so agent tries to maximize cleanliness of some measure by moving and vacuuming in the environment i.e., living room.

The main essence of RL is to learn by interacting with the environment. The RL problem has three main components: states, actions, observations (which includes information on next state and rewards). Based on the reinforcement of the actions conducted in a current state by the RL agent, the environment sends the agent some reward and the next state as an observation. Based on the observation, the agent then tries to modify its behavior policy to make its actions more optimal towards the objective function.

The basic mathematical formulation of RL problem is presented with Markov Decision Processes (MDPs). MDPs are used to model sequential decision-making problems such as the RL problem. In MDPs the action chosen by the agent influence the next states and the rewards provided by the model. MDPs can be used to prove convergence of a sequential decision-making problem. If it is possible to represent RL problem as an MDP then there is a better chance of achieving convergence. However, it is not possible to formulate all problems as an MDP. For these non MDP problems RL can still provide a good solution although not having a proof of convergence.

There is also a class of MDPs in which the agent does not have a certain knowledge of the current state. Instead, it might have some belief of the current state based on the observations namely new states and rewards. That means that the agent does not have straight access to the state itself. This kind of formulation is called Partially Observable Markov Decision Processes (POMDPs).

Both MDPs and POMDPs are defined later in this chapter. The RL problem of this thesis is a POMDP. The agent has access to stochastic environment in which it will have noisy observations and noisy states. That means that the RL agent must have a belief of which kind of state it is in right now in order to make optimal decisions.

2.1.1 States, Actions, Observations and Rewards

In this section, elements of the RL problem that are dependent on the problem formulation, environment and agent's objective function, which the agent tries to maximize, is defined.

Barto & Sutton (2018) defines the state as follows. RL is usually learnt in discrete timestep sequences $t = 1, 2, 3, 4 \dots N$. At each time step t, the learning agent gets a representation from the environment which is called a state and it is usually notated as a $S_t \in S$, where S is the set of all possible states (Barto & Sutton, 2018). For example, in case of the cleaning robot, the states hold the information of orientation of the robot and distance to the walls. States of the environment can be either continuous or discrete or both depending on the specific problem.

Based on the current state S_t the agent chooses to conduct an action $A_t \in A(S_t)$, where $A(S_t)$ is all possible actions given the environments current state. The action a can be either discrete or continuous depending on a specific problem (Barto & Sutton, 2018). For example, in case of the cleaning robot the action can be whether to turn right or left, stop, go forward, or go backwards.

Based on the agent's action A_t at the state S_t the agent will get the observation O_t which will include information of the next state S_{t+1} and a reward R_t. The next state S_{t+1} is result of the changes in the environment based on the agent's actions and for example stochasticity i.e., randomness of the environment. The reward is result of the transition from state $S_t \rightarrow S_{t+1}$ and action a_t and is based on the problem at hand and the engineering of the so-called reward function. The agent's objective is then to maximize the cumulative reward G given by,

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

which is the total sum of reward R_t discounted by discount factor γ received in k steps in future. The discount factor γ accounts the considered future rewards from the future timesteps and the value of it can be between the range $0 \leq \gamma \leq 1$, zero meaning that only

the immediate rewards are considered and one meaning that all the future rewards are considered equally when evaluating the state (Barto & Sutton, 2018).

Based on gathered observations, the agent gains information of goodness of its actions which then can be used to tune the agent's behavior towards the objective function G. The whole learning process is presented in the Figure 1 below.

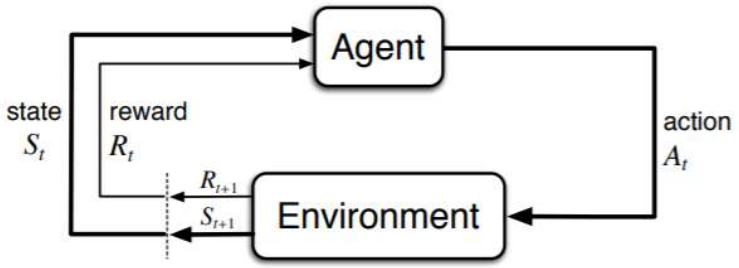

Figure 1 Agent-Environment interaction, Barto & Sutton (2018)

2.1.2 Markov Decision Processes and Optimal Policies

In this section a brief introduction to Markov Decision Processes is given. Based on MDPs it is possible to obtain definition of a policy and a value function. Last, iterative methods to solve optimal policies and value functions are presented.

2.1.2.1 Definitions of Markov Decision Processes

MDP is mathematical model to define sequential decision making and optimization problem. In MDPs the decision making is partly controllable by the decision maker and it might be partly random. One important thing in MDPs is the Markov Property which in short means that all the information that matters is assumed to be in the signal of the current state (Barto & Sutton, 2018). It is possible to model the RL problem with MDPs. MDP consists of set of states S, set of actions A, probability $P_a(s, s')$ i.e., transition probability from state s to new state s' when taking action a and a reward function $R_a(s, s')$ which represents the immediate reward from transition from state s to new state s' taking action a. The objective function is to maximize cumulative rewards over time which is presented in previous section (Feinberg & Schwartz, 2012).

The traditional MDPs assume that the states are known when taking the action. If the MDP do not have that property, the algorithm is unable to solve the problem. However, in many cases agent cannot fully observe the state where it is at i.e., it has imperfect information

of its environment and states. Partially Observable Markov Decision Processes extends the idea of traditional MDP so that it has a belief b of a state s. MDP has states, actions, transitions and rewards, but in POMDP there is also a set of observations Z and set of conditional observation probabilities O. After being in state s and having a belief b of being in certain state, the agent executes action a, transits to new state s' with probability p and receives reward r then the agent must update its belief to b' (Hansen E. A., 1998).

RL can be used to solve MDPs by using the experience gathered during learning. Reinforcement Learning is also able to learn MDPs where rewards are not known beforehand, or the transition probabilities are unknown.

2.1.2.2 Value functions

As defined in previous section 2.1.2.1, at each time step t the agent executes action a_t. The actions are mapped from some policy which is noted by π, which the agent tries to learn. The policy gives us $\pi(A_t|S_t)$, which gives the probability of executing action $a_t = a, if\ S_t = s$ (Barto & Sutton, 2018). Next the methods of finding some policy π is defined.

There are two main categories of finding optimal policy π^*: value iteration and policy iteration. Value iteration is based on estimating the value of action a in the state s which can be then turned into policy π. Policy iteration is based on executing the policy, then evaluating the values of actions.

To find the optimal policy π^*, the evaluation of value function is needed. Value function is defined as given state s and policy π, find value $v_\pi(s)$ of the state, that maximizes the expected reward of the state. The value tells how good or bad being in that state is for the agent. Also, the purpose of value function is to tell the expected rewards of being in a current state s_t and executing policy π onwards. Given the framework of MDPs it is possible to define the value function as:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\,|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \tag{2}$$

where a is action, s is state, r is reward and s' is new state and $p(s', r|s, a)$ is a transition probabilities for rewards and new state. Similarly, it is possible to define the value of taking action a in state s under policy π by expectation of q_π which is defined as:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A = a] \quad (3)$$

This q_π is the expected value of taking an action a in state s by then executing policy π (Barto & Sutton, 2018).

So, by solving the RL problem and the value function it is possible to obtain a policy that is collecting the most rewards in the long run. This is an optimal policy which is defined as:

$$v_* = \max v_\pi(s) \quad (4)$$

and for the state-action pair:

$$q_* = \max q_\pi(s,a) \quad (5)$$

for all $s \in S \; and \; a \in A(s)$. And for state-action pair this gives an expected reward obtained for taking action a in state s and after that continuing to execute the optimal policy which then generalizes as follows:

$$\begin{aligned} q_* &= \mathbb{E}[R_t + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \end{aligned} \quad (6)$$

or in terms of state-action pairs:

$$\begin{aligned} q_* &= \mathbb{E}[R_t + \gamma \max q_* (S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma q_*(s', a')] \end{aligned} \quad (7)$$

where a' is action in new state s'. (Barto & Sutton, 2018)

There is a limitation when estimating optimal values. In real life, one often ends up in a situation where the convergence to optimal policy is either almost impossible or possible but not achieved due to high computational expenses. Therefore, RL algorithms often rely on approximations when estimating these value functions (Barto & Sutton, 2018). Function approximation is later discussed in section 2.1.4.

Optimal policies are usually solved iteratively using Dynamic Programming (DP). DP is an algorithmic technique for solving sequential problems developed by Richard Bellman is 1950s. The DP breaks down the decision making into making decision in steps and using the value functions, which in terms of MDPs and RL have been previously defined in this section. The values for value function V can be learned working backwards recursively combined with use of so-called Bellman Equation. For MDPs, Bellman equation is an expectation of the immediate and future rewards of being in state s i.e., use previously defined value function for MDPs and RL which are already in the shape of Bellman equation. It is possible then to use DP to iteratively solve both optimal policies and value functions if a perfect model of the environment is given as an MDP (Barto & Sutton, 2018). Usually, it is not possible to perfectly define the environment, but this theory gives good framework for solving at least near optimal policies in different learning environments. Next section defines some basic algorithms to iteratively evaluate and solve policies and value functions.

2.1.2.3 Policy Iteration

In order to solve a policy, the agent must first iteratively evaluate the policies to make them better. The policy evaluation replaces old value of v_k in the state s with new value which the algorithm has gotten from the old values of the successor states of s. This gives the expected immediate reward in all one-step transitions that are possible using the policy one is evaluating (Barto & Sutton, 2018). Barto & Sutton (2018) defines the policy evaluation algorithm as the first part after initialization of equation 8.

Given the policy evaluation algorithm it is possible to evaluate how good a policy is, but there is also a need to know, if the policy is better than others or should the algorithm change the existing policy. By using the value function, agent then can alter the action a in state s and see whether the value is greater or less than $v_\pi(s)$ (Barto & Sutton, 2018). There are different ways to alter the chosen action which are discussed in later section. To simplify, greedy action selection is chosen i.e., take action a with highest value in state s.

Now, two main components of the policy iteration algorithm are defined, in which agent first evaluate the existing policy, then improve it and then again evaluate it. Next, it is looped over and over until convergence. The pseudocode for the Policy iteration algorithm is defined by Barto & Sutton (2018) as follows,

Algorithm 1. Policy Iteration

1. *Initialization*

$V(s) \in \mathbb{R}$ *and* $\pi(s) \in A(s)$ *arbitrarily for all* $s \in S$

2. *Policy Evaluation*

Repeat

$\Delta \leftarrow 0$

For each $s \in S$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$ (8)

until $\Delta < \theta$ (*a small positive number*

3. *Policy Improvement*

$policy - stable \leftarrow true$

For each $s \in S$:

$a \leftarrow \pi(s)$

$\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

If $a \neq \pi(s)$, ***then*** $policy - stable \leftarrow true$

If $policy - stable$, *then stop and return* V *and* π; ***else*** *go to* 2.

With this algorithm, agent can then evaluate and improve the existing policy until convergence. Next section presents an alternative way of obtaining optimal policy through value iteration.

2.1.2.4 Value Iteration

With policy iteration the agent first evaluates and then improves the policy which may cause some computational problems. The convergence properties of the policy iteration can be truncated by using value iteration algorithm. It combines the policy improvement and truncated policy evaluation steps with:

$$v_{k+1}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s, A_t = a] \quad (9)$$

This equation means that in each state s agent take action a with highest expected value. This modifies the policy iteration algorithm to a value iteration as follows.

Algorithm 2. Value Iteration

$$
\begin{aligned}
&\textbf{\textit{Initialize}}\ array\ V\ arbitrarily \\
&\textbf{\textit{Repeat}} \\
&\quad \Delta \leftarrow 0 \\
&\quad \textbf{\textit{For each}}\ s \in S: \\
&\qquad v \leftarrow V(s) \\
&\qquad V(s) \leftarrow \max_a \textstyle\sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] \\
&\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|) \\
&\textbf{\textit{until}}\ \Delta < \theta\ (a\ small\ positive\ number \\
&\textbf{\textit{Output}}\ a\ deterministic\ policy\ \pi, such\ that \\
&\quad \pi(s) \leftarrow \operatorname{argmax}_a \textstyle\sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]
\end{aligned}
\tag{10}
$$

As the algorithm demonstrates, it effectively combines both policy evaluation and improvement at each of its iterations. Both algorithms policy iteration and value iteration converge to optimal policy for discounted finite MDPs when the discount factor γ is $0 \leq \gamma < 1$. (Barto & Sutton, 2018)

Almost all the RL algorithms are based on these foundations represented previously. Basically, they differ on how they end up to optimal policy, whether they use so called Q-learning, SARSA, Policy Gradient (PG) or Actor-Critic (AC) framework, it all comes down of this iterative nature of finding the optimal policy.

2.1.2.5 Exploration and Exploitation

In RL, one often ends up with the dilemma of exploration and exploitation. This is a very broad topic and numerous different solutions have been developed over the course to balance out this trade-off. First brief definition of both exploration and exploitation is given, then the tradeoff dilemma between them is discussed. Lastly some of the most popular exploration algorithms are briefly discussed.

The exploration means that the RL agent is exploring the environment with some set of rules. The purpose of the agent is to find new states that it has not yet reached or try different actions in known states hoping it will get a better reward in the long run. The exploitation on the other half is to take advantage of the known rewards and optimal actions

in that state. However, if agent does not reach all the states or try different outcomes, it cannot really exploit the environment fully.

To analyze the performance of an algorithm the regret must be defined. The regret gives a definition of how much rewards are lost by executing a certain action against the optimal action. Bubeck & Cesa-Bianchi (2012) defines regret mathematically as follows in equation 11:

$$R_n = \mathbb{E}\left[\max_{i=1,\dots,K} \sum_{t=1}^{n} X_{i,t} - \sum_{t=1}^{n} X_{I_t,t}\right] \tag{11}$$

In which $X_{i,t}$ is the rewards and the forecasters choices are defined with I. This means that the regret is the difference between best reward minus the forecasters i.e., agents reward given the action. The K is number of actions and n is total number of time steps t.

The dilemma of exploration and exploitation means the tradeoff between lost rewards of not executing the optimal action versus the new state or optimal action when trying different actions. This means that one must decide how much agent can explore before starting to exploit or if the agent can exploit immediately with some amount of random exploration. The more agent can explore the more accurate its estimates get, but the more agent exploits the more reward it should get. The amount of how much agent can explore comes from the specific problem and how dangerous or costly it is to explore. There are methods developed to incorporate risk-aware exploration to environments where the exploration is risky, but they are not discussed in this thesis.

In last sections, the pseudocode assumed greedy action selection using max function over all possible actions a. This means that the agent really does not explore at all, instead it chooses greedily the best action. One popular alternative to greedy action selection is ε-greedy algorithm which selects the best action with probability of (1-ε) and uniformly random action with probability of ε (Vermorel & Mehryar, 2005). The formulation of ε-greedy is as follows in equation 12.

$$Action\ a_t = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q_\pi(a, s_t) & with\ probablity\ 1-\varepsilon \\ a \sim U(\pi(a)) & with\ probability\ \varepsilon \end{cases} \tag{12}$$

The second popular exploration algorithm is Softmax exploration. It is simply a Softmax function over action values in Q_π and is formulated as in equation 13:

$$a_t \sim \pi(a|s_t)$$
$$where\ \pi(a|s_t) = \frac{e^{Q_\pi(a_i,s_t)/\tau}}{\sum_{i=0}^{n} e^{Q_\pi(a_i,s_t)/\tau}} \tag{13}$$

where Q_π is the value of a current policy in state s_t, a_i is single possible action in policy and τ is the temperature which can smooth distribution of action probabilities. If τ is high then each action probability is almost equivalent, if it is low the agent will act almost greedily. Also, if the value of action becomes a lot higher than the other actions then it will be more certain to be selected as an action and if there is no difference in action values the actions are more uniformly distributed.

Explorations previously defined are based on random exploration. Upper confidence bound exploration algorithm UCB1 try to balance and explore those states and actions that the agent does not know much about. The UCB1 algorithm for sequential state RL problem is described as follows in equation 14:

$$a_t = \underset{a}{\operatorname{argmax}}\, Q_\pi(a, s_t) + U_t(a, s_t)$$
$$U_t(a, s_t) = \sqrt{\frac{2 \log t}{N_t(a, s_t)}} \tag{14}$$

In other terms this exploration algorithm tries to explore reward values on rewards upper confidence bounds so that the true rewards lie with high probability under that bound given by U. This algorithm will prevent the agent from randomly ending up in a bad state but will explore states that it sees a potential for high rewards. (Aurélien & Moulines, 2011)

With these different exploration algorithms, the learning agent can balance the exploration and exploitation in the environment. There are lots of different schemas developed in this specific research area, however these three are still one of the most popular ones. UCB1 is also used in search algorithms such as Monte-Carlo Tree Search which was used in implementation of famous AlphaGo by Google Deepmind.

2.1.3 Reinforcement Learning Algorithms

In last section, the framework for RL through MDPs was defined. Also, the dilemma of exploration and exploitation was defined with some most used solutions in modern practice.

Next some of the most popular RL algorithms are presented: Monte-Carlo (MC) and Temporal Difference (TD) methods. Also, the Deep Reinforcement Learning (DRL) versions of these algorithms are introduced later in this chapter.

2.1.3.1 Monte-Carlo Methods

Monte-Carlo method in RL is a method of solving the value function. This method can use actual observations from the environment or be based on simulated interaction. The Monte Carlo in terms of RL means that agent uses average of returns of an episode (single run) as an estimate of the rewards. MC methods are used to sample and average state-action pair returns. This means that action taken now and later in the sequence both affect the result of an episode i.e., average returns (Barto & Sutton, 2018). Barto & Sutton (2018) present the Monte Carlo algorithm as:

Algorithm 3. Monte Carlo Control

$$
\begin{aligned}
&\textbf{Initialize}, for\ all\ s\ \in S, a\ \in A(s) \\
&\quad Q(s,a) \leftarrow arbitrary \\
&\quad \pi(s) \leftarrow arbitrary \\
&\quad Returns(s,a) \leftarrow empty\ list \\
&\textbf{Repeat}\ forever\ (or\ number\ of\ episodes) \\
&\quad Choose\ S_0\ \in S\ and\ A_0\ \in A(S_0)\ \text{s.t. all pairs have probability} > 0 \\
&\quad Generate\ an\ episode\ starting\ from\ S_0, A_0, following\ policy\ \pi \\
&\quad \textbf{For each}\ pair\ s,a\ appearing\ in\ the\ episode: \\
&\qquad G\ \leftarrow return\ following\ the\ first\ occurence\ of\ s,a \\
&\qquad Append\ G\ to\ Returns(s,a) \\
&\qquad Q(s,a) \leftarrow average(Returns(s,a)) \\
&\quad \textbf{For each}\ s\ i\ the\ episode: \\
&\qquad \pi(s) \leftarrow\ argmax_a Q(s,a)
\end{aligned}
\tag{15}
$$

MC methods are episodic, which means that they learn after each episode thus the length of the episode must be defined. Next section presents RL algorithms that can learn step by step without waiting to the end of the episode.

2.1.3.2 Temporal Difference Methods: SARSA & Q-Learning

Watkins & Dayan (1992) introduced one of the most popular RL algorithm, Q-Learning. The other popular method is variation of Q-Learning algorithm first called Modified Connectionist Q-Learning, but more popularly known as SARSA (Rummery & Nirajan, 1994). SARSA and Q-Learning are both model-free RL algorithms as they do not require model of the environment as needed in for example MDPs. The main difference to traditional value iteration lies in update function which is based on TD update of the value function $V(s)$:

$$V(s) \leftarrow V(s) + \alpha\big(r + \gamma V(s') - V(s)\big) \tag{16}$$

where $V(s)$ is the value of the old state, $V(s')$ is the value of a new state, α is a step size aka learning rate and γ is a discount rate. For TD update the term $r + \gamma V(s')$ is considered as TD target. Using this TD update it is possible to define it to state-action pair for some policy Q_π.

$$Q\big(S_t, A_t\big) \leftarrow Q\big(S_t, A_t\big) + \alpha\big[R_t + \gamma Q\big(S_{t+1}, A_{t+1}\big) - Q\big(S_t, A_t\big)\big] \tag{17}$$

The advantage of using TD is that the agent can learn step by step rather than episode to episode. This update rule is also the update rule for so-called SARSA algorithm (State-Action-Reward-State-Action) (Rummery & Nirajan, 1994). The whole SARSA algorithm is as follows in equation 18:

Algorithm 4. SARSA & Q-Learning

$$
\begin{aligned}
&\textbf{Select } type\ either\ SARSA\ or\ Q-Learning\\
&\textbf{Initialize } Q, \forall\ s\ \in S, a\ \in A(s), arbitrarily\ and\ Q(terminal-state,\cdot)=0\\
&\textbf{Repeat } (for\ each\ episode):\\
&\quad Initialise\ S\\
&\quad Choose\ A\ from\ S\ using\ policy\ derived\ from\ Q\\
&\quad \textbf{Repeat } (for\ each\ step\ of\ episode):\\
&\quad\quad Take\ aciton\ A, observe\ R, S'\\
&\quad\quad Choose\ A', S' using\ policy\ derived\ from\ Q\\
&\quad\quad \textbf{If } type = Q-Learning:\\
&\quad\quad\quad Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + max_a \gamma Q(S',A') - Q\left(S,A\right)\right]\\
&\quad\quad \textbf{else}:\\
&\quad\quad\quad Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]\\
&\quad\quad S \leftarrow S', A \leftarrow A'\\
&\quad \textbf{Until } S\ is\ terminal
\end{aligned}
\tag{18}
$$

By modifying the TD target of SARSA from $R_t + \gamma Q\left(S_{t+1,}A_{t+1}\right)$ to $R_t + \gamma \max Q\left(S_{t+1,}A_{t+1}\right)$, the result is so-called Q-Learning algorithm (Watkins & Dayan, 1992) The Q-Learning update is shown also in equation 18 which gives the pseudocode for both SARSA and Q-Learning.

Both algorithms are iteratively solved like value iteration. The main difference between these algorithms is that SARSA is on-policy and Q-Learning is off-policy algorithm. The difference between on-policy and off-policy learning is that the on-policy tries to update the policy that is used to generate the decisions made by agent, whereas off-policy means that the agent try to update based on the optimal policy. This means that Q-Learning can make updates based on greedy policy without following a greedy policy (Barto & Sutton, 2018). This makes Q-Learning a good option when applying function approximations like artificial neural networks as it is possible to apply small batch updates sampled from agent's memory.

2.1.4 Function Approximation

The core of the RL problem is estimating some policy π and its corresponding value v_π. The basic RL algorithms work with so-called tabular states, which are discrete and an expectation of a value function is easy to compute. However, in more realistic environments the agent usually must adapt to noisy and continuous environments and state spaces. Therefore, the need for function approximation rises from a need to be able to generalize and map the states and actions to values even in a complex and noisy environment. The function approximation will have a parameterized form with parameters $w \in \mathbb{R}^n$. This then means that the value function is written as $\hat{v}(s, w) \approx v_\pi(s)$. So, this tells us that the expectation of value function v is an approximation function $\hat{v}$ of state s, parametrized with some parameters w. (Barto & Sutton, 2018)

Basically, this means that almost every algorithm used in supervised learning for example Decision Trees, Nearest Neighbors, Neural Networks & Linear Models can be used as function approximation (Barto & Sutton, 2018). However, Barto & Sutton (2018) states that there are some models more suitable to Reinforcement Learning problems than others. One of the most popular function approximation methods for RL must be Neural Networks given that these models has been achieving a lot of success lately. The neural networks are introduced in the next section.

2.1.4.1 Artificial Neural Networks

Artificial Neural Networks are class of machine learning algorithms that mimic the architecture and information flow of human brain. The development of ANN goes way to back to history; earliest and most simple neural networks were developed in 1940s. The basic elements of ANN are neurons that take data as input and then send the combination of input and neural weights and biases through activation function to the next layer or outputs the end results. (Mehrotra, Mohan, & Ranka, 1997). The development around ANNs were rather slow due to lack of computing power but in modern days we have tons of computing power in our basic laptops. This has enabled ANNs to become one of the most popular ML algorithms. Also, the term Deep Learning was born because of the growth of computing power, and thus one can now stack layer after layer to have even deeper models. The ANNs are still not near as same as biological brain which has millions of neurons compared to hundreds or thousands in ANNs. Next, a brief introduction to ANNs is provided.

2.1.4.1.1 Basics of Artificial Neural Networks

ANNs are popular function approximation method for RL. It is based on simple idea of lots of linear functions with non-linear activation functions to pass the information from one layer to another. The layers relate to each other and the connections between layers of neurons are called weights w. The input x of neuron is multiplied with the weights and then they are activated through some function $f(x)$. The activation function is usually a non-linear function such as sigmoid or tanh-function, but it can also be a linear combination of weights and inputs. Before the activation, usually bias term b is added to the linear combination of input x and weights w. The artificial neural network with connections between all previous layer's neurons and current layers neurons is called fully connected neural networks. (Bengio, Goodfellow & Courville, 2015)

Neuron is expressed mathematically as:

$$h = f\left(\sum_{i=0}^{d} w_i x_i + b\right) \tag{19}$$

Where h is the output of the neuron and w are weights, x is the input, d is number of dimensions, b is a bias and $f(\cdot)$ is the activation function (Bengio, Goodfellow & Courville, 2015).

The output of neuron is activated with some function $f(\cdot)$, usually a non-linear function. Below a list of popular activation functions is presented:

Table 1 List of Popular Activation Functions

Name of the Activation	**Formula**
Identity	$f(x) = x$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$
Tanh	$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$
Softplus	$f(x) = \ln(1 + e^{x})$
Softmax	$f(x) = \frac{e^{x_i}}{\sum_n^N e^{x_i}}, for\ i = 1\ to\ N$

Now, a model for neuron and activation functions for output of the neuron is presented. Then one just can add N size of neurons to J number of layers and stack them. The architecture of a simple artificial neural network is presented in Figure 2.

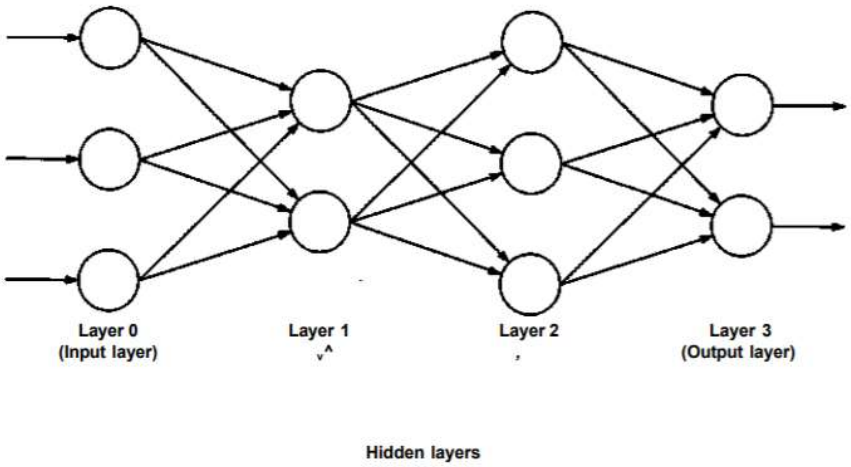

Figure 2 Architecture of feed-forward neural network, Mehrotra, Mohan & Ranka (1997)

If N size of neurons are connected with J number of layers with linear activation function or no activation function at all, the model does not become any better, as combination of linear functions is a linear function. Therefore, the activations are usually non-linear.

ANNs are most often trained with a technique called backpropagation, in which gradients are calculated. The gradients are calculated based on an error function between the target value of a problem and the output of artificial neural network. The algorithm used to calculate the gradients is Gradient Descent algorithm. There are different varieties of algorithms such as Stochastic Gradient Descent (SGD), Adam and RMSProp which are used to calculate and optimize the gradient. The gradients are then sent backwards to the ANN and the weight and bias parameters are then tuned using gradient info. The gradient calculations are defined as:

$$\nabla_{\theta} J\big(g(\theta)\big) = \nabla_{g(\theta)} J\big(g(\theta)\big) \frac{\partial g(\theta)}{\partial \theta} = \frac{\partial E}{\partial W_{jk}^{i}} \tag{20}$$

Where E is the error function and W_{jk}^{i} is weights going from j:th neuron in n:th layer to k:th neuron in i:th layer (Bengio, Goodfellow & Courville, 2015). It possible to then apply chain rule to backpropagate the gradient through whole network. For example, calculating the gradients for three-layer network as:

$$\begin{aligned} Layer\ 3\text{:}\ \frac{\partial E}{\partial W^{(3)}} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W^{(3)}} \\ Layer\ 2\text{:}\ \frac{\partial E}{\partial W^{(2)}} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial W^{(2)}} \\ Layer\ 1\text{:}\ \frac{\partial E}{\partial W^{(1)}} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial W^{(1)}} \end{aligned} \tag{21}$$

In the calculations above E is the error function, $\hat{y}$ is the predicted output of the last layer of artificial neural network, $h^{(i)}$ is outputs of neurons in i:th layer and $W^{(i)}$ is the weights of i:th layer.

In following sections two different types of artificial neural networks are presented. Methods defined in those sections extend the artificial neural networks to work such problems as time-series prediction and computer vision.

2.1.4.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of artificial neural networks that enable the use of grid like inputs. Some examples that can be presented as a grid like input are: in 2-Dimensional convolutional networks, a N-by-N pixels of pictures and for example 1-D example, time series of length N. CNN use convolution of the input instead of the multiplication. To give an example of convolution, suppose that our network outputs a single output $x(t)$ where x is our measurement of the data and t is time. Both variables x and t are real-valued meaning that we might get different measurements depending on the time t. We then want to try to smooth the input with some weighting function $w(a)$ to get an average, where a is the age of the measurement. The convolution can be obtained with weighted average at every time step providing a smooth estimate of s (Bengio, Goodfellow & Courville, 2015).

The convolution can be written as:

$$s(t) = \int x(a)w(t-a)da \\ = s(t) = (x * w)(t) \tag{22}$$

where the x is the input of the model, w is so-called kernel and the output $s(t)$ is called a feature map. Kernel is a multidimensional array of learnable parameters, tensors. Convolution has three crucial aspects that usually improve the learning process in machine learning: sparse interactions, parameter sharing and equivariant representation. (Bengio, Goodfellow & Courville, 2015). Convolution layer is presented in Figure 3.

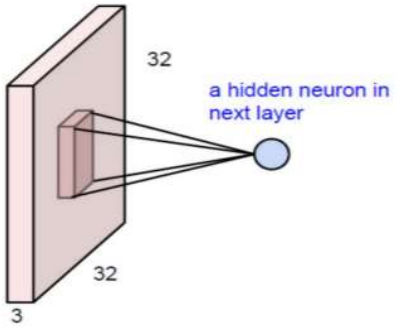

Figure 3 Convolutional Layer, Albami, Mohammed & Al-Zawi (2017)

The Kernel, also known as filter, is rolled over the grids of the input. The number of grids it moves in a step is called a stride (Albawi, Mohammed & Al-Zawi, 2017).

CNN also have a third step after executing the convolution and activation, which is pooling. Pooling means that some of the outputs of the convolutional layer are then replaced by some summary statistic of nearby outputs. Pooling improves computational efficiency of the network because it reduces the number of inputs to the next layer to process. CNNs are also trained with backpropagation presented previously. (Bengio, Goodfellow & Courville, 2015)

CNNs have been widely used in field of RL. One remarkable achievement for it was when Q-Learning with CNN as a function approximator was able to solve Atari game environments.

2.1.4.1.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a type of ANN that are designed to handle time series and sequential data. Instead of having only weight vector W, RNN has U, V & W which are the weights for input-to-hidden, hidden-to-output and hidden-to-hidden. RNN takes inputs in sequentially and stores the information of the past data points in the inner loop in weights of W. The forward pass for simple RNN is:

$$\begin{aligned} a_t &= b + Ws_{t-1} + Ux_t \\ s_t &= f(a_t) \\ o_t &= c + Vs_t \end{aligned} \tag{23}$$

where b and c are both biases, s_t is the hidden output at time t of the sequence, o_t is output and U, V, W are the weights. (Bengio, Goodfellow & Courville, 2015). The process of recurrent neuron is presented in Figure 4.

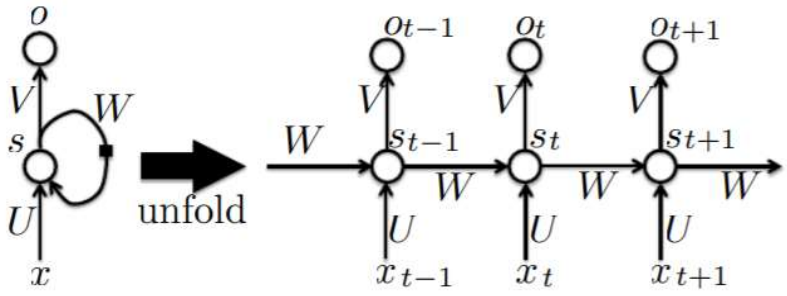

Figure 4 The flow of Recurrent Neural Network, Bengio, Goodfellow & Courville (2015)

These networks are trained with backpropagation through time which is a modification of the previously mentioned backpropagation algorithm (Bengio, Goodfellow & Courville, 2015).

There are different types of recurrent neural networks such as Gated Recurrent Unit (GRU) and Long-Short Term Memory (LSTM) but as the RNN are not used in this thesis, they are not covered any further.

2.1.5 Deep Reinforcement Learning

The previous sections talked about elements of RL and introduced some of the most famous algorithms in solving RL problem. Also, in the previous section ANNs and function

approximation for RL was covered. This section is going to talk about some of the most used combinations of RL and ANN i.e, Deep Reinforcement Learning (DRL). Figure 5 shows the core framework for DRL.

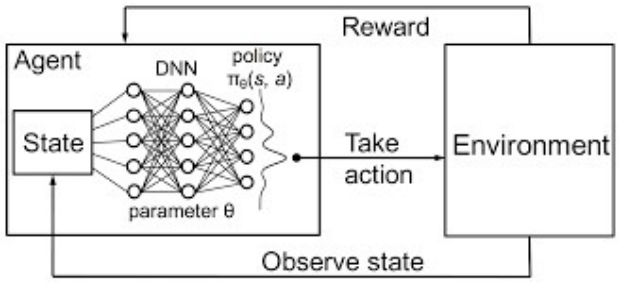

Figure 5 Process of Deep Reinforcement Learning.

One of the most popular DRL algorithms is Deep Q-Network. It is just a regular Q-Learning algorithm integrated with two ANNs, where θ is the agent's parameters for learning network and θ^- is the parameters of the target network used to compute the policy updates. Recall the update rule for Q-Learning:

$$Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',A') - Q(S,A)] \tag{24}$$

In DRL, one must define error (loss or cost) function to be minimized by gradient descent optimization. So, to update policy Q towards the objective which is the TD update $R + \gamma \max_a Q(S',A') - Q(S,A)$, this update is switched into a Mean Squared Error (MSE) loss function to train the networks with gradient:

$$J(\theta) = \left(R + \gamma \max_a Q(S',A',\theta^-) - Q(S,A,\theta)\right)^2$$
$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{25}$$

The trick is also that the target network θ^- is only updated before predefined number of rounds and is fixed for between individual updates. (Mnih, et al., 2015)

This Deep Q-Learning algorithm is then to be implemented into a many different problems. It is also an example of a value based DRL. Next Policy Gradients and Actor-Critics are both introduced as they are also very popular in DRL.

Policy Gradient algorithms use the framework of MC methods instead of TD. It uses episodic samples to update the policy not the value function. The first PG was REINFORCE algorithm proposed by Williams (1992). It uses the MC control framework, but the update of the policy parameters θ ,which in this case are parameters of an ANN, are updated in respect to natural logarithm of the policy probabilities multiplied with returns G (Williams, 1992). The update function for REINFORCE is:

$$\begin{gathered} J(\theta) = \sum_{s \in S} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(a|s)\, Q^{\pi}(s,a) \\ = J(\theta) = Q^{\pi}(s,a) ln \pi_{\theta}(a|s) \\ = J(\theta) = [G_t ln \pi_{\theta}(A_t|S_t)] \\ \theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \end{gathered} \tag{26}$$

That is the gradient regards to error (loss) function that use information from collected rewards to update the gradients of a policy. It is possible to use advantage function to make updates robust against variance:

$$A(s,a) = Q(s,a) - V(s) \tag{27}$$

Both returns G and advantage A act as a baseline for the update (Sutton, McAllester, Singh, & Mansour, 1999). This means that the baseline is guiding the learning process to encourage actions with high positive advantage and discourage actions with high negative advantage. The REINFORCE is the base algorithm to other PG methods such as Actor-Critic models which integrate both worlds having value function with value network and a policy with policy network.

In AC models, instead of using some baseline, for example collected average of collected rewards, the algorithm uses separate approximation for value function. This approximation is then used as an advantage to give better guidance of learning the policy network. The loss function and update rule for value network is:

$$\begin{gathered} J_w(w) = \big(G_t - V_w(s)\big)^2 \\ = J_w(w) = \big(r + V_w(s') - V_w(s)\big)^2 \\ w \leftarrow w + \alpha \nabla_w J(w) \end{gathered} \tag{28}$$

From that it is possible to derive the advantage function:

$$\widehat{A^{\pi}}(s_i, a_i) = r(s_i, a_i) + V_w^{\pi}(s_i') - V_w^{\pi}(s_i) \tag{29}$$

Which then is used in the update rule for the policy net:

$$\begin{aligned} J(\theta) &= A^{\pi}(s, a) \ln \pi_{\theta}(a|s) \\ \theta &\leftarrow \theta + \alpha \nabla_{\theta} J(\theta) \end{aligned} \tag{30}$$

The advantage function presented previously can also be transformed into TD update instead of MC update in the update rule of the value network (Mnih, et al., 2016). The architecture of AC algorithms is demonstrated in the Figure 6.

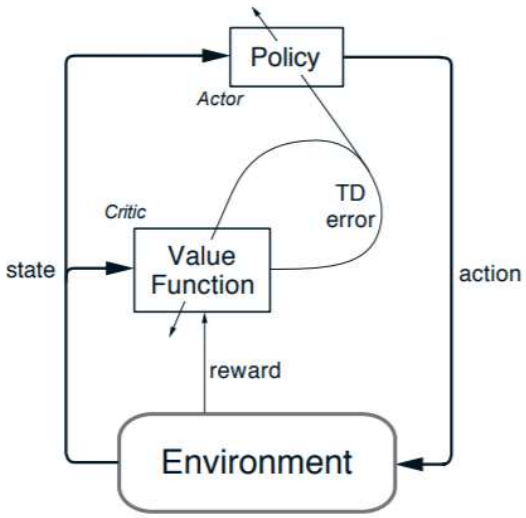

Figure 6 Actor-Critic Architecture, Barto & Sutton (2018)

AC methods have evolved into numerous different variations, but the basic element is still the same; use value network to guide policy network. One great benefit in PG and AC methods is that they can be incorporated into continuous action spaces. Meaning that instead of having a discrete set of possible actions, agent can have set of continuous actions. This can be beneficial in lots of environments such as self-driving cars where steering is not a discrete action, but rather a smooth continuous action. In this study PG based agent with ANN are deployed. However, instead of training the PG agent with gradient descent and

backpropagation, the networks are trained with algorithms that belong to a class of evolutionary algorithms, evolution strategies. That means that agent tries to find optimal policy and network parameters through evolution over generations of parameter-populations. The next topic in this thesis discusses this evolution-based optimization technique and how it can be used in the RL setting.

2.2 Evolution Strategies

This section discusses optimization and search algorithms that are based on biological inspiration. Evolutionary algorithms (EA) date back to the 1960s and their most common application is black-box optimization and search in continuous search spaces. The basic elements of EAs are mutation, recombination, and selection with population-based candidate solution (Hansen, Arnold, & Auger, 2015). The EAs are most often based on elitism and survival of the fittest in Darwinian inspiration. Survival of the fittest means that the best candidate solutions can survive from current population to next generation and elitism means that some number of elites of fittest solutions get selected (Beyer & Schwefel, 2002).

In this study evolution strategies algorithms are used to train and optimize RL agent's behavior and function approximator, ANN. ES as an algorithm gives us an optimization method for parameter space of the RL agent with stochastic sampling of trainable parameters (Hansen, Arnold, & Auger, 2015). This means that the ES injects noise to the best existing solution in order to search the parameter space locally and randomly around it. From the sampled candidate solution, the best solutions are selected and then the process is repeated. In the following sections the purpose is to first introduce the ES algorithm. Then some of the modifications are discussed and finally the ES optimization framework is tied to RL.

2.2.1 Evolution Strategies

As mentioned before, the ES are part of family of EAs. The key components are mutation, recombination and selection, which are iteratively used to find optimal solution for the problem concerned. Next the basics of ES are defined and the whole algorithm is introduced.

The idea behind ES is rather simple, change the parameters all the time and if some of the candidate solutions are better than existing one, then keep it (Beyer & Schwefel, 2002). The first main component of the parameter randomization is mutation. Mutation means that during search, add some amount of noise to the parameters to be solved. The noise can be drawn for example from the normal Gaussian distribution. Most often the Gaussian is used

to add noise, but also other distributions can be used. The noise adding operation can be defined as linear combination of noise (step-size) parameter σ and the noise ε:

$$\begin{aligned} \mu_{new} &= \mathrm{P} + \sigma * \varepsilon \\ where\ \varepsilon &\sim \mathcal{N}(0, \mathbb{I}) \end{aligned} \tag{31}$$

in which μ_{new} is the new generation i.e., population of candidate solutions size of μ, P is current set of parents and σ is the noise or step-size and ε is drawn randomly from standardized normal distribution with zero mean and standard deviation of one. (Hansen, Arnold, & Auger, 2015)

Recombination step selects some number of parents from the previous generation to generate the offspring i.e., population to the next generation of size λ. Usually, multi-recombination is used instead of selecting only one best parent. ES has different recombination operations that can be used. Discrete recombination is also known as uniform crossover, in which single parent is drawn with equal probabilities to all parents in pool ρ after which the noise is then injected to the parameters. Intermediate recombination is based on average values over all the parents in pool ρ. The last recombination introduced is weighted recombination which is a general version of intermediate recombination in which the weight assigned to the parent solution is based on the fitness function (Hansen, Arnold, & Auger, 2015). The fitness function is the objective function to solve the problem for example in RL, cumulative rewards over an episode.

Selection operation is straightforward. Each generation selects some set of parents P based on the performance measured by fitness function $f(x)$. The next generation is then based on this selection of the parents (Hansen, Arnold, & Auger, 2015). By iterating this process of mutation, recombination and selection over and over, the algorithm tries to end up to the maximum of the objective function. ES might suffer from being stuck on local optimum, but this problem can be addressed with altering the step size in the noise parameter σ or making adaptable to the problem at hand. The whole algorithm of ES version $(\mu/\rho^{+}_{,}\lambda)$ is defined as pseudocode:

Algorithm 5. $(\mu/\rho^{+},\lambda) - Evolution\ Strategies$

$$
\begin{aligned}
&\boldsymbol{Given}\ n, \rho, \mu, \lambda \in \mathbb{N}_{+} \\
&\boldsymbol{Initialize}\ \mathrm{P} = \{(x_k, s_k, f(x_k)) | 1 \leq k \leq \mu\} \\
&\boldsymbol{while}\ not\ happy: \\
&\quad \boldsymbol{for}\ k \in \{1, \dots, \lambda\} \\
&\qquad (x_k, s_k) = recombine(select\_mates(\rho, \mathrm{P})) \\
&\qquad s_k \leftarrow mutate\_s(s_k) \\
&\qquad x_k \leftarrow mutate\_x(s_k, x_k) \in \mathbb{R}^n \\
&\quad \mathrm{P} \leftarrow \mathrm{P} \cup \{(x_k, s_k, f(x_k)) | 1 \leq k \leq \mu\} \\
&\quad \mathrm{P} \leftarrow select\_by\_age(\mathrm{P}) \\
&\quad \mathrm{P} \leftarrow select\_\mu\_best(\mu, \mathrm{P})
\end{aligned}
\tag{32}
$$

Over aged individuals are removed in the so-called select by age step. Individual solutions from same generation have the same age (Hansen, Arnold, & Auger, 2015). There are also two different methods of choosing the parents P and producing the offspring. In version of ES $(\mu + \lambda)$ the parents are also included in the next generation and in version (μ, λ) only the offsprings are left in new generation meaning that the parents are rejected (Beyer & Schwefel, 2002). The noise parameter can be either fixed or adaptable. The simplest adaptation of the noise parameter of σ can be defined as:

$$
\sigma^{g+1} \leftarrow \frac{1}{\mathrm{P}} \sqrt{\sum_{i=1}^{\mathrm{P}} \left(\bar{\mu}_i^{g+1} - \mu^{g}\right)^2} \tag{33}
$$

where g is the generation and P is number of parents i.e, set of elite solutions.

Next, variations of basic ES are introduced which use different methods to update the parameters μ and σ.

2.2.2 Covariance Matrix Adaptation and Natural Evolution Strategies

The noise parameter adaptation presented in the previous section suffers from problem that causes the exploration noises σ becoming highly correlated over time. To solve this problem researchers have developed an algorithm that uses covariance matrix to guide the relevant noise i.e., step-size in the search space. This algorithm is called Covariance Matrix

Adaptation Evolution Strategies (CMA-ES). This algorithm introduces new factors: covariance matrix C and two path parameters s_c and s_σ. The s_σ generalizes a path for non-diagonal covariance matrices that are used to implement cumulative step-size adaptation for σ. The s_c disregards σ to accumulate steps for the path of covariance matrix C. The covariance matrix C update consists of rank-one update based on s_c, rank-μ update based μ and with non-zero recombination of weights w_k (Hansen, Arnold, & Auger, 2015). Hansen, Arnold & Auger define the CMA-ES as:

Algorithm 7. Covariance Matrix Adaptation-Evolution Strategies

$$\boldsymbol{given}\ n \in \mathbb{N}_+, \lambda \geq 5, \mu \approx \frac{\lambda}{2}, w_k = \frac{w_{,k}}{\sum_k^\mu w'(k)},$$

$$w'_k = \log\left(\frac{\lambda}{2} + 0.5\right) - logrank(f(x_k)), \mu_w = \frac{1}{\sum_k^\mu {w^2}_k}, c_\sigma = \frac{\mu_w}{n + \mu_w},$$

$$d \approx 1 + \sqrt{\frac{\mu_w}{n}}, c_c \approx \frac{\left(4 + \frac{\mu_w}{n}\right)}{\left(n + 4 + \frac{2\mu_w}{n}\right)}, C_1 \approx \frac{2}{n^2 + \mu_w}, C_\mu \approx \frac{\mu_w}{n^2 + \mu_w}, C_m = 1$$

$$\boldsymbol{initialize}\ s_\sigma = 0, s_c = 0, C = I, \sigma \in \mathbb{R}_+^n, x \in \mathbb{R}^n$$

$\boldsymbol{while}$ *not happy*:

$\boldsymbol{for}\ k = \{1, \dots, \lambda\}$

$z_k = \mathcal{N}(0, I)$

$x_k = x + \sigma C^{\frac{1}{2}} \times z_k$

$\mathrm{P} = select\_best\_\mu(\{(z_k, f(x_k)) | 1 \leq k \leq \lambda\})$ (34)

$x \leftarrow x + c_m \sigma C^{\frac{1}{2}} \sum_{z_k \in \mathrm{P}} w_k z_k$

$s_\sigma \leftarrow (1 - c_\sigma) s_\sigma + \sqrt{c_\sigma (2 - c_\sigma)} \sqrt{\mu_w} \sum_{z_k \in \mathrm{P}} w_k z_k$

$$s_c \leftarrow (1 - c_c) s_c + h_\sigma \sqrt{c_c (2 - c_c)} \sqrt{\mu_w} \sum_{z_k \in \mathrm{P}} w_k C^{\frac{1}{2}} z_k$$

$$\sigma \leftarrow \sigma e^{\frac{c_\sigma}{d}} \left(\frac{\|s_\sigma\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)$$

$$C \leftarrow \left(1 - c_1 + c_h - c_\mu\right) C + c_1 s_c s_c^T + c_\mu \sum_{z_k \in \mathrm{P}} w_k C^{\frac{1}{2}} z_k (C^{\frac{1}{2}} z_k)^T$$

$$where\ h_\sigma = \mathbb{1}_{{\|s_\sigma\|}^2 / n < 2 + 4/(n+1)}, c_h = c_1 (1 - h_\sigma^2) c_c (2 - c_c)$$

$and\ C^{\frac{1}{2}}\ is\ the\ unique\ symmetric\ positive\ definite\ matrix\ obeying\ C^{\frac{1}{2}} \times C^{\frac{1}{2}} = C$

$All\ c - coeffcients\ are \leq 1$

Figure 7 shows how the adaptation of noise (step-size) affects to the exploration in search space if the search space is two dimensional.

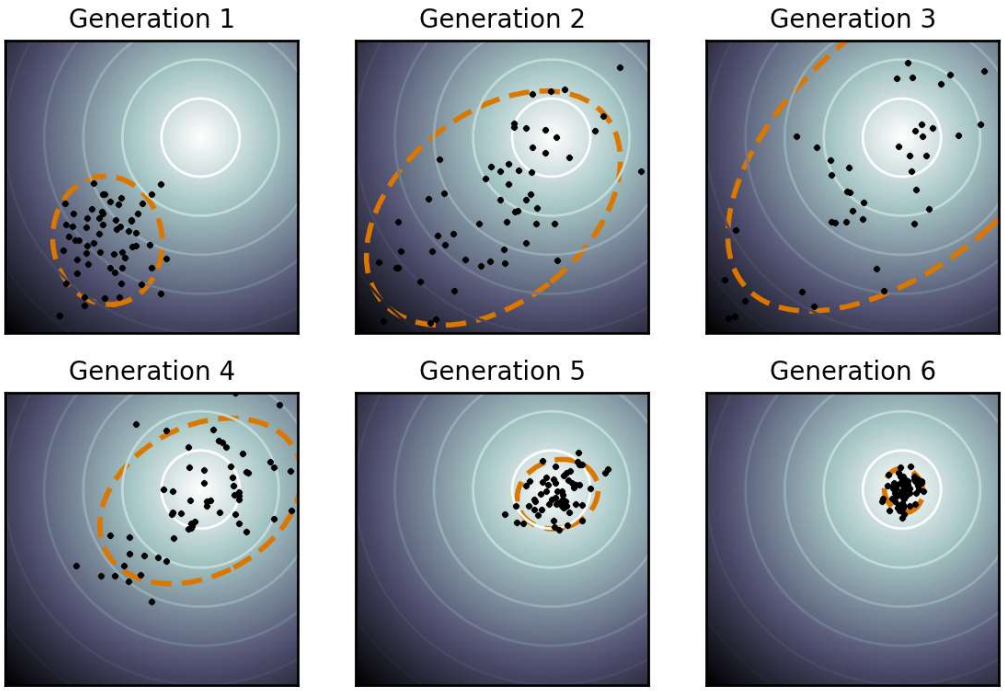

Figure 7 Exploration via Step-size apaptation in CMA-ES

CMA-ES algorithm has analogies to Natural Evolution Strategies (NES), introduced by Wiestra, Schaul & Peters (2008). NES algorithms use parametrized distribution on candidate solutions which are then updated with natural gradient to achieve higher expected fitness. The difference between natural gradients and basic gradients is that instead of euclidean space used in basic gradients, the optimization happens in probability space. This type of operation needs second-order derivatives. Natural gradients were introduced by Amari (1998). The idea of natural gradients is to use Kullbeck-Leibler divergence (KL-divergence) to minimize the distance between approximate distribution and the observation distribution. The KL-divergence is:

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{35}$$

Where P is distribution representing the data and Q is the approximation distribution. To use KL-divergence as an optimization metric, there is a need to estimate curvature of the distribution space. This is given via Fisher Information Matrix F, which is the Hessian for KL-divergence. Hessian is square matrix of second-order partial derivatives which describes curvature (Amari, 1998). Fisher Information Matrix is formulated as:

$$I(\theta) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\log f(X;\theta)\right)^2 \middle| \theta\right]$$
$$= \int \left(\frac{\partial}{\partial\theta}\log f(X;\theta)\right)^2 f(X;\theta)dx \tag{36}$$

Where $f(X;\theta)$ is probability density or mass function for outcome X from probability distribution parameterized with θ.

The connection to CMA-ES is that the Fisher matrix is guiding the learning of our scoring function like the covariance matrix. To incorporate natural gradients in ES, the ES algorithm is modified to use gradient descent step $\nabla_\theta J$ and Fisher information matrix F which are both calculated from fitness values of the sampled new population. Wiestra, Schaul & Peters (2008) defines the Canonical Natural Evolution Strategies algorithm as:

Algorithm 7. Canonical Natural Evolution Strategies

$$\begin{aligned}
&\textbf{input } f, \theta_{init} \\
&\textbf{repeat} \\
&\quad \textbf{for } k = 1, \dots, \lambda \textbf{ do} \\
&\quad\quad draw\ sample\ z_k \sim \pi(\cdot\,|\theta) \\
&\quad\quad evaluate\ the\ fitness\ f(z_k) \\
&\quad\quad calculate\ log-derivatives\ \nabla_\theta \log\pi(z_k|\theta) \\
&\quad \nabla_\theta J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta)\cdot f(z_k) \\
&\quad F \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log\pi(z_k|\theta)\,\nabla_\theta \log\pi(z_k|\theta)^T \\
&\quad \theta \leftarrow \theta + \eta \cdot \boldsymbol{F^{-1}}\nabla_\theta J \\
&\textbf{until } stoping\ criterion\ is\ met
\end{aligned} \tag{37}$$

The benefit of NES algorithm is that it uses not only the elite set of individuals as the update, but the information gained from the whole population of solutions (Wierstra, Schaul, Peters, & Schmidhuber, 2008).

The next section introduces the use of ES in RL and two frameworks that are either similar or derived from the NES algorithm.

2.2.3 Evolution Strategies in Reinforcement Learning

EAs are sometimes used in RL but more often the solutions rely on gradient-based methods. However, EAs and ES have some interesting aspects to apply in RL. First, they are rather simple to implement, especially the basic version of ES. Also, they do not require gradient calculations and backpropagation over the parameter space based on value function or reward function and thus in that way they are gradient-free. Similar to Actor-Critic and Policy Gradients, ES can handle continuous action space with function approximation because all parameters are to be searched. ES also allow more free shaping of reward function as all that matters is the fitness of an individual solution.

One of the most well-known evolutionary algorithms infused RL algorithm is so-called NeuroEvolution of Augmented Topologies (NEAT) developed by Stanley & Miikkulainen (2002). The idea behind that algorithm is that it uses Genetic Algorithm to evolve both topology and parameters in ANN. This algorithm was able to solve continuous action space required by so-called pole-balancing task (Stanley & Miikkulainen, 2002). Igel (2003) then developed an algorithm to that same pole-balancing problem that used CMA-ES with ANN as function approximator. CMA-ES was used to tune parameters of the function approximation algorithm. The use of CMA-ES led to better results than NEAT or other previously developed evolutionary algorithm-based RL algorithms (Neuroevolution algorithms), but stating that the architecture of the network matters and there is also need of algorithms to develop both topology and architecture. Though the algorithm was still able to beat other algorithms with different initializations (Igel, 2003).

After the paper from Wiestra, Schaul & Peters (2008), some of the modifications of NES were developed. Next two sections introduce two different realizations that were used to solve RL problems.

2.2.3.1 Parameter-Exploring Policy Gradients

Parameter-Exploring Policy Gradients (PEPG) algorithm was introduced by Sehnke et. al (2010). The algorithm's purpose was to solve model-free POMDPs and was inspired by PG approach. PEPG tries to learn by estimating likelihood-based gradients with sampling in

parameter space like NES and not policy space like PG. PEPG lead to lower variance gradient estimates than regular policy gradients and was able to learn standing Humanoid problem. It also beat ES based solution in that problem environment. The PEPG comes with two versions; one with regular sampling step as in ES and one with symmetric sampling meaning that each noise (step size) sample ε sampled with noise parameter σ are both added and subtracted from μ. This method also uses the whole population for the update and can adapt the exploration noise i.e., step size on the go (Sehnke, Rückstieß, Graves, Peters, & Schmidhuber, 2010). Sehnke et al. (2010) write the pseudocode for symmetric sampling PEPG algorithm as,

Algorithm 8. Parameter-Exploring Policy Gradient with Symmetric Sampling

$$
\begin{aligned}
&\boldsymbol{initialize}\ \mu\ to\ \mu_{init}\ and\ \ \sigma\ to\ \sigma_{init}\\
&\boldsymbol{while}\ true\ \ \boldsymbol{do}\\
&\quad \boldsymbol{for}\ n = 1, \dots, N\ \boldsymbol{do}\\
&\quad\quad draw\ perturbation\ \varepsilon^n \sim \mathcal{N}(0, I\sigma^2)\\
&\quad\quad \theta^{+,n} = \ \mu + \varepsilon^n\\
&\quad\quad \theta^{-,n} = \ \mu - \varepsilon^n\\
&\quad\quad evaluate\ r^{+,n} = \ r(h(\theta^{+,n}))\\
&\quad\quad evaluate\ r^{-,n} = \ r(h(\theta^{-,n}))\\
&\quad \boldsymbol{end\ for}\\
&\quad \boldsymbol{T} = [t_{ij}]_{ij}\ with\ t_{ij} := \ \varepsilon_i^j\\
&\quad \boldsymbol{S} = [s_{ij}]_{ij}\ with\ s_{ij} := \frac{(\varepsilon_i^j)^2 - \sigma_i^2}{\sigma_i}\\
&\quad \boldsymbol{r}_T = [(r^{+,1}, r^{-,1}), \dots, (r^{+,N}, r^{-,N})]^T\\
&\quad \boldsymbol{r}_S = [(\frac{r^{+,1}, r^{-,1}}{2} - b), \dots, (\frac{r^{+,N}, r^{-,N}}{2} - b)]^T\\
&\quad Update\ \mu = \ \mu + \ \alpha\, \boldsymbol{T}\boldsymbol{r}_T\\
&\quad Update\ \sigma = \ \sigma + \ \beta\, \boldsymbol{S}\boldsymbol{r}_S\\
&\quad Update\ baseline\ b\ accordingly\\
&\boldsymbol{end\ while}
\end{aligned}
\tag{38}
$$

where θ is the model parameters and h is the function approximator. α and β are learning rates for μ and σ respectively.

2.2.3.2 Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Salimans et al. (2017) proposed an ES based RL algorithm which was simplification of the NES algorithm mentioned before. They update the model parameters the same way as in NES by using the information of whole population generated, but they hold exploration noise i.e., step-size constant over training. They were also able to develop highly parallelizable solution of the algorithm meaning that it can be run on multiple different computing cores at parallel. By parallelizing the algorithm was able to solve Atari and MuJoCo environments in fraction of time needed to train other algorithms such as Deep Q-Network and AC algorithms; Advantage and Asynchronous Advantage Actor Critics (A2C & A3C, respectively) with equal level of performance. However, ES methods are not as data efficient as traditional backpropagation methods, which in real-world applications can be issue when there is limited amount of data available (Salimans, Ho, Chen, Sidor, & Sutskever, 2017). Salimans et al. (2017) write this ES as:

Algorithm 9. Evolution Strategies, OpenAI version with no parallelization

Input $Learning\ rate\ \alpha, noise\ \sigma, initial\ policy\ parameters\ \theta_0$

for $t = 0,1,2,..,N$ **do**

$sample\ \varepsilon^n \sim \mathcal{N}(0, I\sigma^2)$

$Compute\ retuns\ F_i = F(\theta + \sigma\varepsilon^n)\ for\ i = 1, \dots, n$ (39)

$Set\ \theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \varepsilon_i$

end for

Onwards from this section this specific implementation of ES algorithm is called OpenAI-ES as the researcher who developed this version of ES algorithm worked at OpenAI which is a research and deployment company that studies different applications for AI.

This concludes algorithmic theory part of this thesis. In this chapter the RL problem and foundations for solving said problem algorithmically was defined and discussed. The use of ANNs and their definitions was covered and some examples of RL incorporating so-called deep learning was also introduced. Lastly the chapter talked about optimization method based on class of evolutionary algorithms: evolution strategies. The idea and framework for optimization with ES was introduced and some advanced applications and connections to RL was discussed. Next the aspects of portfolio management are discussed.

That is going to be the problem environment for this study's implementation of Evolution Strategy based Reinforcement Learning.

3 Financial Portfolio Management

Financial portfolio management (sometimes called asset or investment management) is an interesting topic in the field on finance. In financial portfolio management the purpose is to be able to pick assets for the portfolio and allocate amount invested in each asset to maximize investor's investment objective. The investor who is managing the portfolio must balance with the risk involved in the assets concerned and try to maximize the overall long-term objectives such as maximize returns. The process of financial portfolio management consists of portfolio selection and diversification based on the objective function.

In portfolio selection phase usually some kind of optimization or selection strategy is used. The selection strategies can vary based on the investment objective and risk appetite. In this phase based on the objective and the selection procedure, number of assets are selected to be included in the portfolio.

The portfolio selection strategies usually give us information on what stock to invest in and how the capital-to-be-invested is spread across the assets. This means that the capital is diversified over number of assets instead of betting on only one asset. This leads to diversification in which the risk is minimized by spreading the weights across the portfolio. Diversification has several dimensions such as business sector, geographic, asset class, company sizes, etc.

The management of financial portfolio refers to the methods how the portfolio is managed over time. This involves so-called rebalancing, meaning that in order to select the assets of the portfolio again, re-allocate and diversify them based on the investment objective. There is a huge amount of different management techniques, but the two main categories are passive management and active management. In passive management the investments are fixed and not rebalanced for the chosen investment horizon. In active management the investments are not fixed and can be changed whenever possible (Fabozzi & Markowitz, 2011).

Setting an investment object is typically based on personal preferences and investor's risk appetite. The investment object should be chosen to fit the investor's goals whether it is to make long-term profits or try to exploit the movements of the market in higher-frequency trading. The objectives can be also measured not only by returns but with risk-adjusted returns meaning that the portfolio performance is measured by return against the risk. Markowitz (1952) developed one of the most popular investment objectives that is to maximize returns and minimize variance.

The work by Markowitz (1952) was one of the first works in portfolio selection in which the portfolio selection was based on risk-adjusted measure. The risk in the field of portfolio management is the uncertainty, meaning how uncertain the returns are i.e., how high the standard deviation (volatility) is or how probable losses are. In financial portfolio management the risk should be considered because it will tell the portfolio manager how prone the portfolio is to being exposed to risks and losses. The Mean-Variance model is developed with assumptions of normality, but after the financial market was exposed to some extreme risks and market crashes the need for more robust risk measures has risen such as Conditional Value-at-Risk (CVaR).

In the next sections, some of the main performance measures and risk metrics are introduced. After that some of the portfolio optimization objectives are explained. Lastly, some applications of Reinforcement Learning in Financial Portfolio Management problems are introduced.

3.1 Performance and Risk Measures of Financial Portfolio

The first and the most used performance metric for financial portfolio management is the portfolio return itself. Rate of return r_t of an asset is a relative change in price p between time t and $t-1$.

$$r_t = \frac{p_t}{p_{t-1}} \tag{40}$$

Also, it is possible to use logarithmic returns instead:

$$\ln r_t = \ln \frac{p_t}{p_{t-1}} = \ln p_t - \ln p_{t-1} \tag{41}$$

These both returns illustrate difference between two time periods.

One benefit from computational side is that the logarithmic returns are additive instead of multiplicative nature of the standard returns. If one has some investments in set of assets A in portfolio which weights are w the return of portfolio r^p is then at time t:

$$r_t^p = \sum_{i=1}^{A} r_t^i * w_{t-1}^i \tag{42}$$

The final portfolio p_f value can be then calculated using logarithmic returns as,

$$p_f = c_0 \exp\left(\sum_{t=1}^{t_f+1} \ln r_t^p\right) \tag{43}$$

where c_0 is the initial investment and t_f is the length of trading period.

To measure risk of the portfolio's performance one of the most often used measures is the volatility which defines the uncertainty in returns of some asset. Volatility as a measure is a standard deviation of an asset. Standard deviation σ for a set of samples is,

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{44}$$

where N is the sample size, x_i is individual sample and $\bar{x}$ is the sample mean. The higher the volatility is, the higher the risk is. If volatility is low, then the returns of an asset is very certain (Brooks & Persand, 2003). The problem of volatility as a risk measure is that it does not take account tail-risk, meaning that it does not adapt well to extreme observations in the tail of the return/loss distribution.

Popular metrics to evaluate extreme losses are Value-at-Risk and Conditional Value-at-Risk. Value-at-Risk tells a loss exposure of market value of an asset or a portfolio given time horizon t with some probability level α. (Duffie & Pan, 1997).

$$VaR_\alpha(X) \coloneqq -\inf_{r \in \mathbb{R}} f_X(X) > \alpha \tag{45}$$

Where α is the percentile, $f_X(X)$ is cumulative distribution of sample X. Even though VaR takes account excess losses in some probability level, as a risk measure it ignores the expected value of the loss if the limit is exceeded. In order to be able to take this risk to account, Conditional Value-at-Risk (CVaR) was introduced. CVaR i.e., Expected shortfall, takes account the expected value of the losses that exceeded the VaR limit thus takes account

the fatness of the tail (Acerbi & Tasche, 2002). The formula for Conditional Value-at-Risk is,

$$CVaR^{(\alpha)}(X) = -\frac{1}{\alpha}(\mathbb{E}\left[X1_{\{X\leq x^{(\alpha)}\}}\right] - x^{(\alpha)}(P\left[X \leq x^{(\alpha)}\right] - \alpha) \\ = -\frac{1}{\alpha}\int_0^{\alpha} VaR_{\alpha}(X)d\alpha \tag{46}$$

To conclude this section, couple of Risk-Adjusted Return measures that can be used to evaluate portfolio performance are defined. The first measure to be defined is the so-called Sharpe Ratio and it was derived by William Sharpe (1966). The Sharpe Ratio calculates ratio between expected return versus volatility. Sharpe Ratio is defined as,

$$S_a = \frac{E\left[R_a - R_f\right]}{\sqrt{var[R_a - R_f]}} \tag{47}$$

Where R_a is Return of an asset or a portfolio, R_f is risk-free return for example some treasury bill (Sharpe, 1966). High positive Sharpe ratio means high expected returns and low volatility. Negative Sharpe ratio means that the asset or portfolio has underperformed compared to risk-free return.

The next Risk-Adjusted Return ratio is called Sortino Ratio. The Sortino Ratio developed by Sortino & Price (1994) measures the returns that excess target level divided by downside risk i.e., downside deviation which is standard deviation of returns below the target level. The formula for Sortino Ratio is,

$$S = \frac{R_a - R_t}{\sqrt{\int_{-\infty}^{R_t}(R_t - r_a)^2 f(r_a)dr}} \tag{48}$$

where, R_a is return of an asset, R_t is the target return and in the denominator is downside deviation, in which $f(r_a)$ is the assets return distribution and r_a is random variable of that distribution (Sortino & Price, 1994).

There are number of different measures developed to measure portfolio risk and performance, but Sharpe Ratio remains to be one of the most popular. The measure considered when measuring performance of the portfolio should be able to take in consideration the preferences and risk appetite of individual investor and thus can vary depending on that matter.

Next some of optimization objective i.e., objective functions for solving portfolio selection and weight optimization are presented. In most of them, performance measures and risk metrics presented above apply.

3.2 Optimization Objectives for Financial Portfolio Management

Harry Markowitz (1952) presented his solution to the asset allocation problem which was then widely recognized on portfolio selection and optimization research field. He developed an optimization scheme in which the portfolio was optimized towards minimum variance and maximum expected rewards. This optimization problem can be solved via quadratic programming. By applying this objective, Markowitz (1952) was able to generate so-called efficient frontier by diversifying the weights across different assets. The efficient frontier gives the bounds for return given volatility. The optimization and efficient frontier can be solved in Mean-Variance portfolio formulated as following:

$$\begin{aligned} &Minimize: w^t \Sigma w \\ &s.t\ R^t w = \mu \end{aligned} \tag{49}$$

where variance is given as $w^t \Sigma w$ and expected returns as $R^t w$, where R is vector of expected returns, w is vector of portfolio weights and Σ is the covariance matrix.

As there has been some critique involved in using only volatility as risk measure, similar solutions have been proposed based on different risk measures. One option is to use Conditional Value-at-Risk as the objective function, meaning that the portfolio tries to minimize the tail-loss of the portfolio. With this technique also an efficient frontier for Mean-CVaR portfolio could be derived. (Krokhmal, Palmquist, & Uryasev, 2002).

There are also other types of portfolio management approaches that are not based on risk measures. Some of the portfolio selection and allocation techniques use logic to maximize returns rather than risk measures. For example, two of those logic-based techniques are called Follow-The-Winner and Follow-The-Loser. In Follow-The-Winner

approach the purpose is to put more weights on the assets that are recently performing well. Follow-The-Loser is the opposite of Follow-The-Winner as it tries to put more weight on less successful assets than better performing assets and expecting them to enhance their performance in future (Li & Hoi, 2014). For example, Follow-The-Winner algorithm can be as such: for some time-window from $t - n$ to t assign weights W according to their ranks on average return in that time window.

Again, there are several different optimization objectives and portfolio selection logic developed by different researchers. This thesis focuses on applications that use RL framework. Next section introduces couple of such algorithms.

3.3 Reinforcement Learning applications in Trading and Financial Portfolio Management

Moody & Saffell (1999) developed a method called Recurrent Reinforcement Learning (RRL) that was based on online learning utilizing Q-learning with differentiable Sharpe Ratio as reward function. They used recurrent neural network as a function approximator. The agent was able to execute either long or short position with single asset. According to their studies, the implementation was able to beat buy and hold strategy. (Moody & Saffell, 1999)

Gold (2003) proposed a reinforcement learning agent for Forex (FX) trading based on the Moody & Saffell (1999) RRL agent. They studied high-frequency trading in Forex market based on reinforcement learning for different currency pairs. The study concluded that RRL agent applied in FX market with differentiable Sharpe ratio can be effective traders in the market (Gold, 2003). However, Gold (2003) states that further testing would be needed, and it cannot be claimed that the optimal parameters for the model were found.

Jiang & Liang (2017) proposed a DRL agent for financial portfolio management. Their RL agent was built on both convolution and fully connected settings and it incorporated Policy Gradient framework for learning portfolio weights. The agent's reward function and objective were to gain cumulative returns in trading time. The problem environment for the agent was Poloniex Cryptocurrency market. They stated that the training data for their PG agent should be near to the testing data to perform well due to the so-called expiration problem. They achieved positive results and their algorithm performed well against the benchmark strategies. (Jiang & Liang, 2017)

Later Jiang, Xu & Liang (2017) proposed an improved version of their previous RL agent. The main difference was that they used geometric sampling for the mini-batch

sampling meaning that more recent data is more likely to be selected for training the agent. Also, they developed a framework Ensemble of identical independent evaluators (EIIE) which was based on convolutional artificial neural networks. The idea is that network architecture was to pass each individual asset in the portfolio as an own channel through the network and in the end each asset as an ensemble member has the amount of say in the weighting of activation function conducted by softmax activation. The benefit of this is that the network has shared weights between the channels of network meaning each asset can be used as an input to train the network and share the observations in parallel. They also developed Recurrent version of the same network, but the convolutional version beat the recurrent version in the back tests. The algorithm still incorporated PG algorithm framework and achieved very positive results beating all the benchmarks in cryptocurrency market. (Jiang, Xu, & Liang, 2017)

In footsteps of Liang, Xu & Jiang (2017) reinforcement learning agent for portfolio management learning using adversarial training was proposed by Liang et al. (2018). They proposed a PG based algorithm in which they injected gaussian noise to the price data in process of training to get more robust policies. They also tested other RL frameworks as well in hyperparameter space: Proximal Policy optimization (PPO) and Deep Deterministic Policy Gradients (DDPG). Their implementation also used CNN as function approximator and EIIE framework developed by Liang, Xu & Jiang (2017). As a result, they stated that the algorithm was able to beat benchmark investments in Chinese stock market. They also concluded that CNN networks beat the recurrent version in this portfolio management problem. Based on their findings it was also said that even though they got positive results the reinforcement learning algorithms are still not performing as well as in environments such as games. (Liang, Chen, Zhu, Jiang, & Li, 2018)

Huang (2018) studied financial trading with Deep RL based on recurrent neural network. The study applied Online Recurrent Q-Learning Agent with Long-Short Term Memory neural network. The agent was able to execute either long or short positions as an action. The study was based on FX data and different currency pairs. The study utilized smaller replay memory, sampled longer sequences and did not train for every time step in the data (Huang, 2018). The study of Huang (2018) stated four main achievements: MDP model for signal-based trading that is flexible to future extensions with minimal modifications, modification on deep recurrent Q-learning to be more suitable for financial trading, achieving positive trading results based on 12 currency pairs and finding that slightly increased spread led to better performance.

Zhang, Zohren & Roberts (2020) applied DRL to financial trading for futures contracts and different asset classes such as forex, commodities & equities. They considered both discrete and continuous action spaces, but the actions in both cases was -1 to short position, 0 do nothing & 1 long position. In the continuous action spaces the action could be something between -1 and 1. The study compared the DRL method to traditional time series momentum strategies and showed that DRL method outperformed such baseline models. Their implementation followed large market trends without switching position frequently. They also compared different DRL methods like Deep Q-Learning (DQN), Advantage Actor-Critic (A2C) and Policy Gradient (PG), where DQN obtained the best performance. (Zhang, Zohren, & Roberts, 2020)

Based on the previous studies, single asset trading i.e., long-short strategies are more studied than the actual portfolio management. In single asset trading deep q-learning methods are more popular than actor-critic and policy gradients. In portfolio management policy gradients and deep deterministic policy gradients have been applied.

This concludes the theory part of this thesis. First the RL and its theories was introduced in chapter two. Then ES algorithms were defined as an optimizer to find optimal policies in RL. Then in chapter three. the financial portfolio management was defined, and some key metrics and methods were introduced. Last, the connection between RL and financial portfolio management was created by discussing the applications in trading and portfolio management with DRL and PG in financial market settings. Next chapter introduces the empirical study methods in which the Evolution Strategies based Reinforcement Learning agent and its application to portfolio management problem is defined.

4 Empirical Study

This empirical study implements financial portfolio manager agent which is implemented using Evolution Strategies based Reinforcement Learning. First the problem setup is given as it is necessary to give definitions of what are states, actions and observation in this specific implementation. Next, the choices for the algorithms are made for the RL framework, function approximation, how the learning is optimized and what are the objectives that the implemented agent must try to overcome. Next the selected assets are defined, and their features are briefly introduced. Also, data processing is described in more detail. Last, the agent's algorithm is defined with introduction on how the agent is trained and evaluated.

4.1 Financial Portfolio Management as a Reinforcement Learning Problem Environment

This next section describes the problem environment of this thesis. The needed components for MDP formulation for RL problems are state space, action space and observation space.

4.1.1 States

State space is a representation of the environment given a moment t in time that is given to the agent as an input when considering the next action a. In financial portfolio management problem, the state space can for example be the prices of assets presented to an agent. Also, the state space can include technical indicators of the assets, such as moving averages or social media data such as Twitter tweets. The state space can also be a sequence of changes in prices given a time window.

The implementation of this study implements historical price data as a state space for the learning agent. Like in the study of Jiang, Xu & Liang (2017), the agent gets a state space as an input of Close, High and Low prices of assets for some time window $t - n$ to t, where n is the window size. To stabilize the learning of an artificial neural network, the state space needs to be normalized. This study uses the same normalization schema for the raw input data as in Jiang, Xu & Liang (2017) and the normalization is defined for each price's column as,

$$V_t^{Close} = \left[\frac{v_{t-n+1}^{Close}}{v_t^{Close}}, \frac{v_{t-n+2}^{Close}}{v_t^{Close}}, \dots \frac{v_{t-1}^{Close}}{v_t^{Close}}, 1\right]$$
$$V_t^{High} = \left[\frac{v_{t-n+1}^{High}}{v_t^{Clos}}, \frac{v_{t-n+2}^{High}}{v_t^{Close}}, \dots \frac{v_{t-1}^{High}}{v_t^{Close}}, \frac{v_t^{Hig}}{v_t^{Close}}\right] \quad (50)$$
$$V_t^{Low} = \left[\frac{v_{t-n+1}^{Low}}{v_t^{Close}}, \frac{v_{t-n+2}^{Low}}{v_t^{Close}}, \dots \frac{v_{t-1}^{Low}}{v_t^{Close}}, \frac{v_t^{Low}}{v_t^{Close}}\right]$$

The input X_t vectors is size of $f * m * n$, where f is number of features (in this study 3), m is number of assets (in this study 10) assets and n time steps (which in study is chosen to be 50 timesteps as in study of Jiang, Xu & Liang (2017)). This normalization transforms the price data to relative changes related to current day's closing price, this matrix is then given as a state input for an agent. The shape of input data can be seen in Figure 8.

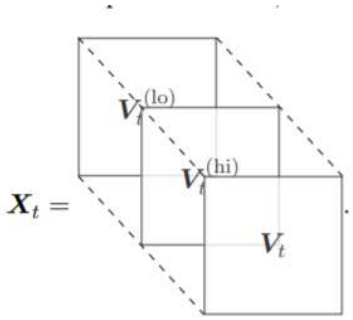

Figure 8 Dimensions of State Space, Jiang, Xu & Liang (2017)

The implementation of the state space is rather straightforward and lets the ANN explore the phenomena in the prices rather than injecting any predefined indicators. Next the agents action space is defined.

4.1.2 Actions

In order to be able to interact with some policy π in the state space, the agent needs actions a to be defined. If the agent was trained to select whether some asset is selected to portfolio, it will be possible to use some set discrete actions like in Q-Learning, in which each state-action pair is presented with Q-value. But, because there is a need to optimally allocate the portfolio weights to optimize the objective function, the need for continuous actions arises. In this study the action space is a vector of weights w_t^a at given time step t for asset a. The

weight is a continuous value between zero and one. The sum of all weights in time step is equal to one, meaning that 100 % of the available capital is invested in assets.

4.1.3 Observations

The observations in financial portfolio management are the changes in the prices of assets. Meaning, that in time t the agent obtains a vector of relative changes x_t in prices of the assets p_t, executes action $a_t = w_t$ and observes new vector of prices p_{t+1}. The relative change y_t is the relative change of prices between these two timesteps.

$$y_t = \frac{p_{t+1}}{p_t} \tag{51}$$

Then the agent updates the time t to be $t + 1$ and with this is possible to define that the return for the portfolio manager agent y is the logarithmic rate of return weighted by the portfolio weights vector w i.e., action.

$$\ln y_t^p = \ln(y_t \cdot w_{t-1}) \tag{52}$$

The logarithmic returns are then used to construct the reward functions which are defined later.

Next the choices for algorithmic implementation are presented. Also, the objectives and reward functions are formulated and then the assumptions according to the implementation are defined.

4.2 Implementation and Assumptions

Next the algorithmic framework for the implementation is presented. Also, the objective and reward functions are introduced. Last, the limitations and assumptions are defined.

4.2.1 Selection of a Reinforcement Learning Framework

This study implements Policy Gradient methods with some modifications as the main framework for the financial portfolio management problem. PG has been also used in other studies that have been conducted to the financial portfolio management problem. The reason for selection of PG framework is simple, as there is a need to be able to use continuous

actions and use the policy instead of a Q-values or a value function. The training is conducted in Monte Carlo method fashion, meaning training after each episode with averaged rewards.

But instead of using gradient info of the policy and backpropagation, the ES are implemented as optimizer for the agent's function approximator. This means that the implementation of PG is gradient-free in terms of state-action-reward and the information of fitness function is used to train the population instead. The main reasons to implement ES instead of traditional backpropagation is listed as follows:

1. Easy and simple implementation
2. No need for backpropagation
3. Flexible reward function engineering
4. Possible help on vanishing and exploding gradient issue in time series data.
5. Parameter level exploration
6. Parallelization

ES is rather simple to implement as it only needs the feedforward and fitness evaluation phase, and it skips the traditional and sometimes heavy backpropagation phase. The ES also allows the use of non differentiable reward functions that might be customized for specific problems. However, in this study the mean logarithmic daily return can also be used in traditional backpropagation methods. ES might also help in the vanishing and exploding gradient problem as it does not send the gradient information back to layers of parameters but instead uses the fitness information and sampled parameters to improve the performance. One main benefit of ES is to solve the exploration-exploitation dilemma with the in-built feature of sampling the model parameters. This leads to parameter level exploration and there is not needing to implement any separate exploration-exploitation scheme in training phase. The last feature of ES is the parallelization. This means that the algorithm can be utilized with parallelized computing and the computation can be sent to multiple different CPU's. However, in this study this feature is not utilized because of the implementation is kept as simple as possible in order to serve the purpose of the study. The versions of ES algorithms used in this thesis are presented later. Next the choice of function approximator is presented.

4.2.2 Function Approximator

As a function approximator to the financial portfolio management agent, the convolutional artificial neural network is implemented. The reason for selecting convolutional neural networks was the need to incorporate sequence of time series as state representation. Also studies of Jiang & Liang (2017), Jiang, Xu & Liang (2017) & Liang et al. (2018) presented that CNN most often were a better function approximator than RNN or standard ANN. Next the topology of an ANN used by the agent is presented.

4.2.2.1 Chosen Artificial Neural Network Topology

The topology for the financial portfolio agent is based on the topology developed by Jiang, Xu & Liang (2017), so-called Ensemble of identical independent evaluators. It consists of convolutional layers in which each feature (Close, High & Low) is fed to the network in their own channels and each asset is evaluated independently with shared weights. (Jiang, Xu, & Liang, 2017). The reason for selecting this topology is that based on the studies of Jiang, Xu & Liang (2017) & Liang et al. (2018), this topology showed some positive results on portfolio management task, and because the idea behind the solution is quite intuitive for example having shared weights i.e., use experiences from all assets to evaluate single asset in portfolio.

The implementation of this study consists of two convolutional layers, whereas Jiang, Xu & Liang (2017) had four. The reason for that is that instead of having a separate input for previous weights, the idea is to let ES adapt to the problem. The idea behind having the previous weights incorporated in the model was to reduce transaction costs. Also having shared weights across each asset reduces the number of trainable parameters, which might be helpful for faster convergence and learning. The first layer applies 1 to 3 convolutions, meaning it calculates 3-day weighted smoothing for each asset, which gives 48 smoothed values as feature map in the second layer and after the convolution Rectified Linear Unit activation is applied. The stride time in the convolution is one, meaning the convolution moves one step at each convolution. The next layer uses 1 to 48 convolutions, meaning that based on the smoothed histories computed in previous layer, the output of this layer calculates a feature map of 10 asset logits for the output i.e., the amount of say for each asset to allocation. After the second layer, a softmax activation is conducted which gives the portfolio weights w for each state s. Softmax also ensures that all the weights are between 0 and 1 and the sum of the weights is equal to one. The Network Topology of modified EIIE is presented in Figure 9.

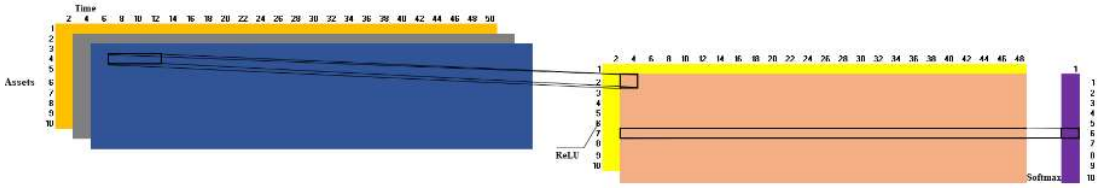

Figure 9 Implemented Network Topology

Next, the algorithms which will be used to train the network are presented.

4.2.2.2 Chosen Parameter and Policy Optimizers

In this study couple of optimizers based on ES are chosen to be tested to train the agent. The first optimizer to be tested is ES developed by OpenAI researcher's Salimans et al. (2017) OpenAI-ES. This is an implementation of natural evolution strategies with fixed exploration noise σ and uses the whole population to update the parameters θ.

The second algorithm adopts same ideas as OpenAI-ES and adapts the exploration noise σ while learning. This is called the Parameter-Exploring Policy Gradient, PEPG. All these algorithms are defined and introduced in section 2.2.

4.2.3 Objectives and Reward Functions

Next the objective function i.e., reward function that is to be tested is defined. The basis of reward function G is simply maximum mean logarithmic return, which is defined as,

$$G = \frac{1}{N}\sum_{t=1}^{N} r_t \tag{53}$$

which also corresponds to maximum cumulative rewards and r_t is given same way as in formula 41. There is no discount factor γ as it is assumed here to be one, meaning that all the future rewards are as important as immediate rewards.

4.2.4 Trading Frequency, Assumptions, and Limitations of the Environment

There are seven main assumptions considering the implementation and the environment:

1. It is assumed that the market has high liquidity, meaning that the agent can execute the trades for assets with the given closing price.
2. The amount invested by the agent is so small that it does not have any market impact.
3. There are no transaction costs involved.
4. As a price information, adjusted closing price is used as a trading price.
5. The agent has access to only one market at time.
6. Trading is conducted at maximum on daily basis.
7. No short selling is allowed.

In real world these assumptions usually do not hold strongly, as there is some real-time movement in the prices of assets. If considering non-institutional investor, then the assumption number two. holds, because the amounts invested by regular investor is rather small. Also, if the volume of an asset is high, then this will also hold. But if there is some behavior that causes crowds of investors to act the same then there will be an impact. However, these are assumed to be integrated in historical price data. Also, in real life there are transaction costs involved, but they usually vary depending on which type of investor the agent is. If rather normal individual investor, then transaction costs really have impact on trading action and if institutional then it would be reasonable to assume that the transaction costs are no harm if the trading strategy is good. The trading frequency is one day, but whether to change weights once a day or more rarely, is left to the agent to decide.

One limitation to the environment is that it does not offer the agent all the possible assets to be invested in. This limitation is made because of the computation time. In order to serve the purpose of this study the computation time must be kept reasonable and thus is limited to include ten assets per market.

4.2.5 Tools for the implementation

The algorithm for the RL agent is done by using Python coding language. For CNN, the PyTorch library was used to implement the feed-forward aspect of the process. The ES are coded from scratch based on those pseudocodes given in section 2.2. The agent is trained on CPU and is not parallelized. The parallelized implementation would speed up the training process and should be considered in the future developments of the algorithms coded for purposes of this thesis.

4.3 Data and preprocessing

Next, the data, data gathering, and processing are described. Also, the descriptions for all assets and their descriptive statistics are given.

4.3.1 Asset selection and descriptions

There are two different markets that are given to the learning agent to be solved. First portfolio consists of ten Exchange-Traded-Funds (ETFs) from Pacific Exchange (PCX). The second portfolio includes ten stocks from Finnish Stock Exchange i.e., Helsinki Stock Exchange (OMX Helsinki).

ETFs are like other funds, but they can be bought and sold in stock-exchange-like manner. The reason to choose ETFs as one example portfolio is that there are several different funds with different investment classes and preferences. For example, one ETF can incorporate investing based on small corporates, while the other can invest in commodities like wood or gold and other to bonds or treasury bills etc. The ETFs chosen to this study are picked in a manner that each ETF has some unique property and are somewhat different to the others. The list of ten chosen ETFs is given in Table 2. Also, the descriptive statistics for the stocks are given in the same table. In Table 2. The best performing asset in each metric is bolded.

Table 2 Descriptive statistics and class descriptions for PCX assets 2012-2020

Tick	Expected Daily Log-Return	Daily Volatility	Annual Sharpe Ratio	Daily VaR 90 %	Daily CVaR 90 %	Description
IXJ	0.051 %	0.984 %	0.898	-0.999 %	-1.750 %	Tracks S&P 1200 HealthCare IndexTM
FAN	0.056 %	1.273 %	0.801	-1.313 %	-2.256 %	Tracks ISE Clean Edge Global Wind EnergyTM Index
FXE	-0.006 %	0.501 %	-0.135	-0.605 %	-0.899 %	Tracks USD-EUR Currency rates
LQD	0.022 %	**0.459 %**	0.797	**-0.375 %**	**-0.698 %**	Tracks investment grade Corporate Bonds
TLT	0.022 %	0.862 %	0.476	-1.011 %	-1.528 %	Tracks 20 + US Treasury Bond
GLD	0.006 %	0.961 %	0.174	-1.052 %	-1.764 %	Tracks prices of Gold
IYR	0.032 %	1.193 %	0.525	-1.126 %	-2.082 %	Tracks Dow Jones U.S Real Estate Index
VV	**0.057 %**	1.052 %	**0.937**	-0.919 %	-1.886 %	Tracks CRSP US Large Cap Index
VB	0.051 %	1.210 %	0.761	-1.192 %	-2.168 %	Tracks Small Cap Indeces
CUT	0.038 %	1.247 %	0.583	-1.278 %	-2.234 %	Tracks MSCI ACWI IMI Timber Select Capped Index

The second portfolio consists of only risky assets from Finnish stock exchange. The purpose is to see whether the agent can survive with only risky assets. To diversify the portfolio, ten assets chosen to be included are picked from different business sectors. The descriptions of the stocks are given in Table 3 and their descriptive statistics are given in same table. In the Table 3. The best performing asset in each metric is bolded.

Table 3 Descriptive Statistics and Industry Descriptions for OMX Helsinki assets 2012-2020

Ticks	Expected Daily Log-Return	Daily Volatility	Annual Sharpe Ratio	Daily VaR 90 %	Daily CVaR 90 %	Industry
YIT	-0.014 %	2.174 %	0.073	-2.342 %	-4.071 %	Engineering & Construction
KNEBV	0.069 %	1.429 %	0.878	-1.524 %	-2.549 %	Special Industrial Machinery
ATRAV	0.036 %	1.589 %	0.487	-1.721 %	-2.832 %	Packaged Food
LAT1V	0.030 %	**1.396 %**	0.457	-1.513 %	-2.556 %	Waste Management
FSKRS	0.034 %	1.453 %	0.491	-1.451 %	-2.430 %	Home Improvement Retail
SAMPO	0.051 %	1.441 %	0.676	**-1.328 %**	**-2.403 %**	Insurance Diversified
WRT1V	0.032 %	1.979 %	0.412	-1.979 %	-3.477 %	Special Industrial Machinery
NESTE	**0.170 %**	2.175 %	**1.410**	-2.049 %	-3.554 %	Oil & Gas Refining & Marketing
OLVAS	0.063 %	**1.396 %**	0.827	-1.506 %	-2.465 %	Beverages-Brewers
TIETO	0.058 %	1.608 %	0.698	-1.688 %	-2.765 %	Information Technology Services

The decision whether the agent ever invests in some of the assets is left to the agent to decide. The number of assets is limited to ten in order to keep the computational time reasonable for the purposes of this study. In real world applications, it would be reasonable to offer the agent as many assets to be evaluated as possible.

4.3.2 Data gathering and processing

The data is gathered from Yahoo Finance: finance.yahoo.com. The gathering is done by using the API (application programming interface) offered by Yahoo, as Yahoo Finance offers the whole history of prices of the assets. The connection is accessed from Python environment by using pandas_datareader library which then downloads the data to Pandas dataframe format.

The data is then processed. First it gets checked for any missing values. If there are any, then the price of the day before is used. Next the dates and prices are formatted and normalized to be like the state space represented in 4.1.1.

4.4 Training, Testing and Performance Evaluation of the Agent

4.4.1 Training and Testing process

The training is conducted iteratively for 10 000 generations. The training period i.e., in-sample period of the assets starts from 1.1.2012 and ends in 31.12.2017, which means that there are six years of training data. Also, from the first day of the training period the lookback window is 50 days that are presented as an input to the agent. Figure 10. shows the performance of each single asset in the portfolio if one buys the asset at first day of in-sample period and holds it until the end of the period.

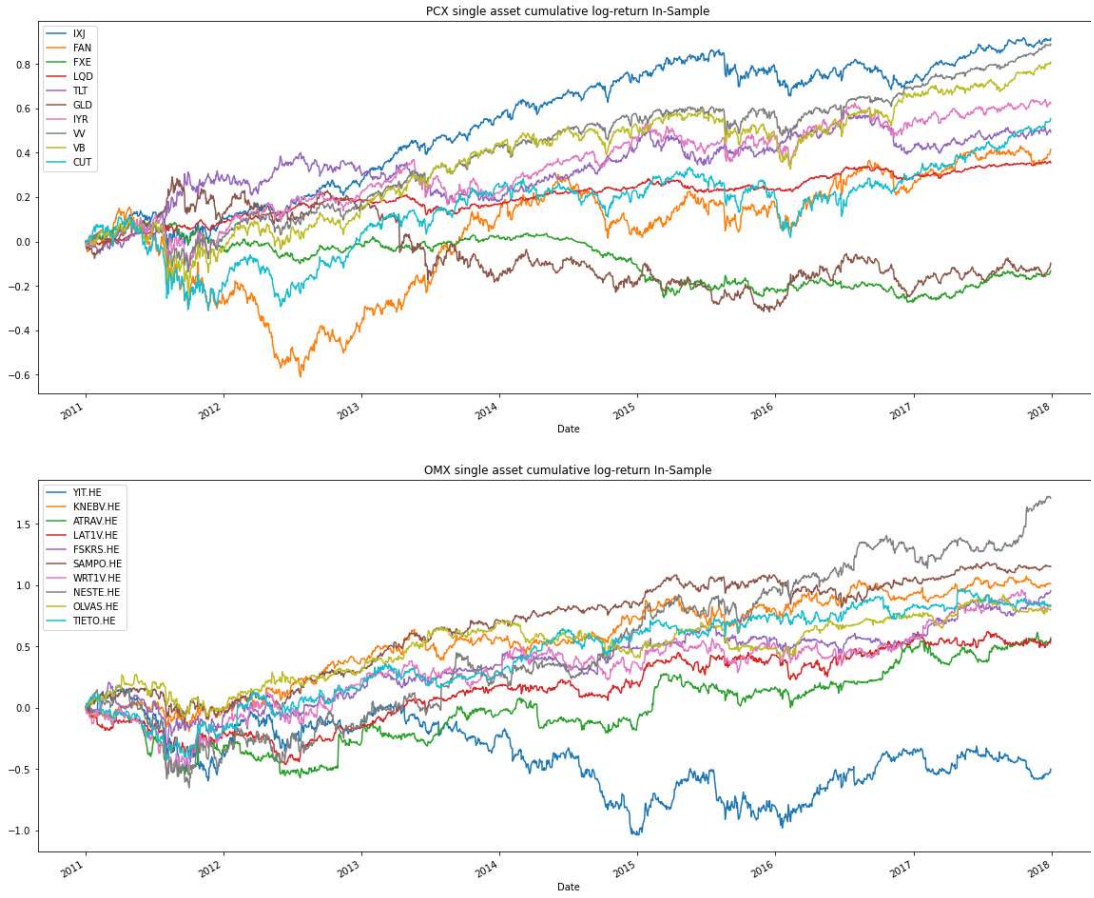

Figure 10 Performance single assets of PCX and OMX Helsinki assets measured in logarithmic return in In-sample period

For the testing period i.e., Out-of-sample period, the agent executes the learned policy π. The out-of-sample period for assets starts from 1.1.2018 and ends at 31.12.2020. The out-of-sample includes COVID-19 outbreak which starts around March 2020. The idea to

include the COVID-19 period is to see how the agent manages to do in exceptional market conditions. Figure 11. shows the performance of each single asset in the portfolio if one buys the asset at first day of out-of-sample period and holds it until the end of the period.

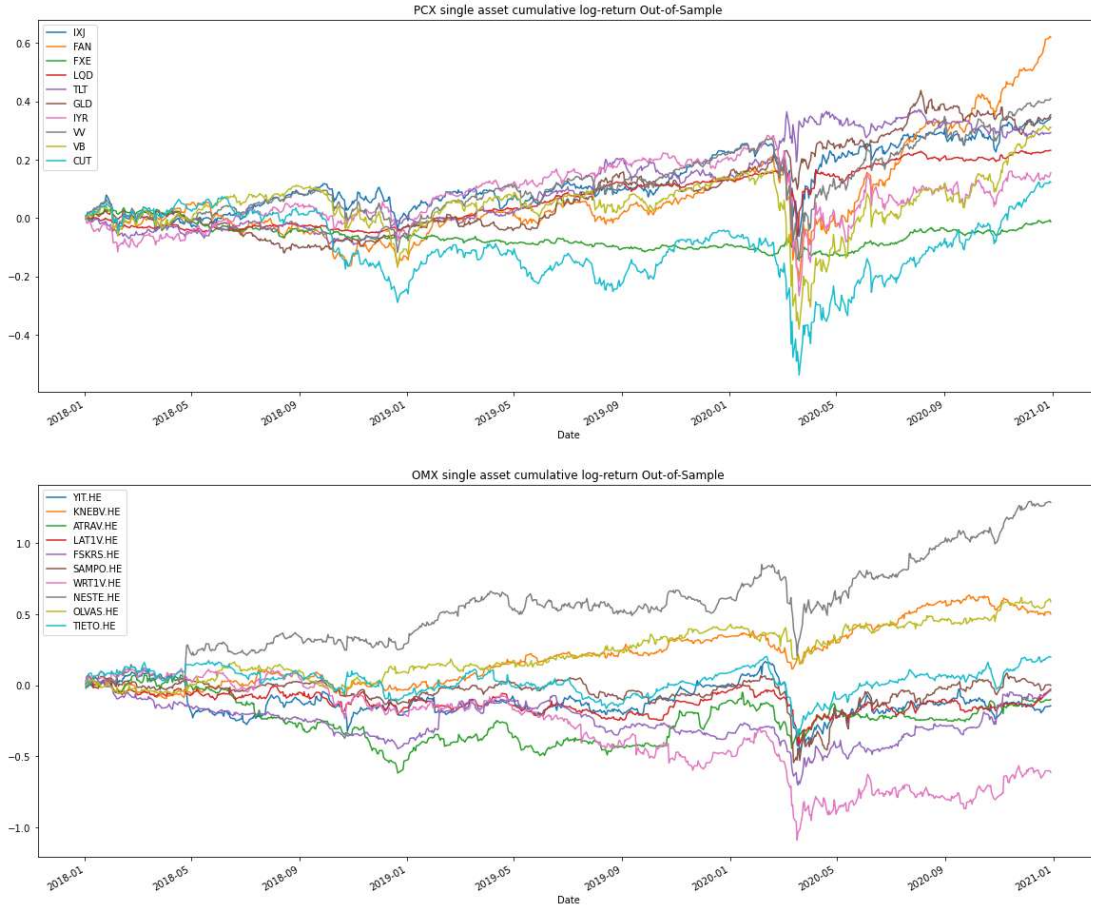

Figure 11 Performance single assets of PCX and OMX Helsinki assets measured in logarithmic return in Out-of-sample period

The pseudocode for the learning algorithm is given as:

Algorithm 9. Framework for training Evolution Strategies based Portfolio Management Agent

$$
\begin{aligned}
&Select\ h\ as\ function\ approximator\\
&Initialize\ \mu\ and\ \sigma\ for\ h\\
&Select\ size\ of\ population\ P\\
&Select\ number\ of\ generations\ N\\
&Initialize\ reward\ array\ R\\
&Select\ Evolution\ Strategies\ optimizer\ u(x)\\
&Select\ Reward/Fitness\ function\ f(x)\\
&Select\ assets\ A\text{: }input\ X, and\ prices\ Y\\
&\quad For\ each\ n\ in\ N\\
&\qquad For\ each\ p\ in\ P\\
&\qquad\quad draw\ noise\ \varepsilon \sim N(0, I)\\
&\qquad\quad \theta = \mu + \sigma\varepsilon\\
&\qquad\quad W = h(X, \theta)\\
&\qquad\quad r = evaluate\ according\ to\ f(Y, W)\\
&\qquad\quad append\ r\ to\ R\\
&\qquad \mu = Update\ according\ to\ u(R, \mu, \theta)\\
&\qquad \sigma = Update\ according\ to\ u(R, \sigma, \theta)
\end{aligned}
\tag{54}
$$

The hyperparameters i.e., noise, learning rate and population size is tested in order to see how they affect the learning and performance of the algorithm. Also, to speed up the training and reduce overfitting, mini-batch updates are used. Mini-batch implemented in this study means that each generation is trained over random sample of the full batch. The mini-batch shuffles the training data and reduces correlation between data samples which should lead more stable learning overall, but causes fluctuations to the objective functions as the updates are stochastic. For a mini batch 20 % of the in-sample data is sampled to make an update based on fitness function.

4.4.2 Benchmarks and Evaluation Metrics for testing

As a benchmark to evaluate the performance of the portfolio management agent, a dynamic, equally weighted portfolio is built. This means that the weight of each asset is set equally each timestep. Also, as a benchmark, performance of related index is given. In case of PCX the SPY (S&P 500 index ETF) is the benchmark for index and in case of OMX Helsinki the OMXHPI index is used as a benchmark.

All results of each algorithm alternatives are compared using the mean daily logarithmic return, daily standard deviation, annual sharpe ratio, daily VaR and daily CVaR to other agents. Also, benchmarks are evaluated based on the same metrics.

Also, the algorithms are compared on basis of generations taken to maximize the reward function, meaning how fast they learn and how the training affects the out-of-sample performance.

4.5 In-Sample Learning Curves & Out-of-Sample Performances

This section presents the learning curves for the portfolio management agents. The learning curves give information on how well the agents learn in-sample. Also, the out-of-sample performance for different hyperparameters are looked at. Based on the convergence information given the learning curves and the out-of-sample performances, the best performing hyperparameters for both agent types are selected and they are later compared against benchmarks in chapter 5.

4.5.1 PCX Portfolio

Several different combinations of hyperparameters were tested for both algorithm types. For OpenAI-ES, different learning rates, exploration noises and population sizes were tested. For PEPG the exploration noise was not varied because the algorithm adapts the exploration noise while learning. The chosen hyperparameters for OpenAI-ES and PEPG are learning rate 0.005 and population size 50 and only for OpenAI-ES exploration noise 0.01. All the learning curves for in-sample and performance tables for out-of-sample can be found in appendices and thus only the selected hyperparameter settings are presented here.

The learning curves for both OpenAI-ES and PEPG agents are shown in Figure 12.

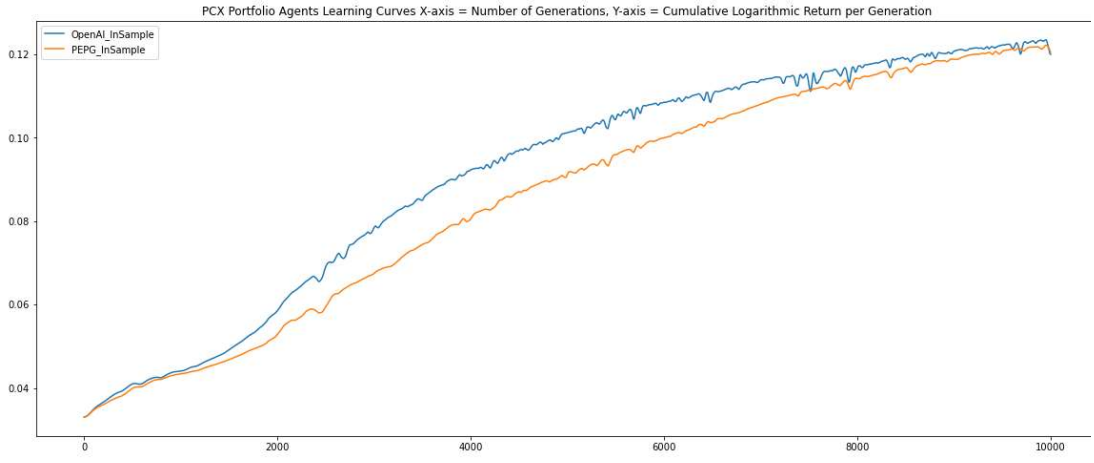

Figure 12 In-Sample learning curves of PCX Portfolio Management Agents

Learning curves show the learning process in terms of reward function, expected logarithmic daily return. The figure shows that both agents are learning in-sample. Because of the use of mini-batch, the chance of overfitting can be reduced as the updates are sampled in random order from the full batch which also reduces correlation between updates. The in and out-of-sample performance for both agents can be shown in the Table 4. Both agents end up in a similar in-sample and out-of-sample performance.

Table 4 Mean Daily Logarithmic Return for Both Algorithm in PCX Portfolio

Algorithm	In/Out-of-Sample	Mean daily logarithmic return
OpenAI-ES	In-Sample	**0.120 %**
	Out-of-Sample	**0.144 %**
PEPG-ES	In-Sample	**0.120 %**
	Out-of-Sample	**0.144 %**

4.5.2 OMX Helsinki Portfolio

Similarly, to the training for portfolio agents in PCX portfolio, different set of hyperparameters were also tested. The learning curves for each tested set of hyperparameters can be found in appendices. The selected hyperparameters for OpenAI-ES are learning rate 0.0025, exploration noise 0.1 and population size 150. For PEPG, learning rate is set to 0.01 and population size 50. The learning curves for both algorithms with selected hyperparameters are shown in Figure 13.

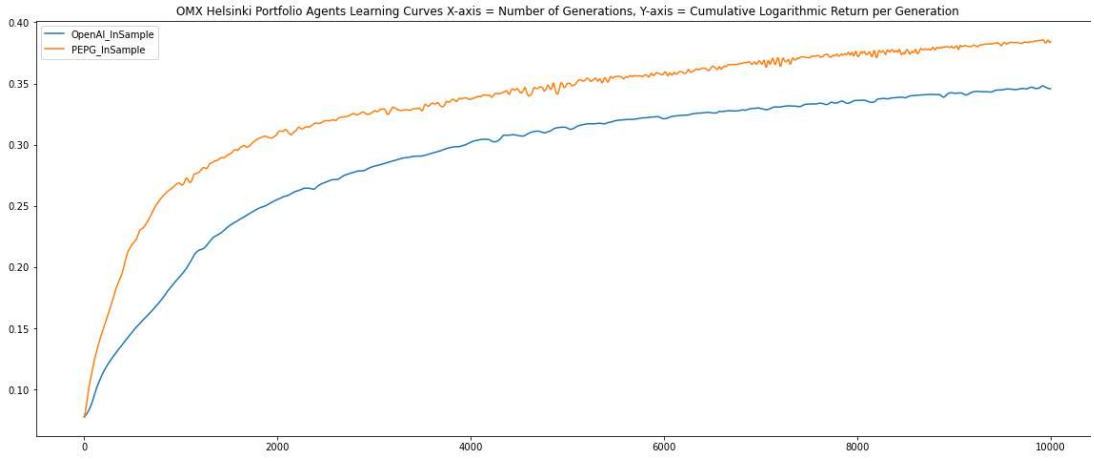

Figure 13 In-Sample learning curves of OMX Helsinki Portfolio Management Agents

PEPG shows faster learning properties in-sample. The OpenAI learning curve is lot smoother which is caused by lower learning rate. Higher learning rate in PEPG leads to faster learning but might cause some overfitting. The performance in and out-of-sample for both agents can be seen of Table 5.

Table 5 Mean Daily Logarithmic Return for Both Algorithm in OMX Helsinki Portfolio

Algorithm	**In/Out-of-Sample**	**Mean daily logarithmic return**
OpenAI-ES	In-Sample	0.346 %
	Out-of-Sample	0.107 %
PEPG-ES	In-Sample	**0.384 %**
	Out-of-Sample	**0.126 %**

5 Results

This chapter presents the key results of this study. The performance of the portfolio management agents in both PCX and OMX Helsinki asset portfolios are presented. The best performing settings in both algorithm types selected in previous chapter, are to be compared with benchmarks.

5.1 Trading Performance

5.1.1 PCX Portfolio

Now, a comparison of how well the portfolio management agents perform against benchmarks and each other is conducted. The benchmarks are SPY ETF, which follows the S&P 500 index. The other benchmark is dynamically equally weighted portfolio (DEW), which at each timestep distributes wealth uniformly to all assets in portfolio. Figure 14 shows the cumulative logarithmic return over out-of-sample period of each agent versions and benchmarks.

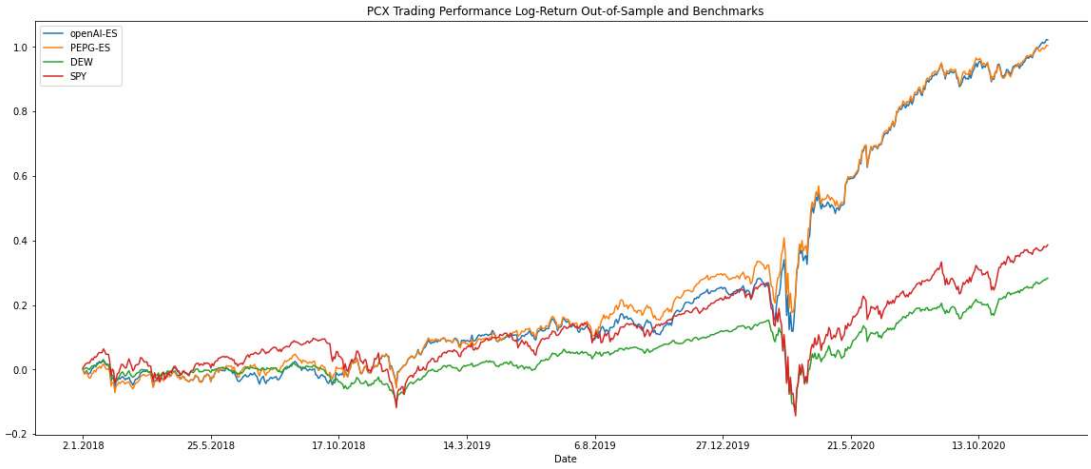

Figure 14 Evolution Strategies PCX Portfolio Management Agents against Benchmarks

During the year 2018, the SPY ETF dominates all the portfolios and benchmarks having best performance according to cumulative logarithmic return but at the end of the year both evolution strategies agents start to perform best. During 2019 both of portfolio agents keeps higher cumulative logarithmic returns than the index ETF, but don't gain any extra advantage. Still, both agents managed to achieve as good performance as the index ETF. In year 2020 the markets have a remarkable drop due to the outbreak of COVID-19. In the year

2020, both agents outperform both benchmarks remarkably and gain advantage from the volatility and instability of the market. Table 6 shows mean daily returns, standard deviation, Sharpe ratio, VaR and CVaR for both algorithms and benchmarks. The best performance in each metric is bolded. Both agents beat benchmarks in daily log-returns in 2018 and 2020. SPY benchmark has the best performance in 2019 in logarithmic daily return, but agents gain performance close to it. Equally weighted portfolio has the lowest volatility, VaR and CVaR, but agents beat SPY benchmark in those metrics. Also, both agents have higher Sharpe ratio than benchmarks in 2018 and 2020, but in 2019 DEW has the best Sharpe ratio.

Table 6 Summary Statistics of PCX Portfolios and Benchmarks for Out-of-Sample Period

Portfolio	Year	Daily Mean Log-Return	Daily Volatility	Annual Sharpe Ratio	Daily VaR 90 %	Daily CVaR 90 %
OpenAI-ES	2018	**-0.002 %**	0.890 %	**-0.04**	-1.102 %	-1.763 %
	2019	0.115 %	0.746 %	2.45	-0.857 %	-1.273 %
	2020	0.321 %	1.940 %	2.63	-1.455 %	-3.106 %
	Total	**0.144 %**	1.310 %	**1.75**	-1.108 %	-2.078 %
PEPG-ES	2018	-0.006 %	0.887 %	-0.11	-1.102 %	-1.752 %
	2019	0.107 %	0.739 %	2.29	-0.844 %	-1.253 %
	2020	**0.332 %**	1.976 %	**2.67**	-1.407 %	-3.111 %
	Total	**0.144 %**	1.327 %	1.72	-1.128 %	-2.074 %
DEW	2018	-0.029 %	**0.546 %**	-0.85	**-0.627 %**	**-1.071 %**
	2019	0.080 %	**0.406 %**	**3.14**	**-0.446 %**	**-0.682 %**
	2020	0.074 %	**1.331 %**	0.89	**-1.118 %**	**-2.492 %**
	Total	0.042 %	**0.863 %**	0.77	**-0.667 %**	**-1.472 %**
SPY	2018	-0.033 %	1.085 %	-0.49	-1.455 %	-2.326 %
	2019	**0.118 %**	0.777 %	2.41	-0.747 %	-1.372 %
	2020	0.066 %	2.127 %	0.50	-1.865 %	-4.195 %
	Total	0.050 %	1.448 %	0.55	-1.248 %	-2.749 %

Figure 15 shows the difference between logarithmic returns of all agents and all benchmarks in out-of-sample period compared with each other.

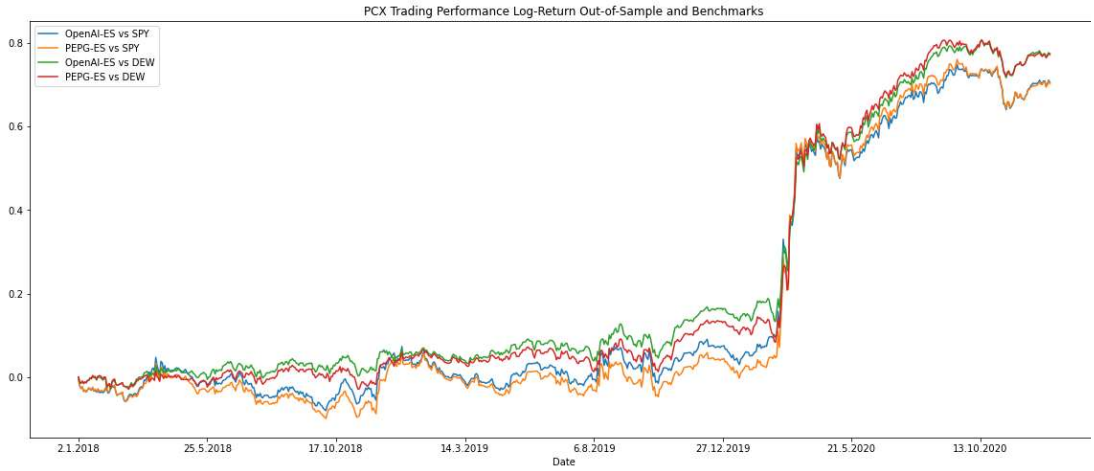

Figure 15 Differences of PCX Agents returns against Benchmarks

Figure 16 shows the portfolio weights during 1.1.2020-31.7.2020 for the best performing agent, OpenAI-ES.

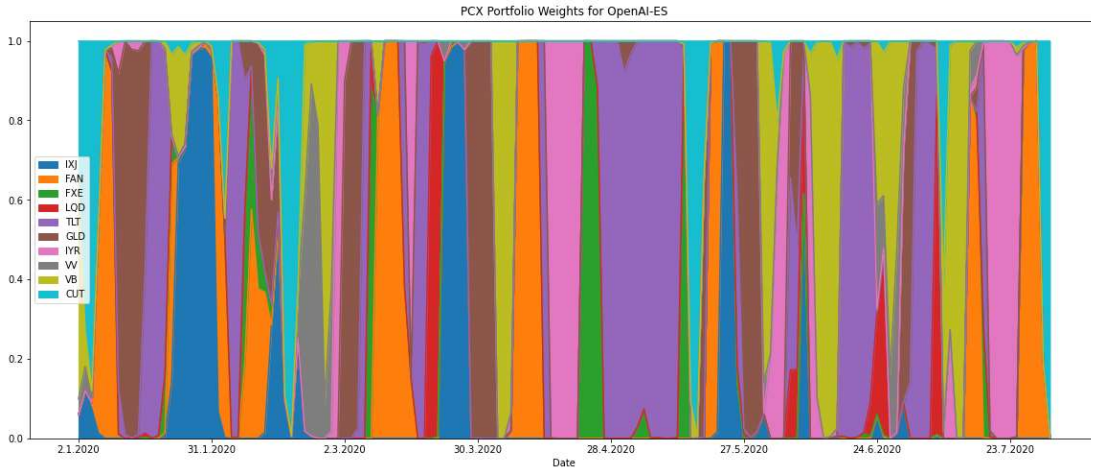

Figure 16 Portfolio Weights in year 1.1.2020-31.7.2020 for PCX Portfolio OpenAI-ES

It can be seen from the figure that there are some periods where some asset is held for a longer time and for some periods the trading frequency is much higher. For example, at the end of February the weights are dominated by VV and VB which are large and small cap ETFs, but during the beginning of the march an the COVID-19 outbreak, the agent shifts the weights quickly between more stable ETFs; FAN (Wind power), IXJ (Healthcare), GLD (Gold), TLT (Treasury Bonds), LQD (Corporate Bonds) and IYR (Real Estate). Given the performance for 2020 it can be said that the agent is actively seeking profits from the

unstable environment by frequently switching the portfolio weights. Also notice that during the out-of-sample period, the portfolio is usually clearly dominated by one asset even though there might be some small weights in other assets too. This behavior is most probably caused by the reward function which seeks to maximize the logarithmic returns of portfolio and thus seeking the best performing asset for the period. This might be an issue when considering regulation, diversification, and risk aversion. In order to avoid weight concentration, one might seek different reward functions and regularization to prevent the weights of single asset getting too large. For example, the use of differentiable Sharpe ratio in study of Moody & Saffell (1999) lead to less frequent trading actions, which enhance Sharpe Ratio in single asset setup. The use of risk-adjusted reward function might lead to more conservative weights.

To summarize both agents manage to learn profitable portfolio management strategies both in-sample and out-of-sample. Both algorithms can beat uniformly weighted portfolio during the whole out-of-sample period. Also, both algorithms were able to have as good or even better performance in out-of-sample period than SPY index-based ETF. Interesting note is that both agents perform well when there was instability in markets and were able to gain positive returns and advantage over benchmarks despite the market conditions.

5.1.2 OMX Helsinki Portfolio

OMX Helsinki portfolio management agents are compared to benchmarks: dynamically equally weighted portfolio (DEW) and OMXHPI which is the index of OMX Helsinki. Figure 17 shows the performance of each agent and benchmark in cumulative logarithmic return in out-of-sample period.

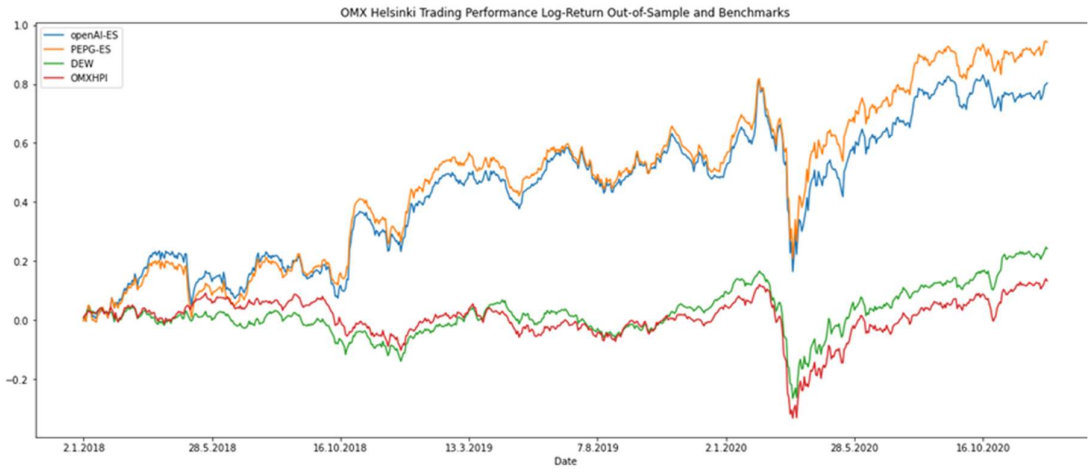

Figure 17 Evolution Strategies OMX Helsinki Portfolio Management Agents against Benchmarks

Both portfolio management agents have similar performance in out-of-sample period. Both agents can beat both benchmarks in the out-of-sample period. The year 2018 starts nicely in both agent's performances but in the middle of the year the returns start to decrease, although still able to beat the performance of benchmarks in cumulative return at that year. In year 2019 the agent gains nice returns over the year increasing the difference to benchmarks. In year 2020 during the COVID-19 outbreak, the agents and benchmark suffer from decrease caused by instability in the market. However, the portfolio management agent manages to gain advantage over the benchmarks even in that period. It is also important to note that for the agent it is not possible to avoid the crash when all the assets in portfolio experience the dip, because there is no cash or less risky asset included in the portfolio. Figure 18 shows the cumulative logarithmic return difference between both agents and both benchmarks.

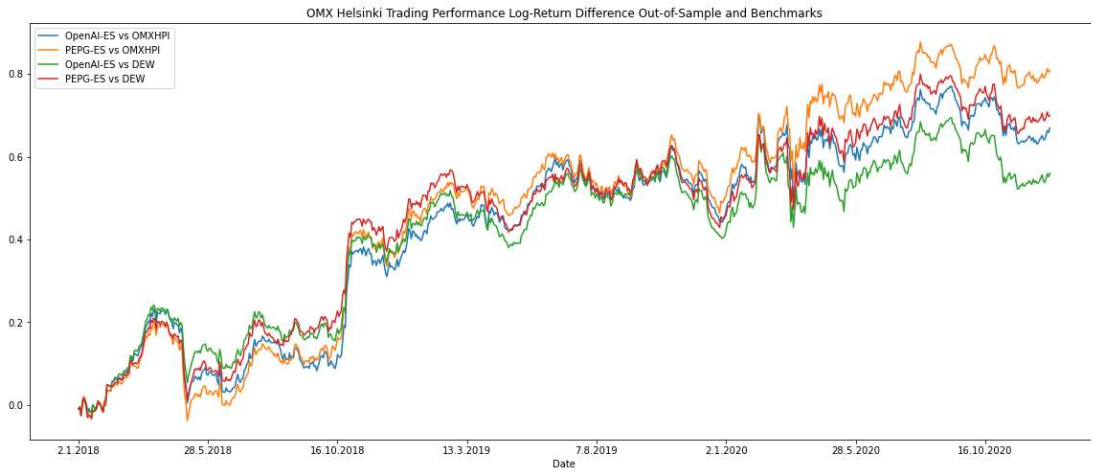

Figure 18 Differences of OMX Helsinki Agents returns against Benchmarks

The Table 7 shows the daily return, standard deviation, Sharpe ratio, VaR and CVaR for all the agents and benchmarks. Best performance in each metric is bolded.

Table 7 Summary Statistics of OMX Helsinki Portfolios and Benchmarks for Out-of-Sample Period

Portfolio	Year	Daily Mean Log-Return	Daily Volatility	Annual Sharpe Ratio	Daily VaR 90 %	Daily CVaR 90 %
OpenAI-ES	2018	0.113 %	1.692 %	1.060	-1.692 %	-2.852 %
	2019	**0.095 %**	1.328 %	**1.135**	-1.341 %	-2.164 %
	2020	0.114 %	2.720 %	0.663	-2.205 %	-4.899 %
	Total	0.107 %	1.999 %	0.851	-1.820 %	-3.364 %
PEPG-ES	2018	**0.129 %**	1.703 %	**1.205**	-1.737 %	-2.894 %
	2019	0.093 %	1.409 %	1.052	-1.480 %	-2.253 %
	2020	**0.155 %**	2.705 %	**0.907**	-2.164 %	-4.708 %
	Total	**0.126 %**	2.014 %	**0.991**	-1.828 %	-3.323 %
DEW	2018	-0.047 %	**0.845 %**	-0.887	-1.179 %	**-1.538 %**
	2019	0.083 %	**0.750 %**	1.758	**-0.762 %**	**-1.191 %**
	2020	0.058 %	**1.618 %**	0.571	**-1.576 %**	**-3.303 %**
	Total	0.031 %	**1.139 %**	0.437	**-1.070 %**	**-2.076 %**
OMXHPI	2018	-0.037 %	0.888 %	-0.658	**-1.120 %**	-1.635 %
	2019	0.053 %	0.824 %	1.016	-1.026 %	-1.419 %
	2020	0.034 %	1.736 %	0.310	-1.694 %	-3.507 %
	Total	0.017 %	1.222 %	0.216	-1.120 %	-2.232 %

Both algorithms can beat the benchmarks on logarithmic return and Sharpe ratio during the whole out-of-sample period. However, the portfolio management agents are more exposed to fluctuations and have higher risk than benchmarks, higher standard deviation, VaR and CVaR throughout the out-of-sample period.

Figure 19 shows the development of portfolio weights in period 1.1.2020-31.7.2020. Again, the trading frequencies varies, sometimes holding assets a bit longer, sometimes switching assets every day. OMX Helsinki portfolio agents suffer from the same problem as the PCX agents, meaning that they are betting one asset with big weights. This way it is possible to make profits from fluctuations in single asset, however it exposes the agent to risk and concentration over diversification.

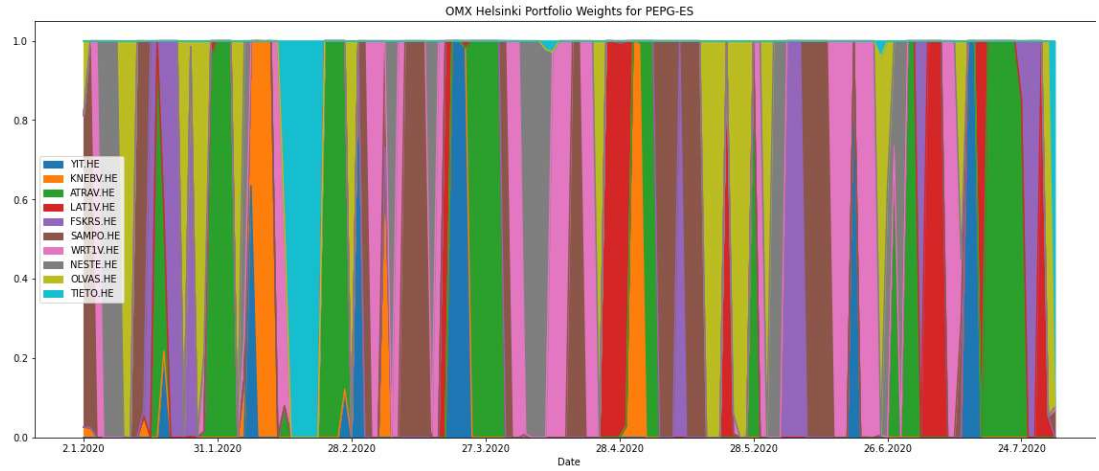

Figure 19 Portfolio Weights in 1.1.2020-31.7.2020 for OMX Helsinki Portfolio PEPG agent

Risk awareness and risk aversion is one aspect that should be studied, as not all investors are seeking high rewards in cost of high risk.

To summarize the results, both agents were able to beat the benchmarks and were able to execute profitable strategies over out-of-sample period. Both agents gain almost similar performance, PEPG being more profitable. Also, both agents performed well compared to the market during the outbreak of COVID-19. With these assets the agent experienced more instabilities in convergence of out-of-sample. This might be caused by higher stochasticity and volatility in the assets and market. Also, the Finnish stock market has much lower trading volume than the PCX market.

6 Discussion and Conclusions

This chapter concludes this thesis. First, it gives a summary and discussion of the results. Last the limitations of the study and a couple of future research ideas are presented.

6.1 Summary and Discussion

Evolution strategies-based reinforcement learning agent applied to financial portfolio management yield some interesting results. First, the algorithms were able to learn portfolio management strategies by only using price data of selected assets. The algorithms were able to generalize the learning outside of the training set and were able to dynamically allocate the portfolio over time.

During training there were a couple of things that should be considered when training machine learning and reinforcement learning algorithms which include models such as neural networks. The algorithms are quite powerful in learning phenomena from the training data, but the use of larger learning steps or too greedy learning parameters often lead to overfitting, which means that the model perfectly learns the training data but are unable to generalize the learned representations into unseen data. That is why the hyperparameter optimization is a huge part of training these kinds of algorithms. Evolution strategies-based reinforcement learning algorithms had for example learning rate, exploration noise and population size to be handled. The overfitting can be fought with parameter regularization, early stopping and by lowering learning rates. In this study lowering the learning rate most often led to worse in-sample learning but better out-of-sample performance. Also, it is highly important to note that evolution strategies are randomization-based optimization algorithm, meaning that the whole process relies on random numbers and their generation. That is why it is important to set a seed number for the learning process in order to be able to reproduce the experiments. Also, the seeding might influence learning and different seed numbers should be tested if there are some problems, such as slow learning. In this study the seed was held same for all algorithm versions.

The algorithm variants of each type and each portfolio experience positive cumulative return over out-of-sample period. In PCX portfolio case the learning agent clearly beat the portfolio asset dependent benchmark, equally weighted portfolio, by the objective function metric; mean logarithmic daily returns. However, this objective function exposed the agent's portfolio to be more volatile and riskier than the benchmarks. Compared to the S&P500 index-based ETF benchmark, SPY, the agent was able to outperform or at least gain

comparable results given the mean logarithmic return and cumulative total return during the out-of-sample period. Also, the agent's portfolio was less risky and volatile as this benchmark. However, the risk margin was not big. The agents were also able to outperform the benchmarks in Sharpe Ratio. When comparing the optimization techniques, the OpenAI-ES method outperformed the PEPG version, however the behavior was similar and margins between these algorithms were small.

In OMX Helsinki portfolio the outperformance of the agents was clearer. Both algorithms were able to beat both benchmarks every year in mean logarithmic daily return and Sharpe ratio. The equally weighted benchmark was the less volatile one during out-of-sample period. In this case OpenAI-ES was outperformed by PEPG agent. However, the behavior resulted in similar returns.

One interesting note on the results was that in both portfolios the agents outperformed and executed quite profitable policy during the outbreak of COVID-19. It is important to note that the training data did not have any similar huge drops and the training data ended at the end of the year 2017. Also, the algorithm seemed to perform well long after end of the training period which has sometimes been issue in algorithm solutions such as policy gradient, meaning that it loses the signal, or the signal loses the significance. This ability to work with longer timeframes and actions with long lasting effects was also presented in Salimans et al. (2017) as one benefit over traditional gradient based methods. Portfolio management is usually a long-term process, and this is an important feature to have in the portfolio management agent. However, the statement of Salimans et al. (2017) regarding to this specific should be tested and comparisons with traditional methods like PG should be conducted in order to really see the difference between these methods.

Implementation wise both algorithms were quite accessible to implement, OpenAI-ES being a bit simpler having the fixed exploration noise. By using population size 50-100 the computation was from 1 – 2 h to run 10 000 generations which was reasonable in the scope of this thesis.

6.2 Answers to Research Questions

This section gives the answers to research questions. Can the Evolution Strategies Based Reinforcement Learning Agent learn profitable portfolio management policy? The answer is yes in both algorithm types and both portfolio types. However, having limited back testing, full guarantee to be successful in live trading cannot be given. In both portfolio cases, both algorithms were able to have profitable portfolio management strategies over the out-of-

sample period. Also looking average daily profits in each year, the algorithms were profitable. Obviously, there were periods where the portfolio agent's portfolio value decreased but in overall trend the agent performed well.

Can the portfolio management agents beat the related benchmark measured by the performance of the objective function i.e., average logarithmic daily return? The answer is mostly yes. In PCX' case the algorithms beat the portfolio dependent equally weighted portfolio strategy. Compared to the SPY, S&P500 index ETF, the algorithms gain comparable or better performances and the agent was a little less volatile than this benchmark. In OMX Helsinki's case both algorithms beat the benchmarks in out-of-sample period.

6.3 Limitations and Future considerations

There are some limitations considering this study. First, due the scope of this thesis the amount of assets in the portfolio is limited to ten. This means that the agent does not have access to all assets in the market it operates. Despite this limitation the agent worked well, but it can be speculated that the performance would be better if all the available assets would have been used. This limitation was also done by the purpose of to keep the computational time reasonable for this thesis.

Next limitation is crucial considering the real-life applications: transaction costs. This model environment did not include transaction costs and the agents did not have to worry accommodating them. However, in real life the transaction costs are present and most definitely should be taken into account. The amount of transaction costs varies depending on the investment horizon and investment type. For example, large institutional investors can achieve low transaction costs due to larger investment volume, whereas single home-trader might experience inefficiency in frequent trading actions caused by relatively high transaction costs. Also, liquidity i.e., bid-ask spread and taxes are not taken account.

Also, the agents are not given all the possible information that is available for the assets. The agents use only three variable High, Low and Closing prices to determine the actions. For example, stock data might be enriched with news data or with financial information of the considered company. Technical indicators such as Relative Strength Index and Moving Average Convergence and Divergence, can be added to the price information. Although, it can be assumed that while the neural network tries to find meaningful signal from the data, the raw input data contains a lot of noise and hence preprocessed variables might increase stability.

The trading frequency for this study is at maximum one day, however there can be drastic changes during the trading days, so one limitation is that the agent cannot do rebalancing whenever feasible. This might not be the issue if the investors trading horizon and window are long, but being able to quickly react to market movement might enhance the performance of the agent.

When using neural networks as a function approximator one should take into account that it is a black-box model which means that it is hard or nearly impossible to understand the behavior of the model. For example, compared to linear models where the model parameters can be easily accessed and analyzed, in neural networks the parameter relations are multidimensional, non-linear and processed through layers of computation. This might cause problems when one must solve why the model works oddly at some point of time. Tuning neural network models is also important. The training accuracy can be misleading and lead to overfitting. This means that the model does not work outside the training i.e., in-sample period. To put these models into production, the model should be validated carefully with lots of validation data. The validation data in this thesis is three years long, in two markets and with limited asset combination. In real life applications the portfolio size should be a lot wider and bigger and different validation sets should be selected. One limitation of this study is that the tuning of the neural network is conducted with limited amount of hyperparameter sets. Also, different network architecture should be tested and tuned.

For future research topics, a couple of ideas are presented. First ideas consider the issues of instability of convergence and overfitting. The network parameters can be regularized with for example, l2-norm regularization. This means that the fitness function will include the cost of the network parameters in l2-norm. This causes the network weights to stay rather low and keeps the parameters from not exploding too large. Other method is to apply early stopping. This means that the training is stopped after the validation metric stops to increase or starts to decrease after some number of generations. This keeps the parameters from concentrating too much. Last, one method to tackle this issue is to implement trust-region optimization or proximal policy optimization like update clipping. This means that the parameters are updated based on the KL-Divergence as well as advantage or value functions. The update ratios are then clipped to some epsilon to keep the update close the region of the current parameters. This reduces the back and forth jumping of parameters. Evolution Strategies using trust region optimization was developed by Liu et al. (2019) and is a rather new version of evolution strategies.

The other ideas are considering the risk awareness. As not all investors are risk neutral or risk seeking for better profits, the reward function that accommodates the risk awareness should be considered. One idea would be to implement different risk-adjusted metrics as a reward function to see whether the agent would implement risk-aware portfolio management policies. For example, Sharpe Ratio could be used to guide the agents training. The other idea is to incorporate adversarial multi-agent scheme where one agent tries to maximize the returns in trading period and the adversary agent tries to reduce the returns by selecting some strategy to minimize the trading agents returns in minimax fashion.

Evolution Strategies based Reinforcement Learning agents are not limited to the problem of financial portfolio management. One interesting application would be to apply it to trade derivatives and hedging. Out of the field of Finance one interesting task would be apply these agents in product and service price optimization.

This concludes this thesis. The topic of applying reinforcement learning agents in real world problems is an interesting and a broad field. It should be recommended to keep developing and applying these methods to real life business problems as the techniques and computation power are constantly increasing. This makes this type of algorithms feasible to use as they learn doing tasks by interacting with environment, just like humans. Reinforcement learning can then be used as a part of intelligent automation. For example, agents like the portfolio management agent in this thesis can be used not just strictly on automated processes, but also to support specialists and analysts in making decisions. They could also be utilized for giving insights, just like traditional analytics.

References

Acerbi, C., & Tasche, D. (2002). On the coherence of expected shortfall. *Journal of Banking & Finance*, 1487-1503.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)* (pp. 1-6). IEEE.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation* (pp. 251-276). MIT Press.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 26-38.

Aurélien, G., & Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. *International Conference on Algorithmic Learning Theory* (pp. 174-188). Berlin, Heidelberg: Springer.

Barto, A. G., & Sutton, R. S. (2018). *Reinforcement Learning: An introduction.* MIT Press.

Barto, A. G., Thomas, P., & Sutton, R. S. (2017). Some recent applications of reinforcement learning. *Proceedings of the 18th Yale Workshop on Adaptive and Learning Systems.*

Bengio, Y., Goodfellow, I., & Courville, A. (2015). *Deep Learning.* Cambridge: MIT Press.

Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution strategies--A comprehensive introduction. *Natural computing*, 3-52.

Brooks, C., & Persand, G. (2003). Volatility forecasting for risk management. *Journal of forecasting*, 1-22.

Bubeck, S., & Cesa-Bianchi, N. (2012). Regret Analysis of Stochastic and Non-Stochastic Multi-Armed Bandit Problems.

Duffie, D., & Pan, J. (1997). An overview of value at risk. *Journal of derivatives*, 7-49.

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). Challenges of real-world reinforcement learning. *arXiv preprint arXiv:1904.12901*.

Fabozzi, F. J., & Markowitz, H. M. (2011). *The theory and practice of investment management: Asset allocation, valuation, portfolio construction, and strategies.* John Wiley & Sons.

Feinberg, E. A., & Schwartz, A. (2012). *Handbook of Markov Decision Processes: Methods and Applications.* Springer Science & Business Media.

Gold, C. (2003). FX trading via recurrent reinforcement learning. *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.* (pp. 363-370). IEEE.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, (pp. 211-219).

Hansen, N., Arnold, D. V., & Auger, A. (2015). *Evolution Strategies.* Springer.

Huang, C. Y. (2018). *Financial trading as a game: A deep reinforcement learning approach.* arXiv preprint arXiv:1807.02787.

Igel, C. (2003). Neuroevolution for reinforcement learning using evolution strategies. In IEEE, *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* (pp. 2588-2595). IEEE.

Jiang, Z., & Liang, J. (2017). Cryptocurrency portfolio management with deep reinforcement learning. *Intelligent Systems Conference (IntelliSys)* (pp. 905-913). IEEE.

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

Krokhmal, P., Palmquist, J., & Uryasev, S. (2002). Portfolio optimization with conditional value-at-risk objective and constraints. *Journal of risk*, 43-68.

Li, B., & Hoi, S. C. (2014). Online portfolio selection: A survey. *ACM Computing Surveys (CSUR)*, 1-36.

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

Liu, G., Zhao, L., Yang, F., Bian, J., Qin, T., Yu, N., & Liu, T.-Y. (2019). Trust region evolution strategies. *Proceedings of the AAAI Conference on Artificial Intelligence*, (pp. 4352-4359).

Markowitz, H. (1952). Portfolio Selection. *The Journal of Finance*, 77-91.

Mehrotra, K., Mohan, C., & Ranka, S. (1997). *Elements of Artificial Neural Networks.* MIT Press.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, (pp. 1928-1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . others. (2015). Human-level control through deep reinforcement learning. *nature*, 529-533.

Moody, J. E., & Saffell, M. (1999). Reinforcement learning for trading. *Advances in Neural Information Processing Systems*, 917-923.

Rechenberg, I. (1978). Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie* (pp. 83-114). Springer.

Rummery, G. A., & Nirajan, M. (1994). *On-line Q-learning using connectionist systems.* Cambridge: University of Cambridge.

Salimans, T., Ho, J., Chen, X., Sidor, S., & Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Sehnke, F. a., Rückstieß, T., Graves, A., Peters, J., & Schmidhuber, J. (2010). Parameter-exploring policy gradients. *Neural Networks* (pp. 551-559). Elsevier.

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of business*, 119-138.

Sortino, F. A., & Price, L. N. (1994). Performance Measurement in a Downside Risk Framework. *The Journal of Investing Fall*, 59-64.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation* (pp. 99-127). MIT Press.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 1057-1063.

Vermorel, J., & Mehryar, M. (2005). Multi-armed Bandit Algorithms and Empirical Evaluation. *European Conference on machine learning* (pp. 437-448). Berlin, Heidelberg: Springer.

Watkins, C., & Dayan, P. (1992, 8). Q-Learning. *Machine Learning*, pp. 279-292.

Wierstra, D., Schaul, T., Peters, J., & Schmidhuber, J. (2008). Natural evolution strategies. *IEEE World Congress on Computational Intelligence* (pp. 3381-3387). IEEE.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 229-256.

Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 25-40.

Appendix A: Learning Curves for PCX Portfolio

The Appendix A presents learning curves for in-sample performance of different hyperparameters for PCX portfolio.

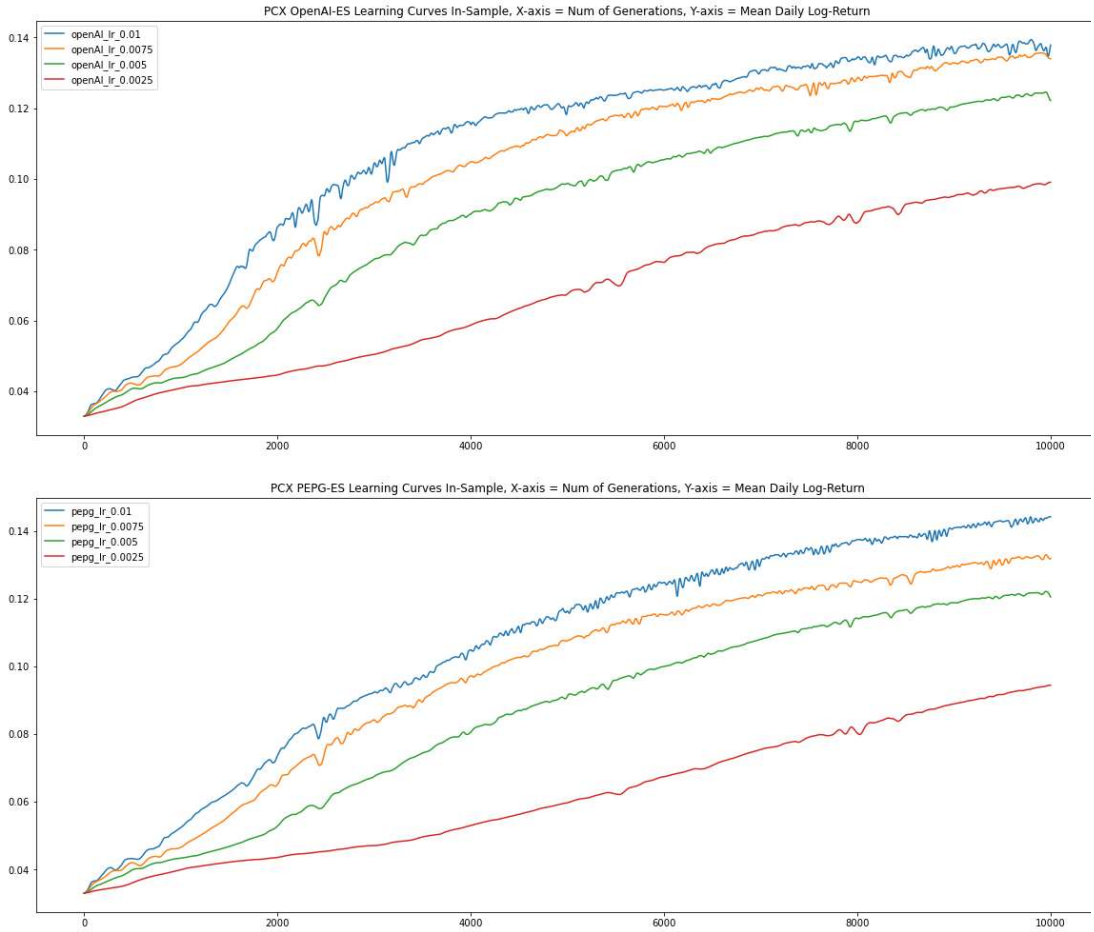

Figure A1 Effect of Learning Rate In-Sample learning PCX portfolio

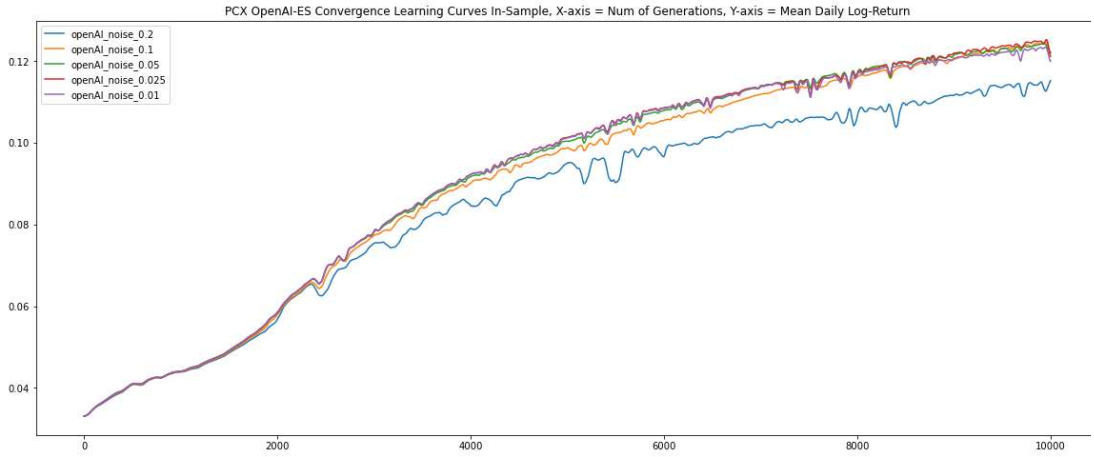

Figure A2 Effect of exploration noise parameter for OpenAI-ES PCX portfolio In -Sample

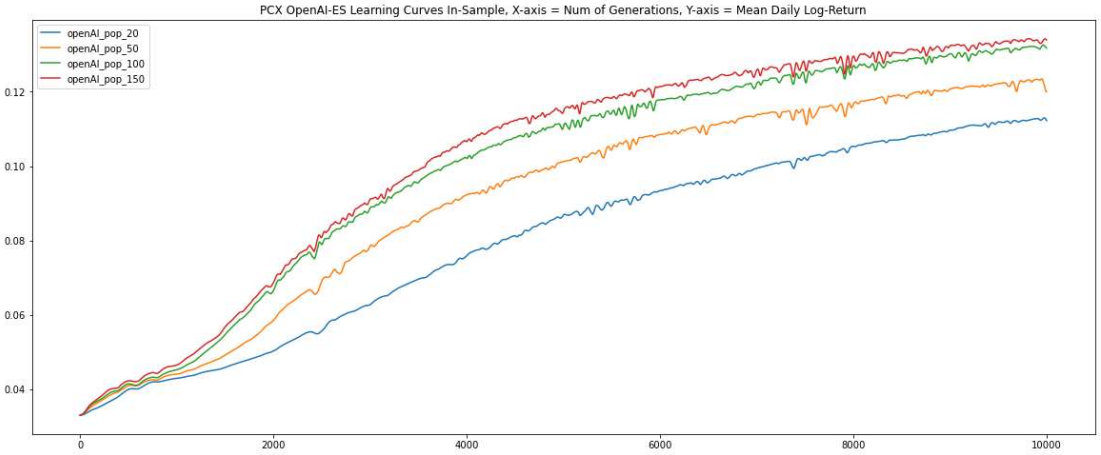

Figure A3 Effect of population size for OpenAI-ES PCX Portfolio In and out-of-sample

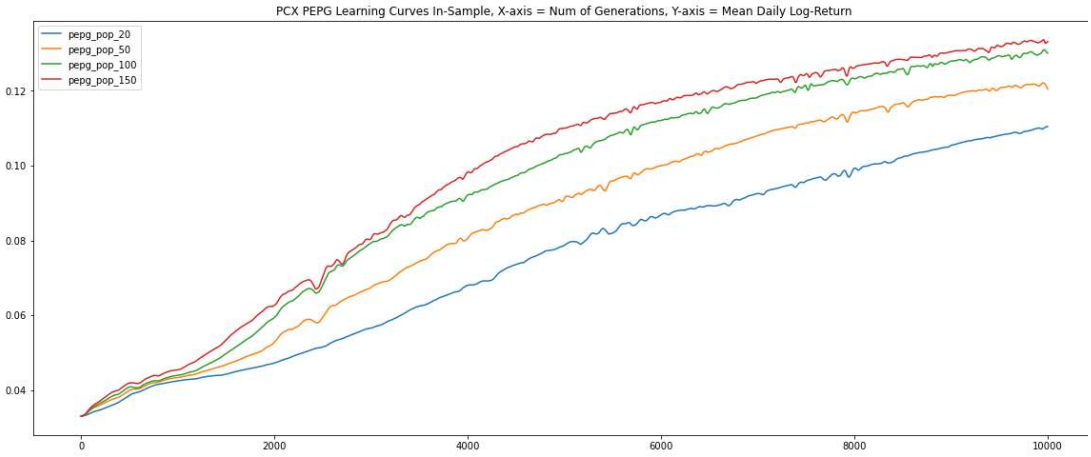

Figure A4 Effect of population size for PEPG PCX Portfolio In and out-of-sample

Appendix B: Learning Curves for OMX Helsinki Portfolio

The Appendix B presents learning curves for in-sample performance of different hyperparameters for OMX portfolio.

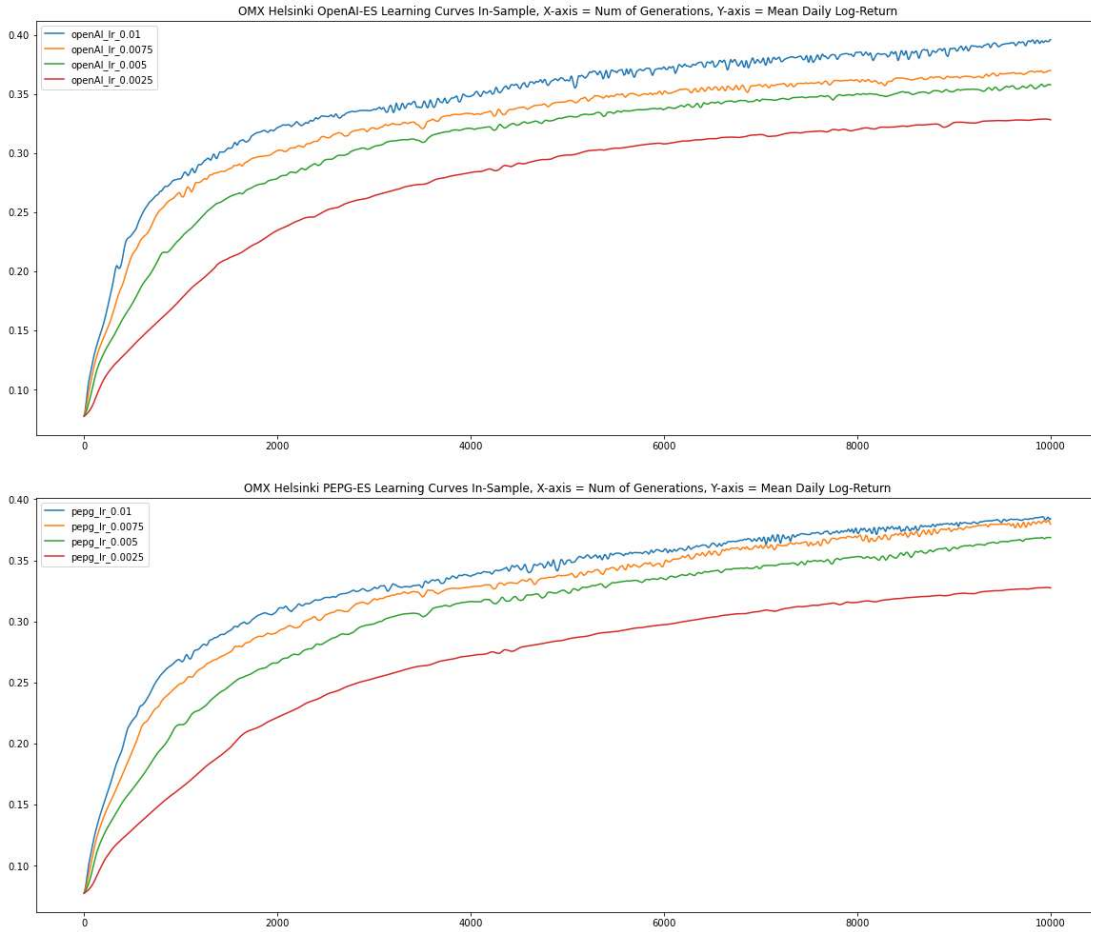

Figure B1 Effect of Learning Rate In-Sample learning OMX Helsinki portfolio

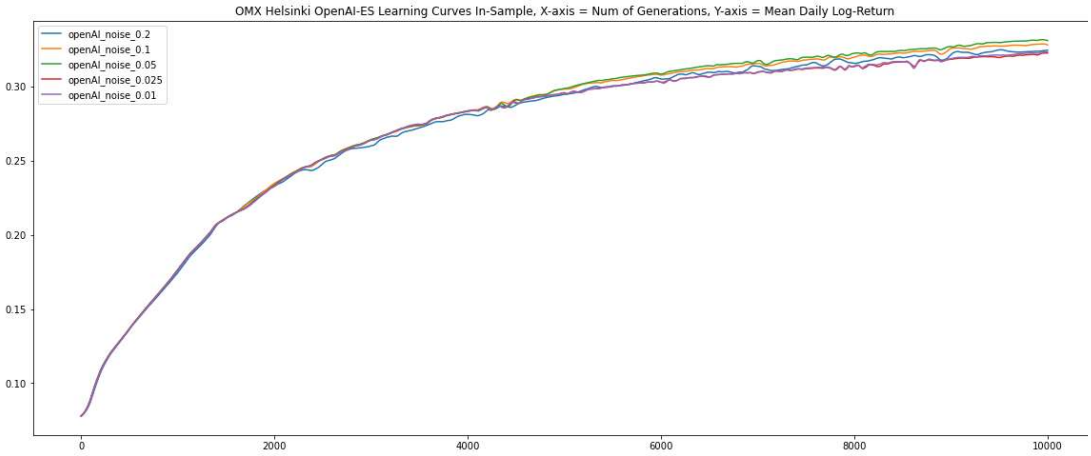

Figure B2 Effect of exploration noise parameter for OpenAI-ES OMX Helsinki portfolio In-Sample

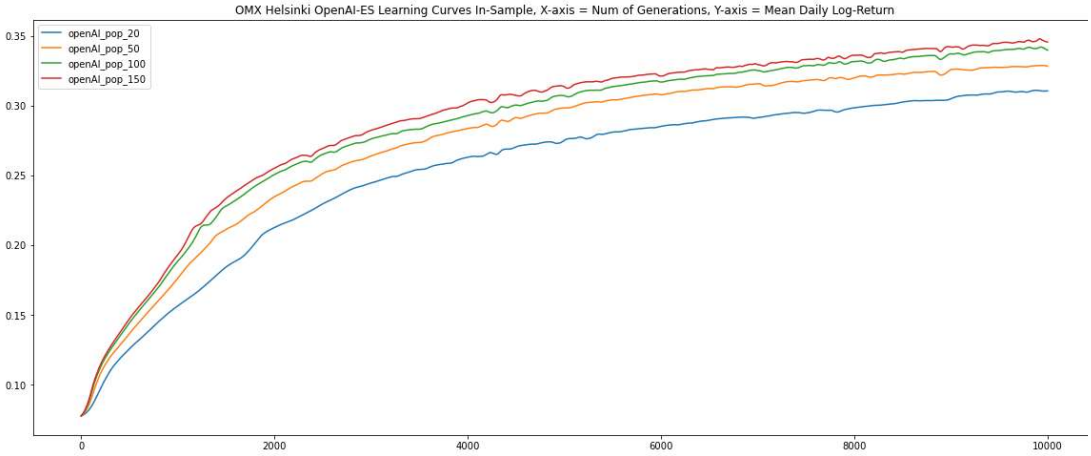

Figure B3 Effect of population size for OpenAI-ES OMX Helsinki Portfolio In-sample

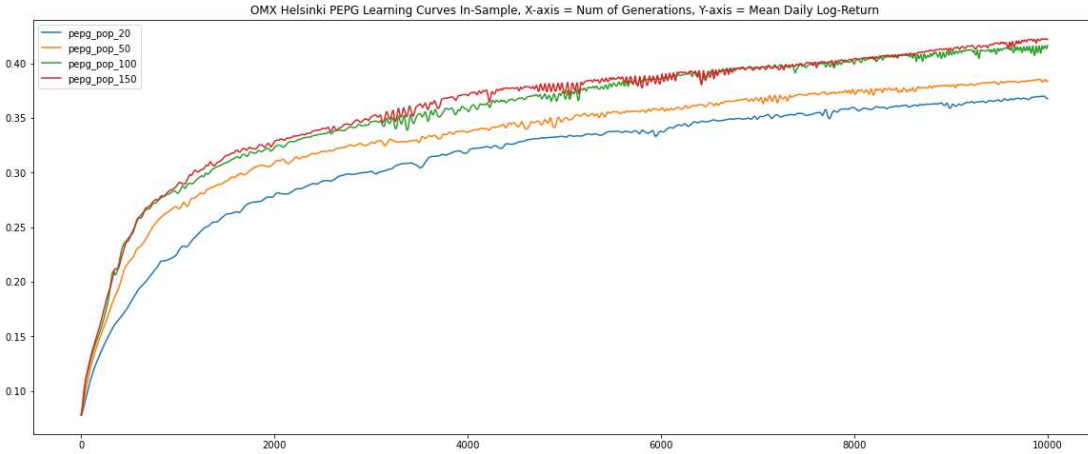

Figure B4 Effect of population size for PEPG OMX Portfolio In-sample

Appendix C: Out-of-Sample Performance for Different Hyperparameters

The Appendix C presents out-of-sample performance of different hyperparameters for both portfolios and algorithms.

Table C1 OpenAI-ES Out-of-Sample Performance for different Hyperparameters for both Portfolios

Algorithm	Hyperparameter	Values	Mean Daily Log-Return PCX	Mean Daily Log-Return OMX Helsinki
OpenAI-ES	Learning Rate	0.0100	0.1346 %	0.0965 %
		0.0075	0.1214 %	0.0883 %
		0.0050	**0.1430 %**	0.0932 %
		0.0025	0.1201 %	**0.1028 %**
	Exploration Noise	0.200	0.1338 %	0.0977 %
		0.100	0.1430 %	**0.1028 %**
		0.050	0.1360 %	0.0984 %
		0.025	0.1441 %	0.0816 %
		0.010	**0.1449 %**	0.0826 %
	Population Size	20	0.1397 %	0.0828 %
		50	**0.1449 %**	0.1028 %
		100	0.1324 %	0.1024 %
		150	0.1297 %	**0.1069 %**

Table C2 PEPG-ES Out-of-Sample Performance for different Hyperparameters for both Portfolios

Algorithm	Hyperparameter	Values	Mean Daily Log-Return PCX	Mean Daily Log-Return OMX Helsinki
PEPG-ES	Learning Rate	0.0100	0.1223 %	**0.1254 %**
		0.0075	0.1374 %	0.1071 %
		0.0050	**0.1445 %**	0.1057 %
		0.0025	0.1158 %	0.1032 %
	Population Size	20	0.1411 %	0.1154 %
		50	**0.1445 %**	**0.1254 %**
		100	0.1339 %	0.1235 %
		150	0.1374 %	0.1152 %