UKRAINIAN CATHOLIC UNIVERSITY

MASTERS THESIS

Replica Exchange For Multiple-Environment Reinforcement Learning

Author:
Dmitri GLUSCO

Supervisor:
Dr. Mykola MAKSYMENKO

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

January 8, 2020

Declaration of Authorship

I, Dmitri GLUSCO, declare that this thesis titled, "Replica Exchange For Multiple-Environment Reinforcement Learning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Abstract

Faculty of Applied Sciences

Master of Science

Replica Exchange For Multiple-Environment Reinforcement Learning

by Dmitri GLUSCO

In this project (Dmitri Glusco, 2019), we treat the Reinforcement Learning problem of Exploration vs. Exploitation. The problem can be rephrased in terms of generalization and overfitting or efficient learning.

To face the problem we decided to combine the techniques from different researches: we introduce noise as an environment's characteristics (Packer et al., 2018); create multiple Reinforcement Learning agents and environments setup to train in parallel and interact within each other (Jaderberg et al., 2017); use parallel tempering approach to initialize environments with different temperatures (noises) and perform exchanges using Metropolis-Hastings criterion (Pushkarov et al., 2019).

We implemented multi-agent architecture with parallel tempering approach based on two different Reinforcement Learning agent algorithms - Deep Q Network and Advantage Actor-Critic - and environment wrapper of the OpenAI Gym (*Gym: A toolkit for developing and comparing reinforcement learning algorithms*) environment for noise addition. We used the CartPole environment to run multiple experiments with three different types of exchanges: no exchange, random exchange, smart exchange according to Metropolis-Hastings rule. We implemented aggregation functionality to gather the results of all the experiments and visualize them with charts for analysis.

Experiments showed that a parallel tempering approach with multiple environments with different noise level can improve the performance of the agent under specific circumstances. At the same time, results raised new questions that should be addressed to fully understand the picture of the implemented approach.

Contents

List of Figures

List of Tables

List of Abbreviations

ML	**M**achine **L**earning
RL	**R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
DQN	**D**eep **Q** **N**etwork
A2C	**A**dvantage **A**ctor **C**ritic
A3C	**A**synchronous **A**dvantage **A**ctor **C**ritic
TD	**T**emporal **D**ifference
DDPG	**D**eep **D**eterministic **P**olicy **G**radients
TD	**T**emporal **D**ifference

Chapter 1

Introduction

Nowadays Machine Learning (ML) achieved a lot of great theoretical and practical results: Image Classification, Recommender Systems, various Natural Language Processing applications, time-series predictions, various advances in Deep Learning (Minar and Naher, 2018), etc. A special place in the ML takes Reinforcement Learning (RL) (Sutton and Barto, 2018). A technique allowing to reinforce the previous experience to predict the future and act based on this prediction. This approach enables agents to act in the unknown environment, explore and exploit it to achieve rewards.

1.1 Reinforcement Learning

There are two main objects in the RL: the agent and the environment. The agent can be controlled, trained to achieve different goals. In contrast, the environment can't be controlled, it can only be observed either fully or partly, where the latter one is a common real-world example. The idea of the RL is to train an agent to behave optimally in the environment to achieve the goal.

Goals may be very different: from winning the Ping Pong virtual Atari game to drive the car in the real world. In both cases, the agent will be a software program that is trained to achieve its goal. But the environments are different: in the first case, it is also a program, where all the environment behaviors are determined, in the second case it is a real-world with much higher complexity and stochasticity.

In contrast with a Supervised or Unsupervised Learning, in RL there is no ready dataset to find an optimal solution. The data is obtained through the interaction of the agent with the environment and almost immediately this data is used to optimize the behavior of the agent.

In the RL environment (Figure 1.1) agent interacts with the environment and tries to maximize the cumulative reward. The agent receives the state and the reward from the environment and interacts with it using actions.

R_t - **reward** at the timestamp t

A_t - **action**

O_t - **observation**

H_t - **history**. $H_t = A_1, O_1, R_1, \ldots, A_t, O_t, R_t$

S_t - **state** - information used to determine what happens next. $S_t = f(H_t)$

The environment is described as a Markov Decision Process (MDP), that is defined by the $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where:

1. $\mathcal{S}$ is a (finite) set of states
2. $\mathcal{A}$ is a (finite) set of actions
3. $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$

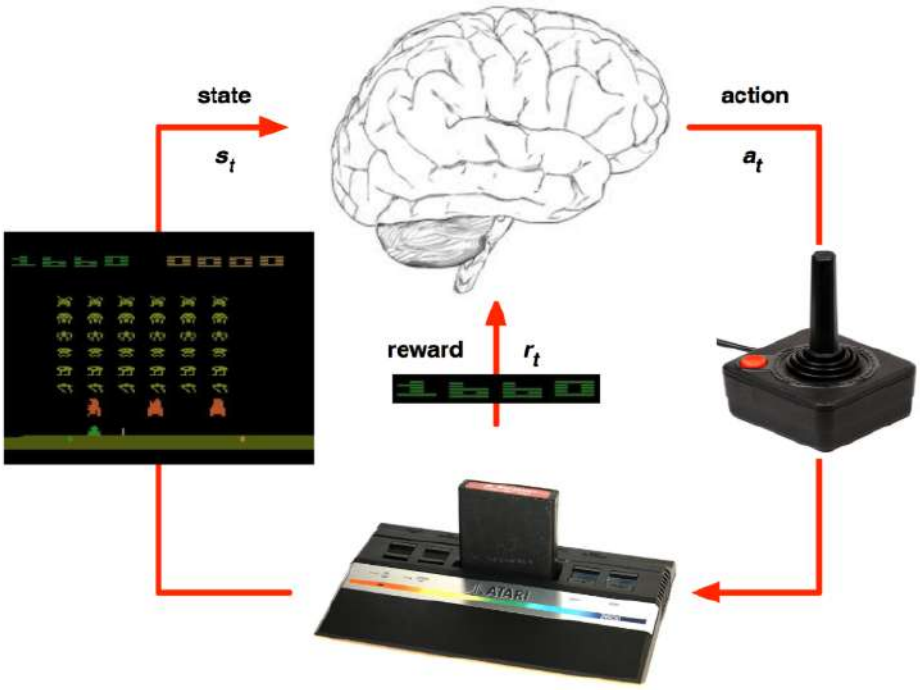

FIGURE 1.1: Agent interaction (*Reinforcement learning lectures by DeepMind*).

4. $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$

5. γ is a discount factor, $\gamma \in [0, 1]$

G_t - **return** - is the total discounted reward from time-step t. $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$v_\pi(s)$ - **state-value function** of MDP - $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$

π - **policy** - a strategy of how to behave in the environment or in other words what actions to take in a given states.

The state-value function represents how good is a state for an agent to be in. It is equal to the expected total reward for an agent starting from state s.

$q_\pi(s, a)$ - **action-value function** of MDP - $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

The action-value function is an indication of how good it is for an agent to pick action a while being in state s.

$v_*(s)$ - **optimal state-value function** - $v_*(s) = \max_\pi v_\pi(s)$

$q_*(s, a)$ - **optimal action-value function** - $q_*(s, a) = \max_\pi q_\pi(s, a)$

Bellman Optimality Equation for MDPs:

$v_*(s) = \max_a q_*(s, a)$

$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$

Combining 2 parts:

$v_*(s) = \max_a (\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s'))$

$q_*(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s', a')$

As mentioned earlier, the environments may be different. To train a good agent in a very simple environment like mentioned Pong or much more complex environment with higher dimensionality different algorithms were developed over time (Figure 1.2).

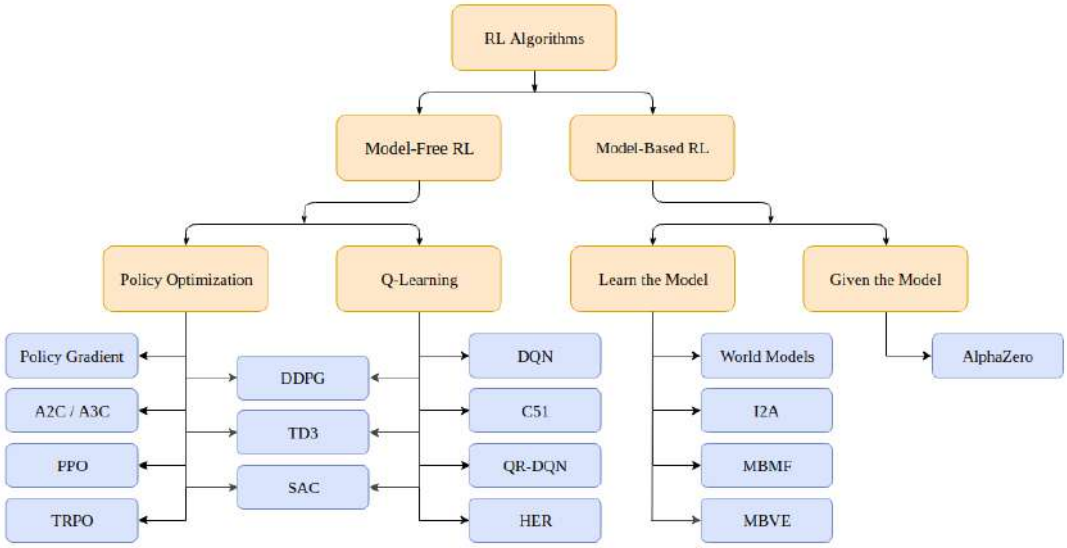

FIGURE 1.2: A Taxonomy of RL Algorithms (*OpenAI Spinning Up: An educational resource produced by OpenAI that makes it easier to learn about deep reinforcement learning*).

There are 2 main groups of the RL algorithms: Model-Free RL and Model-Based RL. In Model-Based, the model of the world is built explicitly. The model represents the transitions and outcomes of the environment. Model-Free RL doesn't build the model of the environment explicitly. In Model-Free agent is trained directly by state/action-values or policies.

In this work, we are focused on the Model-Free approach. Model-Free RL also divides into 2 groups: policy-based methods and value-based methods. Policy-based methods try to find the best policy that may be deterministic or stochastic. Value-based methods are focused on predicting action/state-value.

Chapter 2

Motivation

RL is already used in various applications. However, the common downside is a lack of stability and control both during the training and inference of the agents. In the work (Mao et al., 2016) "Resource Management with Deep Reinforcement Learning" authors designed the algorithm to allocate limited resources to different tasks. The objective was to minimize the average job slowdown. The authors formulated current resource allocation as the state space and used the policy gradients approach to solve the task.

There are a lot of works on applying RL in Robotics. In "Reinforcement learning in robotics: A survey" paper (Kober, Bagnell, and Peters, 2013) the authors provided a survey of work in reinforcement learning for behavior generation in robots. They highlighted both key challenges in robot reinforcement learning as well as notable successes.

Other researches (Arel et al., 2010) used a multi-agent system for network traffic signal control. Their main goal was to minimize the average delay, congestion, and likelihood of intersection cross-blocking. Traffic flow was used as a state. The authors applied Deep Q Network (DQN) algorithm for their task.

Among other fields where RL was used are chemistry (Zhou, Li, and Zare, 2017), personalized recommendations (Zheng et al., 2018), bidding and advertising (Jin et al., 2018), and many other fields, where RL is used.

Probably, the biggest field of experiments, tests, and researches for RL is games. The reason is that the game is a ready-made simulated study environment. A lot of games were developed so far. They have all kinds of different complexities: from the easiest, where state space can be represented with only one dimension with few values, to the hardest, where it is impossible to represent all the possible states and not all states are observable. The goal may sound easy - win the game, but it is a very complex goal with a lot of different tactics, strategies, moves, options and etc. Researches use these game environments to test new approaches, find better algorithms, try to build a generic approach, that can beat totally different games. One of the recent researches (Vinyals et al., 2019) showed that RL can compete with the best players around the world in the game of StarCraft 2 (with some limitations), which is a very complex game that has uncountable possible states and actions.

Creating an agent that can potentially train and achieve human-level results on all the possible games is a challenging task that is far from being achieved yet. There are multiple challenges on the way. Exploration vs. Exploitation dilemma is just a stability and training issue, that prevents agents from stable training and the performance in the unknown environments.

Chapter 3

Problem Setting

To understand better the Exploration/Exploitation dilemma let us understand what does it mean in terms of RL.

Exploration means finding all the possible combinations of states and actions, i.e. explore the environment. If we talk about exploration then there is no other reward function to optimize than finding new state-action combinations, i.e. reward from the environment is not used. Some research is done in the direction of intrinsic motivation, where the external reward from the environment is not used and agents explore the environment ((Barto, 2013), (Aubret, Matignon, and Hassas, 2019)).

In contrast, exploitation means having some goal, some reward function that should be optimized. The agent will only exploit previous knowledge (what was the value of the reward after some state-action pair) to act in the way of maximizing this reward. I.e. the agent will never act differently in the same situation, even if different actions will lead to a better reward because the agent does not know it.

To find the best actions (in terms of getting the highest return) for all the states agents should both explore new knowledge and exploit previous knowledge. To find such optimal policy in any environment the agent should randomly explore it infinite time and visit all the possible combinations of states and actions, which is technically impossible over a finite amount of time (Sutton and Barto, 2018).

There are some simple practical techniques like ϵ-greedy exploration. It means that the agent will choose random action with odd ϵ or the best action according to the obtained knowledge with odd $(1 - \epsilon)$. Usually ϵ is decreased over time to privilege exploitation after enough exploration. Such a technique decreases the time that is required to converge at some good local optima. But for environments with much higher complexity these local optima may be not good enough and the amount of time, needed for getting reasonable results, is still too high.

Exploration/Exploitation dilemma can be viewed from different angles. Basically, this dilemma may be rephrased as the problem of overfitting or the problem of efficient learning.

The overfitting problem is well-known to the ML world. The problem of overfitting can be explained in the way that the model perfectly predicts the data it was trained on and because of that loses the ability to predict properly on the data the model has not seen (Figure 3.1). Another way to observe the overfitting is when the model is sensitive to small perturbations.

The main reason for the overfitting is that the model is too complex for the dataset it is trained on. And there are a lot of different solutions to overcome this problem:

- Simplify the model
- Greatly increase the dataset
- Data augmentation

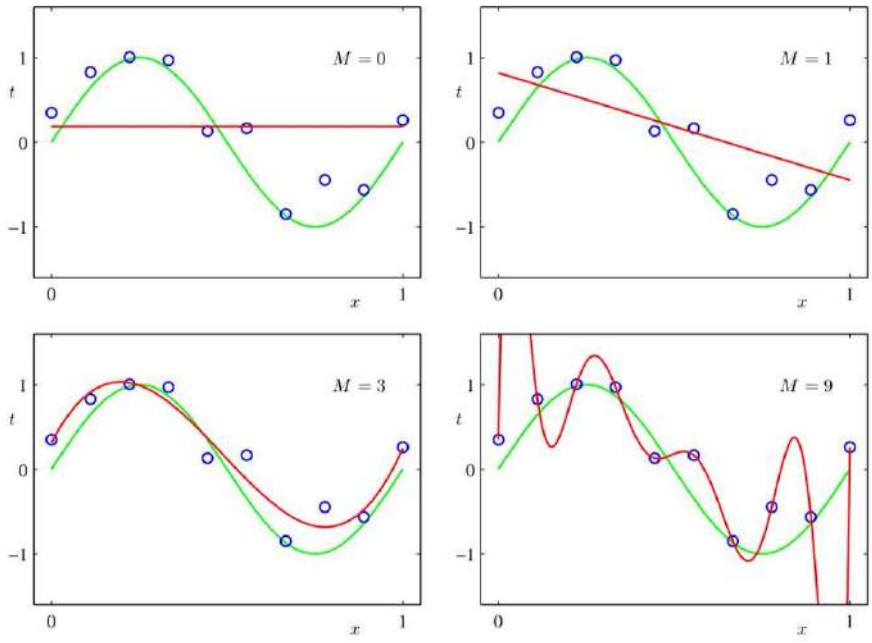

FIGURE 3.1: Underfitting and overfitting (Bishop, 2006).

- Cross-validation
- Add regularization
- Dropout
- Other techniques that improve predictions on the test or unobserved data

In the terms of the RL agent, if the agent overfits that means that he cannot perform on the states that he did not see during training. Usually, agents are trained on the same environments on which they are applied further, it means that they train on the test set. And if the environment will change or if we want to use the agent's knowledge in a similar environment (with some differences) then usually the agent will fail. A good example is the maze environment: the agent can learn how to go through the maze to the finish, but when the maze will change then the agent will fail.

Chapter 4

Related Work

Because the balance between exploring and exploiting the environment is one of the fundamental problems of the RL, a lot of different researches and algorithms investigate this problem in different senses. Nevertheless, no approach was found yet to overcome this problem optimally for all the environments. We reviewed related researches to our idea.

Authors of the "Quantifying Generalization in Reinforcement Learning" paper (Cobbe et al., 2018) investigated the problem of overfitting in deep RL. They introduced a new environment called CoinRun with procedurally generated environments to construct distinct training and test sets. The environment is designed as a benchmark for generalization. It is shown that agents overfit to large training sets. Deeper convolution architects and traditional methods, like L2 regularization, dropout, data augmentation improve generalization.

The idea of the noise as an improvement to the algorithm was used all over the different ML fields. The most commonly used is the dropout technique (Srivastava et al., 2014). Paper (Packer et al., 2018) by Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, Dawn Song from Berkeley presented a good foundation to measure and analyze generalization. Authors implemented a deterministic, random and extremely random environment set up to train the agent on different environment setup and compare the performance using different algorithms. They empirically proved that training vanilla deep RL algorithms with environmental stochasticity may be more effective for generalization than specialized algorithms. The authors used the noise in the way of changing the state observed by an agent. And proved improvement of generalization abilities (or in other words - exploration).

Another approach of improving exploration was investigated by the "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation" paper (Kulkarni et al., 2016) authors. They considered the environment with very sparse and delayed feedback to develop hierarchical-DQN (h-DQN), a framework to integrate hierarchical action-value functions. The authors applied the intrinsic motivation approach to a top-level q-value function. Lower-level function learns a policy over atomic actions to satisfy the given goals. They showed an improvement in the extrinsic reward of their approach comparing to the DQN baseline.

In the "Asynchronous Methods for Deep Reinforcement Learning" paper (Mnih et al., 2016) authors applied the asynchronous approach to the well-known RL algorithms. They introduced the Asynchronous Advantage Actor-Critic (A3C) algorithm that is using parallel actor-learners to update a shared model. The new approach surpassed state-of-the-art in half the training time and had a stabilizing effect on the learning process.

In other research made by Deepmind team (Jaderberg et al., 2017) authors investigated hyperparameter optimization. They noted that the task of RL can be highly non-stationary and it leads to the problem that ideal hyperparameters are also highly non-stationary. The authors introduced Population Based Training (PBT) - a technique to train multiple agents at the same time to find the optimal set-up. The advantage of this technique is that it is built on top of the existing solution, whether it is a simple DQN or a big ML pipeline (Figure 4.1).

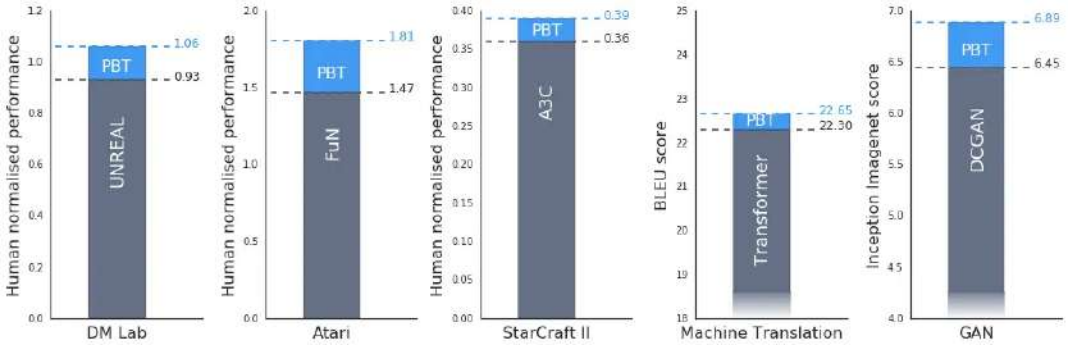

FIGURE 4.1: The results of using PBT over random search on different domains (Jaderberg et al., 2017).

PBT technique is a kind of genetic algorithm. Multiple agents are trained in parallel, at some time an agent can copy the model parameters from the best worker or explore new parameters by changing the values randomly. This greedy copy or random initialization is also an exploration vs. exploitation dilemma but in the terms of hyperparameters.

In the research (Pushkarov et al., 2019) made by Vlad Pushkarov, Jonathan Efroni, Mykola Maksymenko, Maciej Koch-Janusz, authors tried a different approach to find the best hyperparameters. The main idea of the research is that the noise (that is incorporated in the gradient) flattens the loss function. This trick helps to avoid stucking in a lot of local-minima and travel through the hyperparameters space faster. Similarly to the Deepmind research, authors trained multiple models in parallel with the different initial states in the hyperparameter space. After some number of iterations the models are changing the hyperparameters by the Metropolis-Hastings scheme:

$$\mathcal{P}(\boldsymbol{W}, \beta_m; \boldsymbol{W'}, \beta_n) = \begin{cases} 1, & \Delta \leq 0, \\ exp(-\Delta), & \Delta > 0 \end{cases}$$

Transition probability satisfies detailed balance condition in the system. This guarantees in the long-time that there is no dependence on the initial state. Or in other words, the algorithm will explore whole hyperparameter space to find the best hyperparameters.

Chapter 5

Approach to Solution

To find the best way of exploring the environment and at the same time exploiting the good paths we decided to combine approaches from the Chapter 4. We introduce noise as an environment's characteristic similar to the "Assessing Generalization in Deep Reinforcement Learning" research (Packer et al., 2018); add multiple agents and environments setup to train in parallel and interact within each other similar to the "Population Based Training of Neural Networks" research (Jaderberg et al., 2017); use the Metropolis-Hastings scheme to describe the exchanges similar to the "Training Deep Neural Networks by optimizing over nonlocal paths in hyperparameter space" research (Pushkarov et al., 2019).

5.1 High-level overview

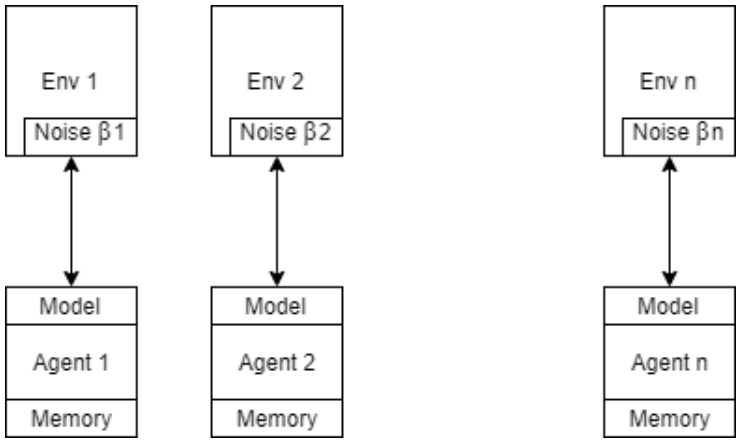

FIGURE 5.1: Multiple agents and environments setup.

We use multiple agents and environments setup (Figure 5.1). Environments are totally similar except one parameter - β that is the noise. Environments are set up in the way that $(i+1)^{th}$ environment has a bigger noise than i^{th} environment: $\beta_{i+1} > \beta i$. After some number of iterations agents will swap their environments by the Metropolis-Hastings scheme.

Metropolis-Hastings exchange rule and detailed balance (Pushkarov et al., 2019) guarantee that in the long-time limit the environment will be explored without any dependence on the initial state of the weights, which influences the actions taken and therefore seen parts of the environment. Comparing to the approach in (Jaderberg

et al., 2017), where agents perform greedily which leads to the dependence on the initial state (Figure 5.2).

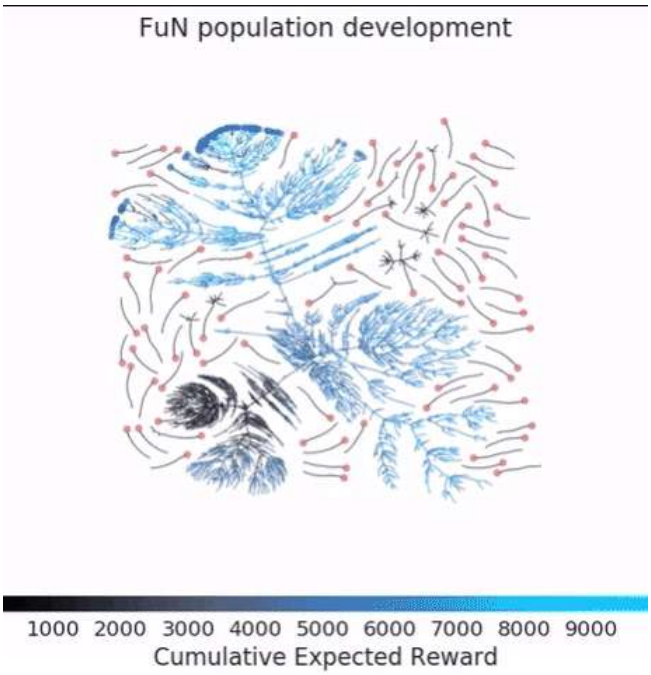

FIGURE 5.2: The evolution of the population during the training of Feudal Networks (FuN) on MS Pacman. Pink dots represent initial agents, blue ones the final ones (Jaderberg et al., 2017).

5.2 Environment

Environment is an OpenAI Gym environment (*Gym: A toolkit for developing and comparing reinforcement learning algorithms*). To test our approach we decided to start with a simple environment - CartPole. The noise is added as a parameter of the environment. Noise influences the state that the agent observes.

Before the agent observes the current state (S_t) and the next state (S_{t+1}), the states are being multiplied by the number sampled from the Gaussian probability:

$S'_t = S_t * X$ and $S'_{t+1} = S_{t+1} * X$, where X is sampled from Gaussian probability density function $P(X) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(X-\mu)^2}{2\sigma^2}}$ with $\mu = 1$ and $\sigma = \beta_i$, where β_i is the parameter of the environment.

5.3 Agent

An agent is any RL algorithm. It can be DQN, A2C, DDPG or any other. It will interact with its environment and train its Neural Network (or anything else that is needed for the algorithm if it is not based on the Deep Learning, e.g. Q table). For simplicity, we decided to start with the DQN and A2C algorithms.

5.3.1 DQN

DQN is a model-free and value-based learning algorithm. DQN uses a neural network to predict the q-value.

$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma max_a Q(s_{t+1}, a))$, where α - is a learning rate, γ - is a discount rate.

To update the weights of the network experience replay approach is used. Experience replay is a biologically inspired process that uniformly (to reduce the correlation between subsequent actions) samples experiences from the memory and for each entry updates its q-value by fitting calculated q-values $Q(s_t, a_t) = (r_t + \gamma max_a Q(s_{t+1}, a))$ for the specified action to the network to update the weights of the network.

There are few hyperparameters of this algorithm:

- Training number of episodes - how many games an agent will play to train.
- Learning rate - determines the trade-off between the old q-value importance and the new q-value Takes 0..1 values.
- Discount rate - determines the discount factor to balance between immediate and future reward.
- Exploration rate - determines if the agent should take a random action or behave greedily by taking action with the max q-value.
- Max exploration rate, min exploration rate, exploration decay rate - values to adjust exploration rate during the training.
- Memory size - the size of the queue that stores (state, action, reward, next state, done) combinations obtained during the training.
- Batch size - size of how many (state, action, reward, next state, done) combinations should be taken into account during experience replay to update the DQN.

5.3.2 A2C

A2C (Advantage Actor-Critic) is a combination of policy-based and value-based learning approaches. A2C uses a neural network with 2 heads: the actor and the critic. The actor takes as input the state and outputs the best action. It controls how the agent behaves by learning the optimal policy. The critic evaluates the action chosen by the actor by computing the value function (Figure 5.3). The actor is being improved in choosing the right action by the critic using the Temporal Difference (TD) error: $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ at every step. The critic is also improved by the TD error.

The advantage function shows how better an action is compared to the others at a given state. $Q(s_t, a) = V(s_t) + A(s_t, a) \implies A(s_t, a) = Q(s_t, a) - V(s_t) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$. In A2C instead of learning Q values (or V values) agent is learning advantage values.

Entropy is used in the loss function to improve exploration of the agent:
$H(\pi(A_t|s_t, \theta_\pi) = -\sum_t \pi(A_t|s_t, \theta_\pi) log \pi(A_t|s_t, \theta_\pi)$.
Resulting Loss function:
$minL = -log(\pi(A_t|s_t, \theta_\pi))\delta_t - \beta H(\pi(A_t|s_t, \theta_\pi)) + \zeta(\hat{v}(s_t) - G_t)^2$, where
$log(\pi(A_t|s_t, \theta_\pi))\delta_t$ - policy loss, that maximizes expected log rewards

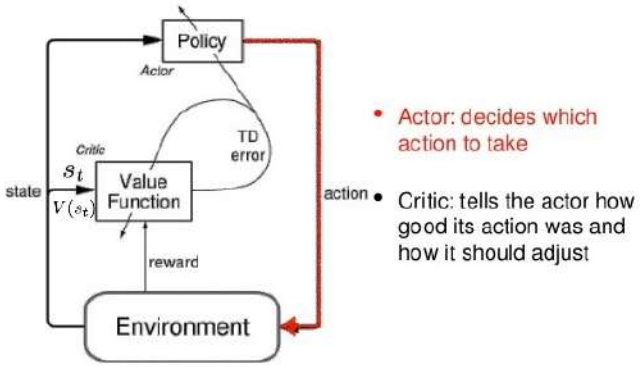

FIGURE 5.3: Actor-Critic (Sutton, Barto, and Others, 1998).

$\beta H(\pi(A_t|s_t, \theta_\pi))$ - entropy loss, that maximizes entropy to explore
$\zeta(\hat{v}(s_t) - G_t)^2$ - value loss, that minimizes predicted value error
Hyperparameters of the A2C algorithm:

- Training number of episodes - how many games an agent will play to train.
- Learning rate - determines the trade-off between the old q-value importance and the new q-value Takes 0..1. values.
- Discount rate - determines the discount factor to balance between immediate and future reward.
- Batch size - size of how many (state, action, reward, next state, done) combinations should be taken into account.
- Entropy loss factor - multiplier to tune the influence of the entropy in the loss function.
- Value loss factor - multiplier to tune the influence of the value in the loss function.

5.4 Replica Exchange

The idea of the replica-exchange is to switch the agent to the environment with higher noise if it "stuck" in some sense in local-minima, comparing to the agent from the environment with higher noise. It means that every k iterations we will swap the environments for the adjacent agents with the probability by the Metropolis-Hastings scheme. Similarly to the "Training Deep Neural Networks by optimizing over nonlocal paths in hyperparameter space" work (Pushkarov et al., 2019) we will arrange the environment noises in an ascending manner. Global detailed balance condition guarantees that in the long-time limit the joint probability distribution will be sampled faithfully, which means that there is no dependence on the initial state. Exchange occurs with analytically computable acceptance probability:

$$\mathcal{P}(\boldsymbol{W_p}, \beta_p; \boldsymbol{W_n}, \beta_n) = \begin{cases} 1, & x > 0, \\ exp(x), & x \leq 0 \end{cases}$$

Where

$$x = \Delta(\beta_n - \beta_p)(L(\boldsymbol{W_n}) - L(\boldsymbol{W_p}))$$

And $L(\boldsymbol{W_n})$ is the average score of the n^{th} agent for the last m episodes.

5.5 Hypotheses

Combining the mentioned approaches we have next hypotheses:

- Higher noise improves the exploration of the agent. This should be possibly reflected in the speed of how the model weights change over time.
- A single agent in a low-noise environment can exploit it and overfit - stuck in some corner of the parameter space. Metropolis-Hastings exchange between agents in different environments fixes the exploration (ergodicity) of the model and leads to better and stable training.

Chapter 6

Solution

To test our solution we implemented two RL learning algorithms: DQN and A2C. We used the CartPole environment from OpenAI Gym as a place for our tests (Dmitri Glusco, 2019). Implemented architecture can be viewed on the Figure 6.1. Code with algorithm consists of the next parts:

- Noise Learning - main code of the proposed algorithm to train agents with different level of noise and agent exchanges.
- Agents - implemented agents: DQN and A2C.
- Environments - wrapper on the OpenAI gym environment to add noise.
- Metrics Manager - manager to work with agent metrics.
- Results Manager - manager to store the results of one execution of the proposed algorithm.
- Visualizer - shows the aggregated results for multiple executions to analyze.

6.1 Noise Learning

This part contains the main code of the algorithm (Algorithm 1).
Algorithm of SmartExchangeAgents function is described in algorithm 2.
Our approach has next hyperparameters:

- Agents number - number of agents that will be simultaneously trained using noise learning.
- Noise environment step - amount to increase the noise for each next new environment. E.g. if the value for this hyperparameter is 0.1 then the noise of the first environment is 0, the noise of the second environment is 0.1, the noise of the third environment is 0.2 and so on.
- Training episodes - general hyperparameter required for both (A2C and DQN) RL agents to train.
- Play episodes - general hyperparameter required for both (A2C and DQN) RL agents. Determines the number of episodes to play on the zero-noise environment after the training.
- Warm-up steps - the number of steps to proceed before the first exchange attempt. It should be tuned with respect to the chosen environment.
- Exchange steps - the number of steps to proceed for every exchange attempt.

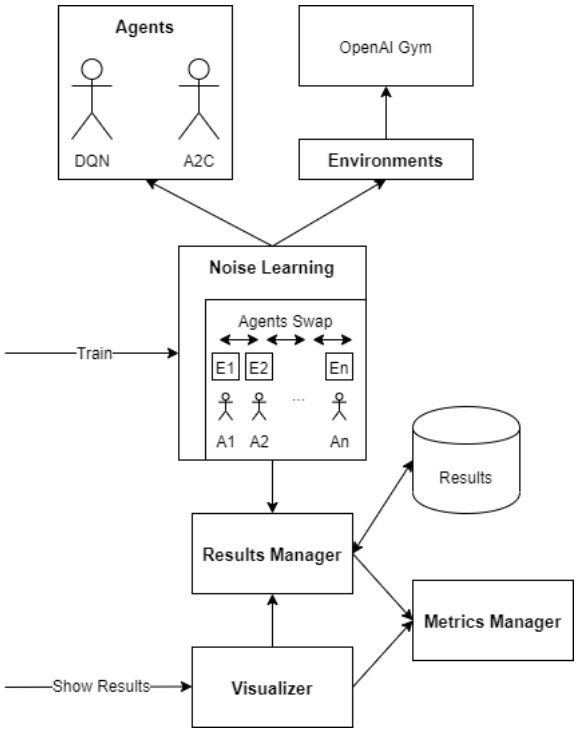

FIGURE 6.1: Implemented architecture diagram.

- Random exchange probability - probability to perform exchange with Random Exchange Type (for each agent).
- Exchange delta - coefficient to tune exchange probability. Lower value increases exchange probability.
- Exchange items reward count - number of agent's last rewards to get average for exchange probability.

6.2 Agents

This part contains implemented RL agents.

6.2.1 DQN

One of the implemented algorithms is DQN. For a CartPole environment, DQN is a pretty simple fully-connected network: 4 input neurons, one hidden layer with 256 neurons and relu activation, 2 output neurons with linear activation and MSE loss. 4 input neurons because CartPole has 4 parameters in observation space and 2 output neurons to predict 2 q-values for 2 actions from action space.

6.2.2 A2C

Second implemented algorithm is A2C. For a CartPole environment, A2C is also a pretty simple fully-connected network: 4 input neurons, first hidden layer with 64 neurons and relu activation, second hidden layer with 128 neurons and relu activation,

2 output heads: actor has 2 output neurons with softmax activation to choose the action, critic has 1 output neuron with linear activation to evaluate the state.

6.3 Environments

Environments contains wrapper on the OpenAI gym environment. Wrapper adds noise in the manner described in the Chapter 5. For our tests we focused on the CartPole environment.

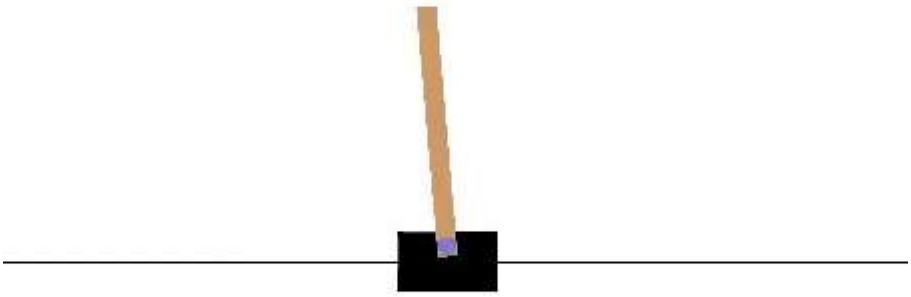

FIGURE 6.2: CartPole (*Gym: A toolkit for developing and comparing reinforcement learning algorithms*).

A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track (Figure 6.2). The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center, or the episode length is greater than 500 steps (*Gym: A toolkit for developing and comparing reinforcement learning algorithms*).

The observation space of the environment has 4 dimensions with continuous values (Table 6.1) and the action space has 2 dimensions (Table 6.2).

6.4 Metrics Manager

The metrics manager contains the manager to work with agent metrics. The manager can aggregate metrics for multiple executions and track the noise exchange for each iteration for each agent. For our experiments, we focused on three metrics: loss, score (reward), distance (diffusion). They are stored as an average value for each iteration.

The loss is simply a value of the loss function for every neural network weights update procedure. The score or reward is the reward of the environment.

The distance or diffusion is the euclidean distance between the initial weights matrix (flattened to the vector) and the weights matrix of the current iteration. We also call it diffusion because of the analogy from physics. This metric shows how far

the weights vector reached compared to the initial state. In other words, it shows the exploration of the agent. We expect it to be higher compared to the case when the exchange is off. Also, we expect that the agent with higher noise explores more in the long run.

6.5 Results Manager

Agent metrics in RL usually have high variance. To have statistically correct results we implemented the possibility to save the metrics for each agent of one execution as the result. With that functionality we processed multiple executions and obtained averaged results for different cases:

- DQN without exchange
- DQN with random exchange
- DQN with smart exchange
- A2C without exchange
- A2C with random exchange
- A2C with smart exchange

6.6 Visualizer

This part contains functionality to average on multiple executions using previous parts (Results Manager, Metrics Manager) and visualize different plots of the training process for analysis:

- The average reward for each iteration per noise.
- The average distance for each iteration per noise.
- The average loss for each iteration per noise.
- Agent reward for each iteration with different noises.
- Agent distance for each iteration with different noises.
- Agent loss for each iteration with different noises.
- Noise exchanges for each iteration per agent.
- Exchange rates for each agent.
- Average reward.

And there is one additional plot to visualize the performance of the agents after the training: average reward for each iteration per agent on the environment with zero noise.

Algorithm 1: Noise learning algorithm

```
input: k - number of agents
       n - number of training episodes
       exchangeType - type of exchange (No, Random, Smart)
       exchangeSteps - the number of steps to proceed for every exchange
       attempt
       warmUpSteps - the number of steps to proceed before the first
       exchange attempt
       envNoiseStep - the number to increase the noise for each next new
       environment
       r - number of agent's last rewards to get average for exchange
       d - coefficient to tune exchange probability
environments[k];
for i ← 0 to k do
    environments[i] = Environment(i * envNoiseStep)
end
agents[k];
for i ← 0 to k do
    agent[i] = Agent()
end
agentsResults[k];
for i ← 0 to k do
    agentsResults[i] = AgentResults()
end
for i ← 0 to n do
    for j ← 0 to k do
        agent = agent[j];
        env = environments[j];
        results = agentsResults[j];
        TrainAgent(agent, env, results);   /* Common RL train algorithm */
    end
    if i % exchangeSteps = 0 and i >= warmUpSteps then
        if exchangeType = No then
            continue;
        else if exchangeType = Random then
            PerformRandomExchange();          /* To compare with smart
             exchange */
        else if exchangeType = Smart then
            if int(i / exchangeSteps) % 2 = 0 then /* exchange direction   */
                for p ← 0 to k - 1 do
                    SmartExchangeAgents(p, p + 1, environments,
                     agentsResults, r, d);
                end
            else
                for p ← k − 1 to 0 do
                    SmartExchangeAgents(p - 1, p, environments, agentsResults,
                     r, d);
                end
            end
    end
end
```

Algorithm 2: SmartExchangeAgents function

input: i - index of the current agent
j - index of the next agent
environments - array of environments
agentsResults - array of agents results
r - number of agent's last rewards to get average for exchange
d - coefficient to tune exchange probability

```
noise1 = environments[i].noise;
noise2 = environments[j].noise;
reward1 = mean(agentsResults[i].rewards[-r:]);
reward2 = mean(agentsResults[j].rewards[-r:]);
prob = min(exp(d * (noise2 - noise1)(reward2 - reward1)), 1);
if random() < prob then
    SwapEnvironments(i, j)
end
```

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~-41.8°	~41.8°
3	Pole Velocity At Tip	-Inf	Inf

TABLE 6.1: Observation space of the CartPole environment.

Num	Action
0	Push cart to the left
1	Push cart to the right

TABLE 6.2: Action space of the CartPole environment.

Chapter 7

Evaluation

For the setup with CartPole environment and our implemented DQN and A2C agents we tried next hyperparameter setup for the noise learning algorithm:

- Agents number = 10
- Noise environment step = 0.1
- Training episodes = 5000
- Play episodes = 500
- Warm up steps = 30
- Exchange steps = 5
- Random exchange probability = 0.05
- Exchange delta = 0.1
- Exchange items reward count = 30

7.1 A2C

To train A2C agent next hyperparameter values were used:

- Learning rate = 0.001
- Discount rate = 0.99
- Batch size = number of steps achieved during one CartPole game (max = 500). I.e. if agent failed at the tenth step then the batch size for the current learning procedure was 10.
- Entropy loss factor = 0.0001
- Value loss factor = 1

7.1.1 No Exchange

We can see the results of agents learning on environments with different noise levels without exchange. Metrics are calculated as a moving average over the last 100 iterations every 50 iterations to visualize them on the charts. The figure 7.1 shows distances and scores for each noise level, it proves our assumption that increased noise level improves exploration in the long run. On the other hand, we can observe that at the beginning increased noise level slows down the learning procedure. The

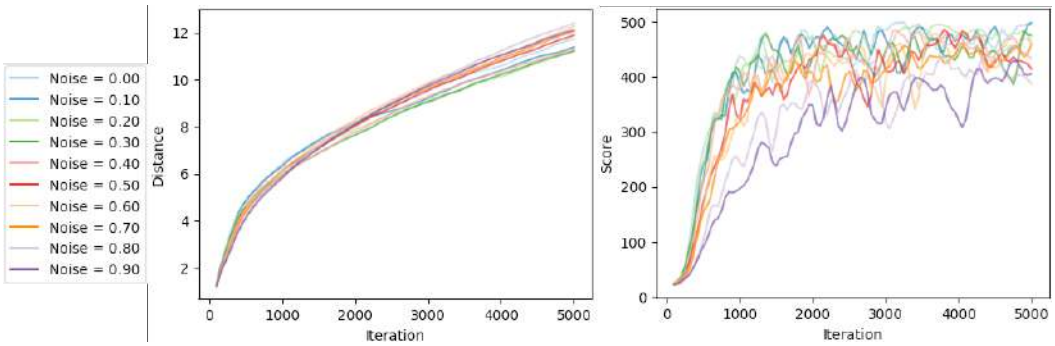

FIGURE 7.1: A2C. Average Distance and Score during training for 10 runs per Noise with No Exchange.

score chart also shows the same idea that increased noise level slows down the learning procedure from the start. We can see that during training even the best agents do not reach the best possible score of 500.

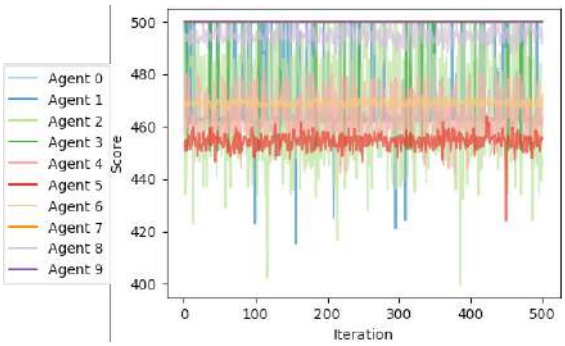

FIGURE 7.2: A2C. Average Score during play for 10 runs per Agent with No Exchange.

After the training, we tested the performance of all agents by playing the game with zero-noise for 500 episodes. Looking at play results (Figure 7.2) we can see that the best performance after training has agents that were trained with higher noises. It again proves the work made in the mentioned paper (Packer et al., 2018). Agents with a medium level of noise are pretty stable at the high score value around 450-460. Agents with a low level of noise have very unstable results with big drops from 500 to 420 scores, which means that they were overfitted during training. This figure basically shows that the noise is a pretty good regularization technique. The average score of the ensemble of all agents for play is 481.79 with standard deviation equals 65.35.

7.1.2 Random Exchange

With Random Exchange in Figure 7.3 we can see that there is no observable pattern in distances, which is expected. All curves are combined into one curve. The score chart looks similar to the case with No Exchange (Figure 7.1) just with increased volatility

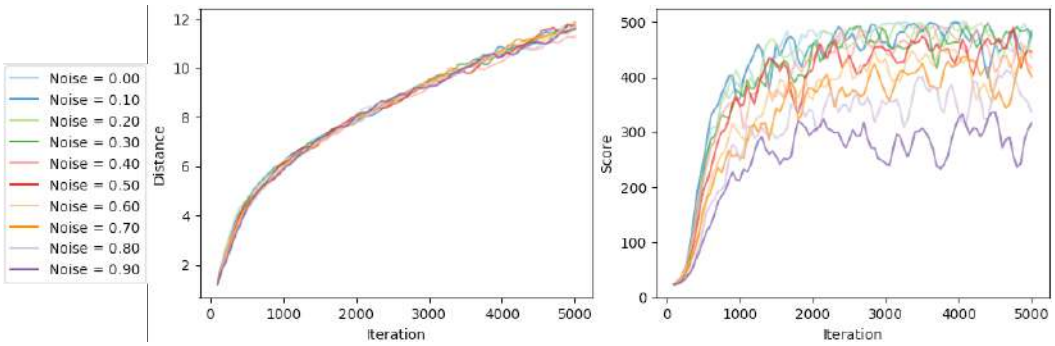

FIGURE 7.3: A2C. Average Distance and Score during training for 10 runs per Noise with Random Exchange.

because of the exchanges. On average scores with the random exchange are lower on 10 than scores with no exchange during the training.

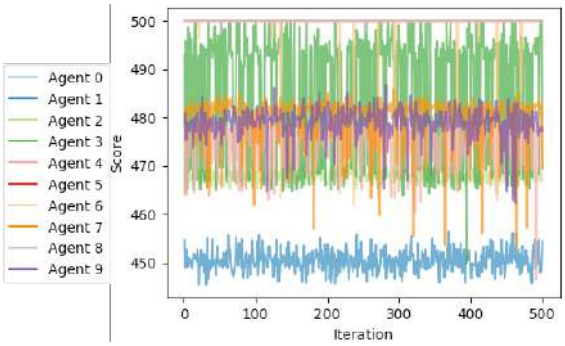

FIGURE 7.4: A2C. Average Score during play for 10 runs per Agent with Random Exchange.

Looking at play results (Figure 7.4) we can see that all agents have a pretty high score, but looking at each agent we can see different results compared to Figure 7.2. Agents have some instability because of random exchanges, though the average score of the ensemble of all agents is pretty similar to the situation with No Exchange - 483.78 ($\sigma = 61.3$), which is a little bit higher.

7.1.3 Smart Exchange

With Smart Exchange in Figure 7.5 we also see a similar picture in the distance chart to the case with Random Exchange (Figure 7.3). Scores picture looks much more stable for small noise in the long run than the case with No Exchange (Figure 7.1) and much better than the case with Random Exchange (Figure 7.3). Agents, that were exchanged to the environments with lower noises achieved the highest score. At the same time agents, that were exchanged to the environments with higher noises have a little bit worse picture than the case with No Exchange (Figure 7.1).

Looking at play results (Figure 7.6) we can see that all agents have a pretty high score again. Looking at individual agents we can see that agents are much more

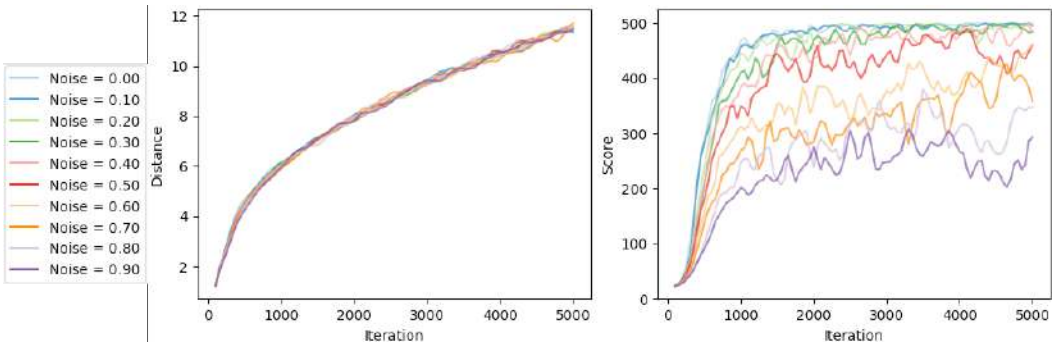

FIGURE 7.5: A2C. Average Distance and Score during training for 10 runs per Noise with Smart Exchange.

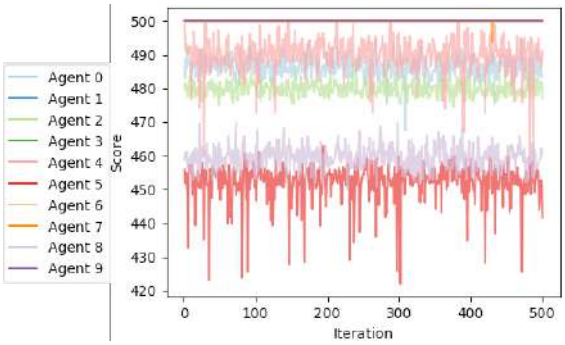

FIGURE 7.6: A2C. Average Score during play for 10 runs per Agent with Smart Exchange.

stable than in Figure 7.2. As the result average score of the ensemble of all agents is higher than in Random or No Exchange cases: 486.76 ($\sigma = 49.72$).

7.2 DQN

To train DQN agent next hyperparameter values were used:

- Learning rate = 0.001
- Discount rate = 1
- Exploration rate = 0.02
- Max exploration rate = 0.02
- Min exploration rate = 0.02
- Exploration Rate Decay = 0.9996
- Memory size = 50000
- Batch size = 32

7.2.1 No Exchange

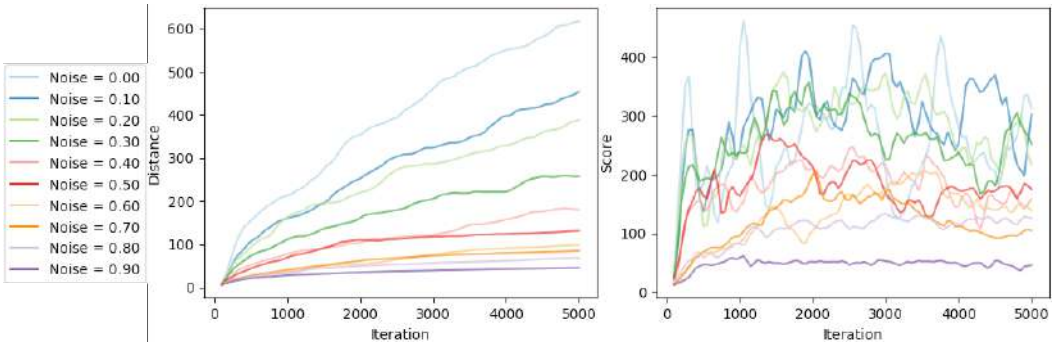

FIGURE 7.7: DQN. Average Distance and Score during training for 10 runs per Noise with No Exchange.

Looking at Figure 7.7 we can see that our implementation of the DQN agent is performing much worse than the A2C agent. With that said it is irrelevant to compare DQN visualizations with A2C. We can treat the experiments with DQN as the task of having an environment with higher complexity and agent that cannot fully solve the environment. We are interested in comparing results with different types of exchanges.

Looking at distances we see a different picture than we saw with the A2C agent. Big noises do not force agents to explore more. This may be a result of what we see about scores: agents with high noises have a very bad performance. Agents with small noise have better performance but it looks like they are overfitting on some biased sample of environment states time to time.

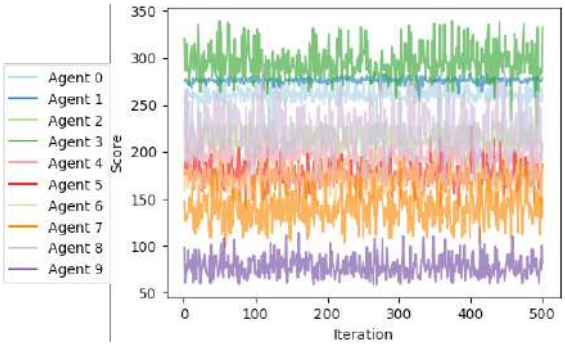

FIGURE 7.8: DQN. Average Score during play for 10 runs per Agent with No Exchange.

Only the agent with low noise achieved decent results (Figure 7.8). The agent with the highest noise was not able to train. The average score of the ensemble of all agents for play is 202.64 ($\sigma = 170.46$).

7.2.2 Random Exchange

With Random Exchange in Figure 7.9 we can see that distances do not have an observable pattern, which is expected. Strong ordering presented in Figure 7.9

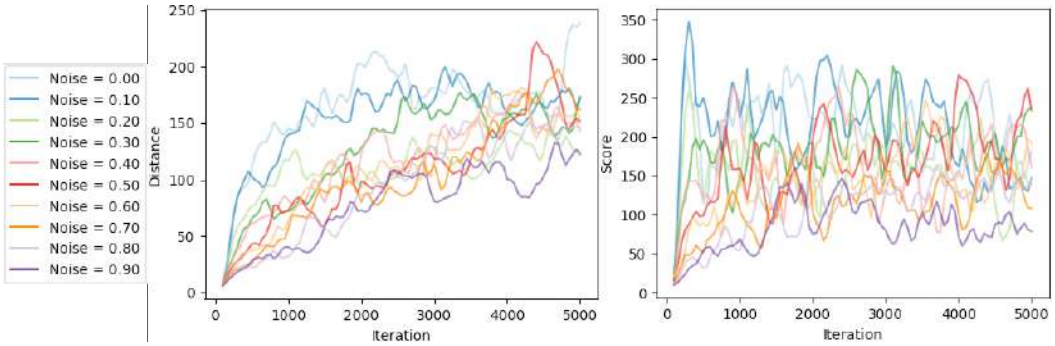

FIGURE 7.9: DQN. Average Distance and Score during training for 10 runs per Noise with Random Exchange.

changed to random.

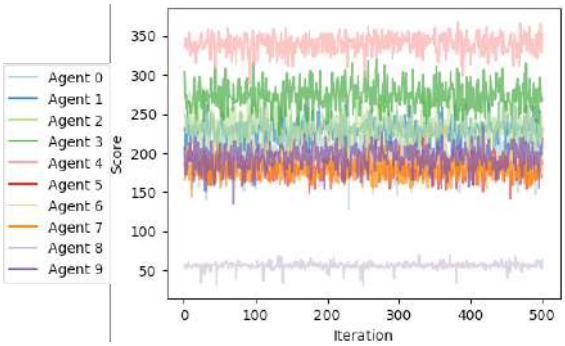

FIGURE 7.10: DQN. Average Score during play for 10 runs per Agent with Random Exchange.

Similarly to A2C the average score of ensemble of all agents for play is a little bit higher - 207.78 ($\sigma = 179.14$) - for Random Exchange (Figure 7.10) comparing to No Exchange - 202.64 (Figure 7.8).

7.2.3 Smart Exchange

With Smart Exchange in Figure 7.11 we also see a similar picture on the distance chart to the case with Random Exchange (Figure 7.9), but overall total exploration looks better. Also, similar to A2C case (Figure 7.5), score looks more stable for small noise than the case with No Exchange (Figure 7.7) and with Random Exchange (Figure 7.9). Agents, that were exchanged to the environments with lower noises achieved the best score. At the same time agents, that were exchanged to the environments with higher noises have a worse picture than the case with No Exchange (Figure 7.7).

Looking at play results (Figure 7.12) we can see that almost all agents combined to a group except the Agent 9. The average score of the ensemble of all agents is 166.76 ($\sigma = 166.77$).

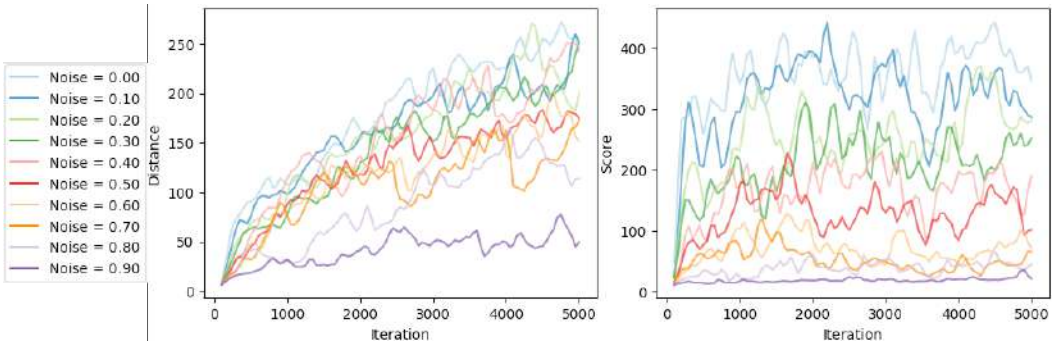

FIGURE 7.11: DQN. Average Distance and Score during training for 10 runs per Noise with Smart Exchange.

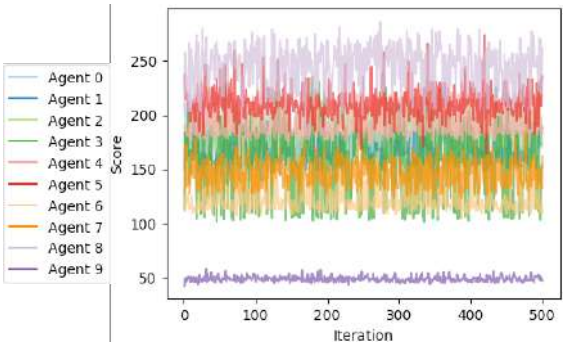

FIGURE 7.12: DQN. Average Score during play for 10 runs per Agent with Smart Exchange.

Chapter 8

Conclusions

We implemented a few RL agents - DQN and A2C -, environment wrapper for noise addition and a noise learning algorithm based on the replica-exchange with the Metropolis-Hastings scheme. We ran preliminary experiments that qualitatively validate several hypotheses and open up multiple directions for further research. In Chapter 5 we formulated a few hypotheses, let us start with them.

8.1 Noise improves exploration

To analyze exploration abilities we implemented a metric called distance (or diffusion). It is calculated as Euclidean distance between initial weights matrices and weight matrices at the exact iteration.

A2C (Figure 7.1) shows that noise slows down the learning speed in the initial phase and at the same time improves exploration a little bit in the long run. The CartPole environment is a pretty simple environment with only 4 state dimensions. The more complex environment may show a better result in having more exploration potential.

On the other hand, DQN (Figure 7.7) does not show the same result partly because implemented DQN shows generally poor performance in such a simple environment. We also observe the effect of noise which is inverted to what we see in the A2C and one needs to adjust replica-exchange trick for such a case. Hence, results from two different RL agents cannot be compared.

We do not have enough results to say that we empirically approved the idea that noise improves exploration. We have some small trend (Figure 7.1) towards our hypothesis and we have results that show that noise can also make agents not able to train at all. We need more experiments with improved DQN and with more complex environments to empirically approve the hypothesis.

We, also, proved again (Figure 7.2) that agent, trained on the environment with high noises, have better and stable results (Packer et al., 2018).

8.2 Metropolis-Hastings replica-exchange improves RL training

To analyze better the exchange impact we ran three types of experiments: no exchange, random exchange, smart exchange (based on Metropolis-Hastings scheme). We further compare resulting trained agents via analysis of their performance in the new rounds of the game.

With the no exchange case, we obtained baseline metrics to compare with. For A2C we saw that during training the agents with smaller noise have better results (Figure 7.1). This is reflected in the fact that between the games such agents explore

high noise environments and, hence, perform better in the low noise environment. Quite the opposite results we observed during the play phase (Figure 7.2). For DQN the agent could not handle the noise and showed bad results during both training (Figure 7.7) and play (Figure 7.8) phases hence we exclude DQN results from further analysis.

With the random exchange case, we learned how exchanges impact the agents comparing different metrics with no exchange case. With A2C we got slightly better results (Figure 7.4, Figure 7.10) comparing with no exchange case (Figure 7.2, Figure 7.8). That implies that simply exchanging the environment gives the agent additional augmented (noised) data and improves the whole ensemble.

For the smart exchange, the results (Figure 7.6) are slightly improved and more stable compared to no exchange (Figure 7.2) or random exchange (Figure 7.4), which means that the Metropolis-Hastings exchange scheme improved the performance of the agent.

For DQN with the smart exchange the results (Figure 7.12) are worse than in both cases: no exchange (Figure 7.8) and random exchange (Figure 7.10). It is unclear the reason for such results for DQN. It may be because of the performance of the agent or maybe the results of the agent are too volatile and more experiments are necessary to draw conclusions. Thus, further research is needed in this direction.

Agent, Exchange Type	Score
A2C, No Exchange	481.79 ± 65.35
A2C, Random Exchange	483.78 ± 61.3
A2C, Smart Exchange	486.76 ± 49.72
DQN, No Exchange	202.64 ± 170.46
DQN, Random Exchange	207.78 ± 179.14
DQN, Smart Exchange	166.76 ± 166.77

TABLE 8.1: Average scores of ensemble of all agents during play phase.

Table 8.1 shows the result comparison of different exchange types for different agents. We showed that the Metropolis-Hastings exchange scheme may improve the performance of the agent, but it depends on the complexity of the environment and capacity of the agent. With the agent, which is unable to perform in the environment with big noise, it is unclear if the proposed solution can improve the results.

Chapter 9

Future Work

The work done gave us a few answers:

- Noise slightly improves exploration, but only under special circumstances when the agent is able to handle the noise and train.
- Metropolis-Hastings replica exchange improves training, but only if the agent is able to perform under the big noises.

And these answers are working only under specific conditions. We need to understand these conditions better and to do that there is a lot of work that can be done:

- In this work, we had a weak DQN agent, which was treated as the case of having an environment with higher complexity. On the other hand, the A2C agent reached the highest reward of 500. Try a different environment with actually much higher complexity, where the agent cannot reach the highest reward and compare the results.
- We used noise on the environment side. It is external noise to the agent. It is interesting to compare the results with internal noise, like dropout or regularization.
- As we have shown in the work, under specific conditions noise improves the exploration. But the agent has it's internal ability to explore (different agents implement it in a different way). Find a way to compare these two exploration techniques and investigate under which conditions the one technique dominates the other.
- Investigate exploration potential of the environment: some environments have much higher exploration potential than others. Find a way to evaluate the exploration potential of the environment. How exploration potential affects the performance of the agents trained with smart exchanges?
- Training multiple agents at the same time in different environments create the opportunity of parallelizing the process or making it asynchronous. In this work, it was computationally inefficient to parallelize the process, because exchanges were happening too often and the exchange procedure had to wait for all agents. For environments with more complexity, RL agents may use much more complicated Neural Networks with more layers and neurons, which may lead to having a bigger exchange interval. In such case parallelization of the agents should improve the training time. Or change the exchange approach to make it asynchronous by removing the synchronization point (waiting for all agents).

Appendix A

Additional Charts

This appendix contains more detailed experiments charts.

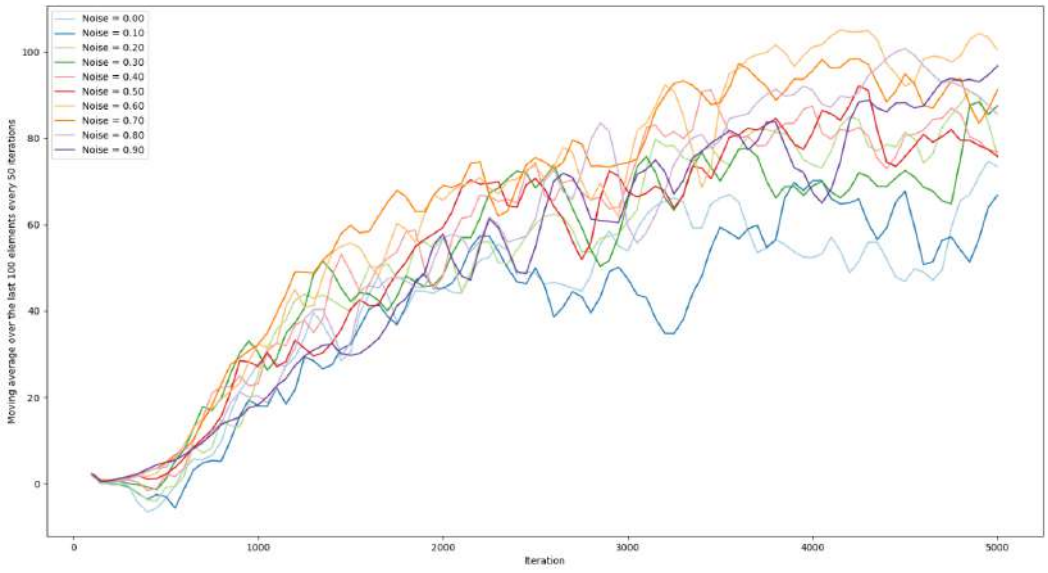

FIGURE A.1: A2C. Average Loss for 10 runs per Noise with No Exchange.

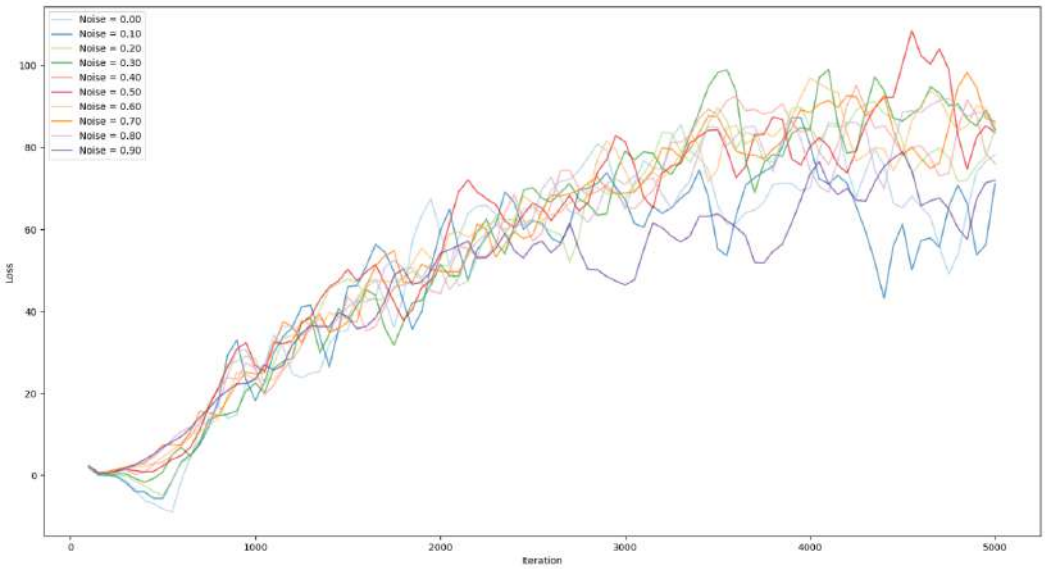

FIGURE A.2: A2C. Average Loss for 10 runs per Noise with Random Exchange.

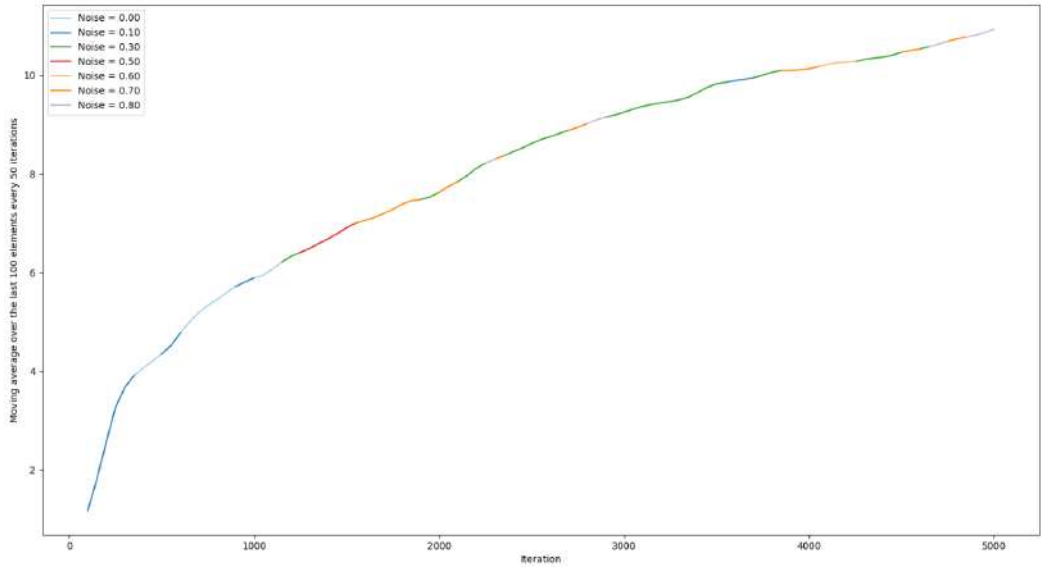

FIGURE A.3: A2C. Average Distance for 1 run for agent 1 with Random Exchange.

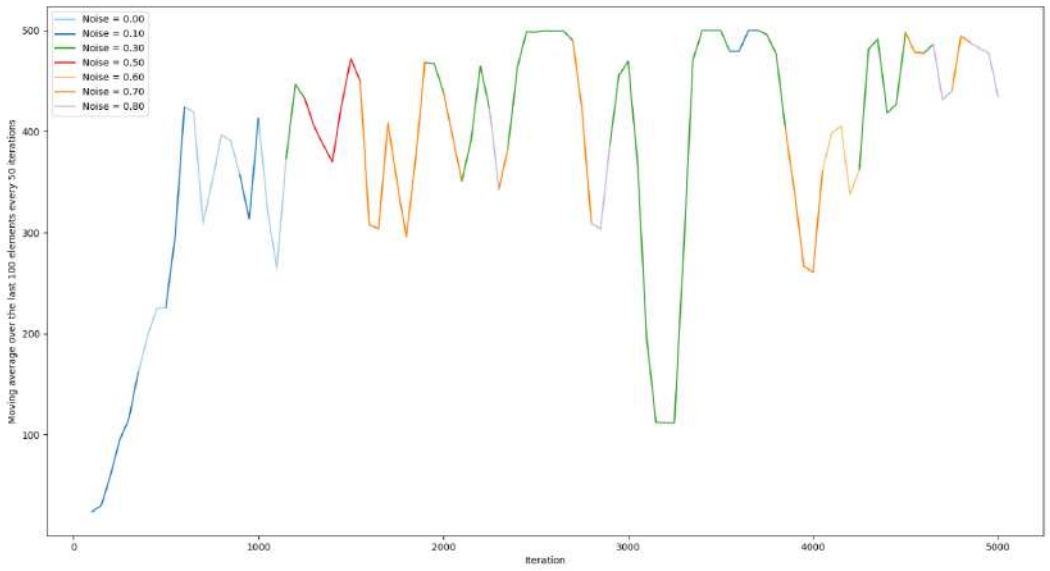

FIGURE A.4: A2C. Average Score for 1 run for agent 1 with Random Exchange.

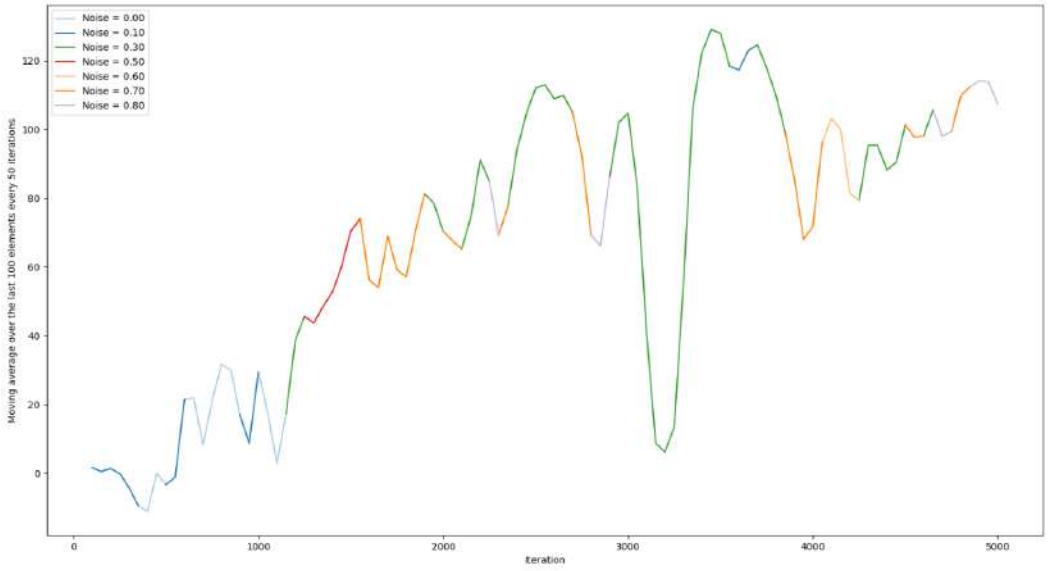

FIGURE A.5: A2C. Average Loss for 1 run for agent 1 with Random Exchange.

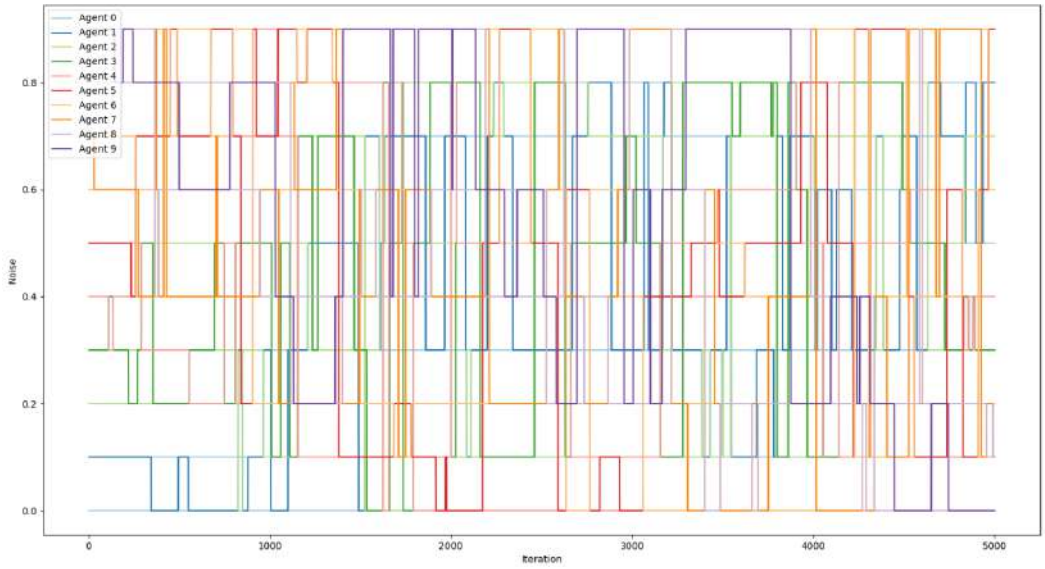

FIGURE A.6: A2C. Noise exchanges for 1 run per agent with Random Exchange.

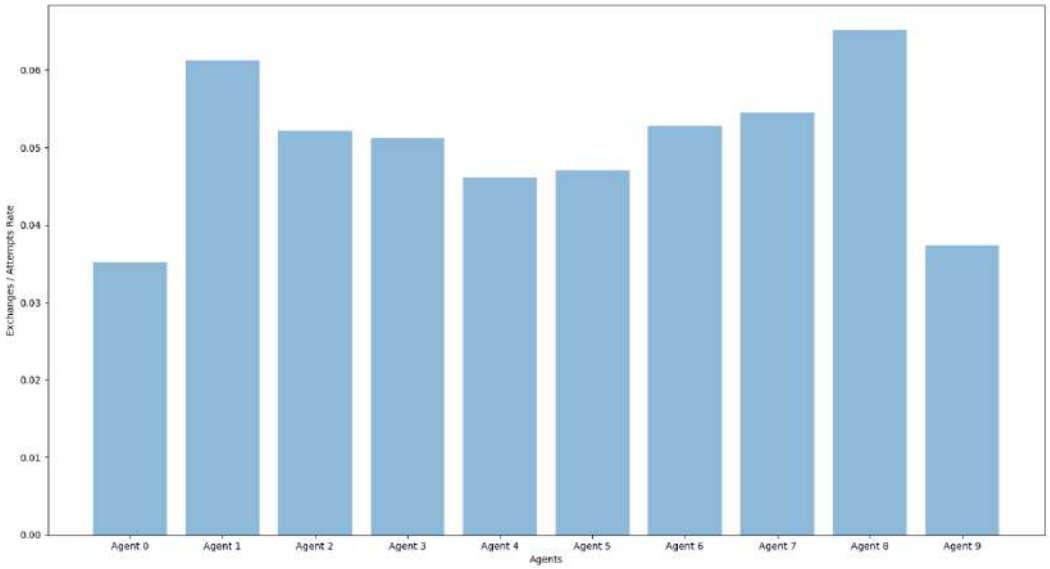

FIGURE A.7: A2C. Exchange rates for 10 runs for each agent with Random Exchange.

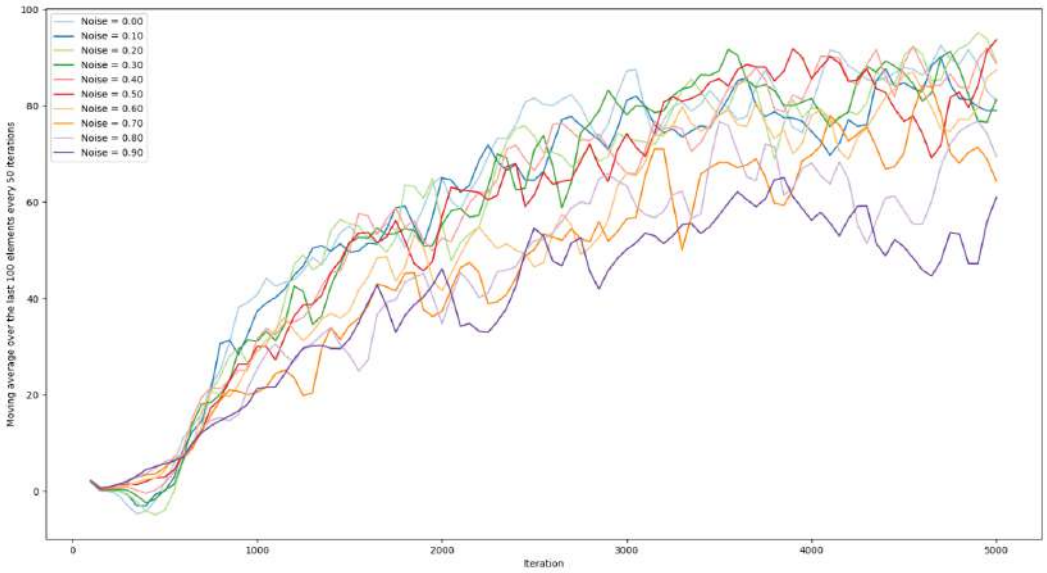

FIGURE A.8: A2C. Average Loss for 10 runs per Noise with Smart Exchange.

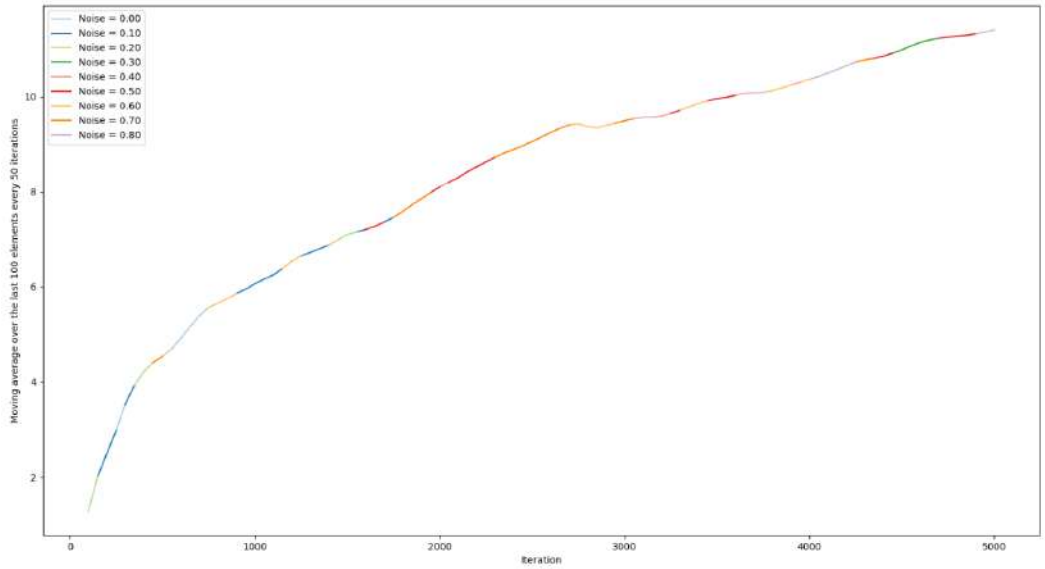

FIGURE A.9: A2C. Average Distance for 1 run for agent 1 with Smart Exchange.

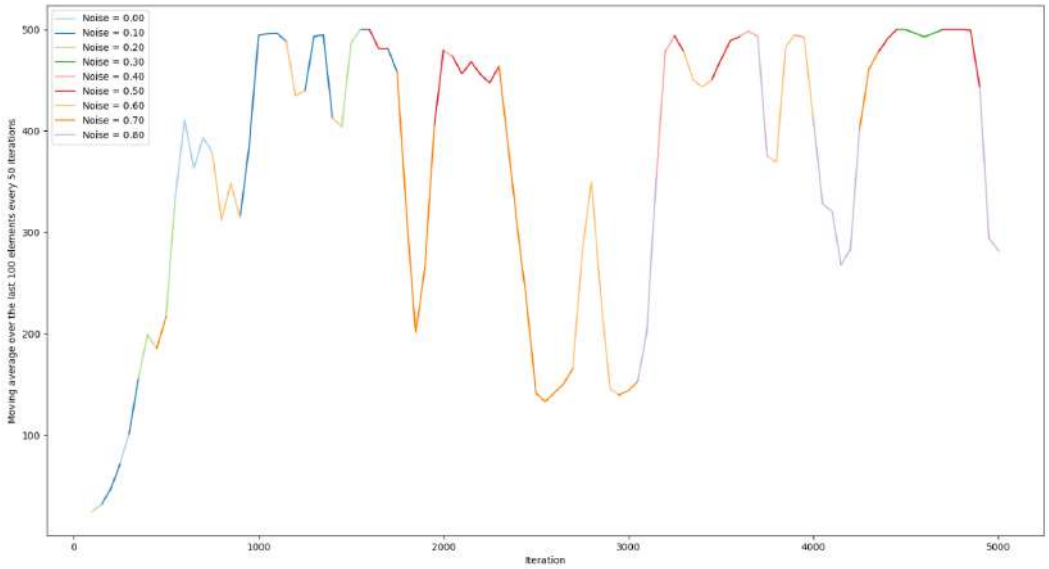

FIGURE A.10: A2C. Average Score for 1 run for agent 1 with Smart Exchange.

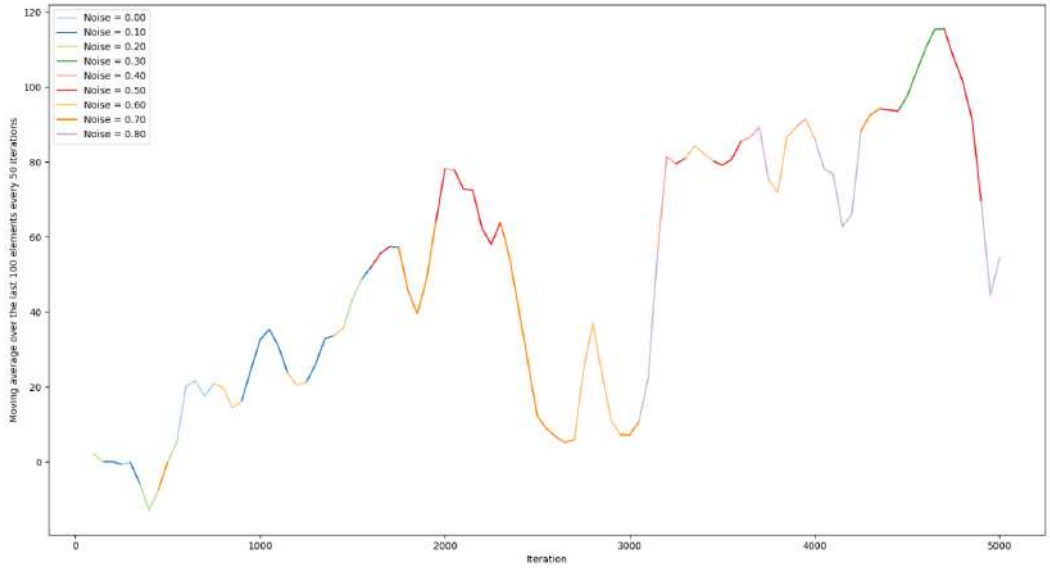

FIGURE A.11: A2C. Average Loss for 1 run for agent 1 with Smart Exchange.

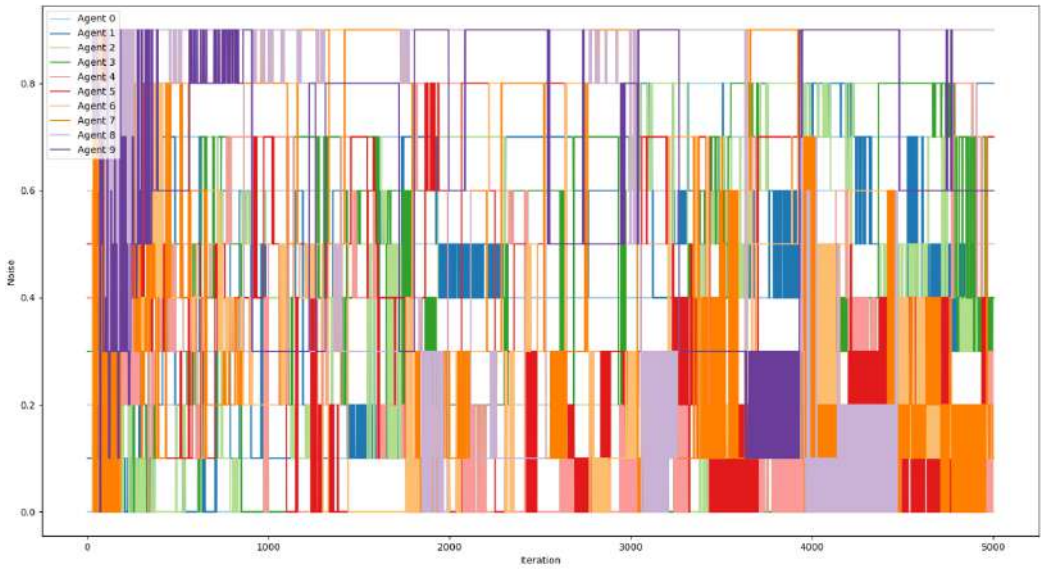

FIGURE A.12: A2C. Noise exchanges for 1 run per agent with Smart Exchange.

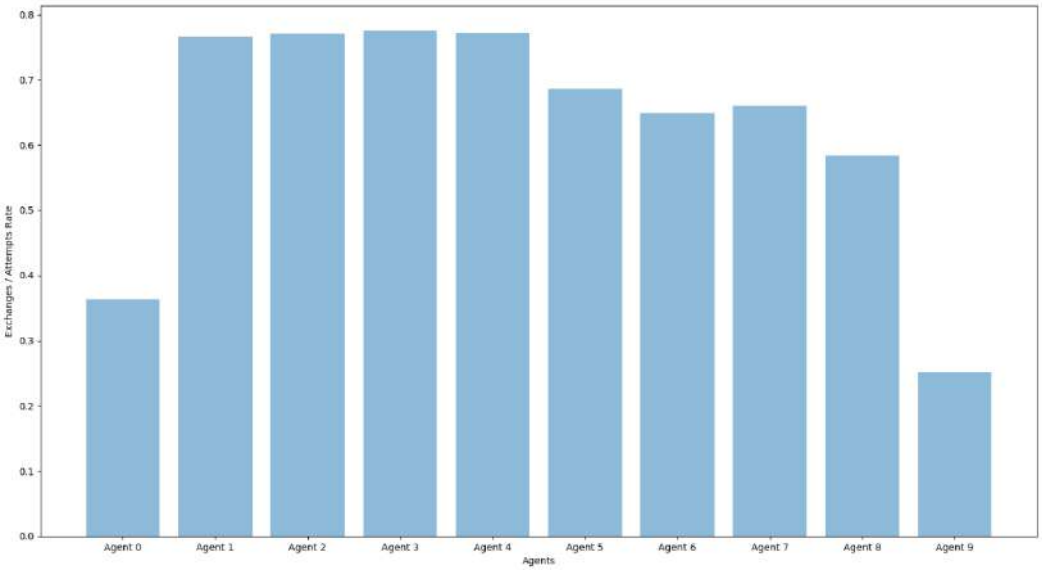

FIGURE A.13: A2C. Exchange rates for 10 runs for each agent with Smart Exchange.

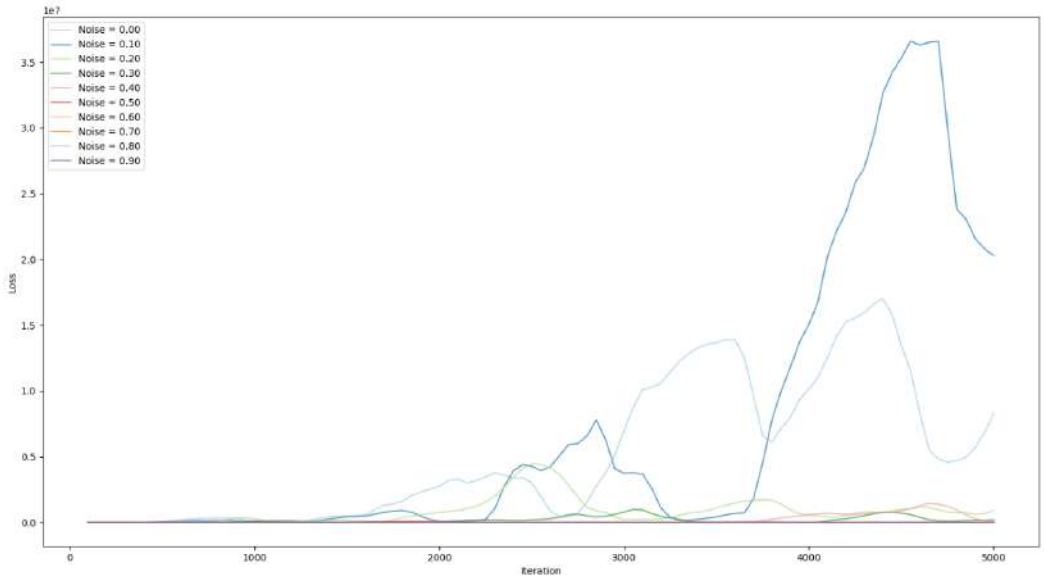

FIGURE A.14: DQN. Average Loss for 10 runs per Noise with No Exchange.

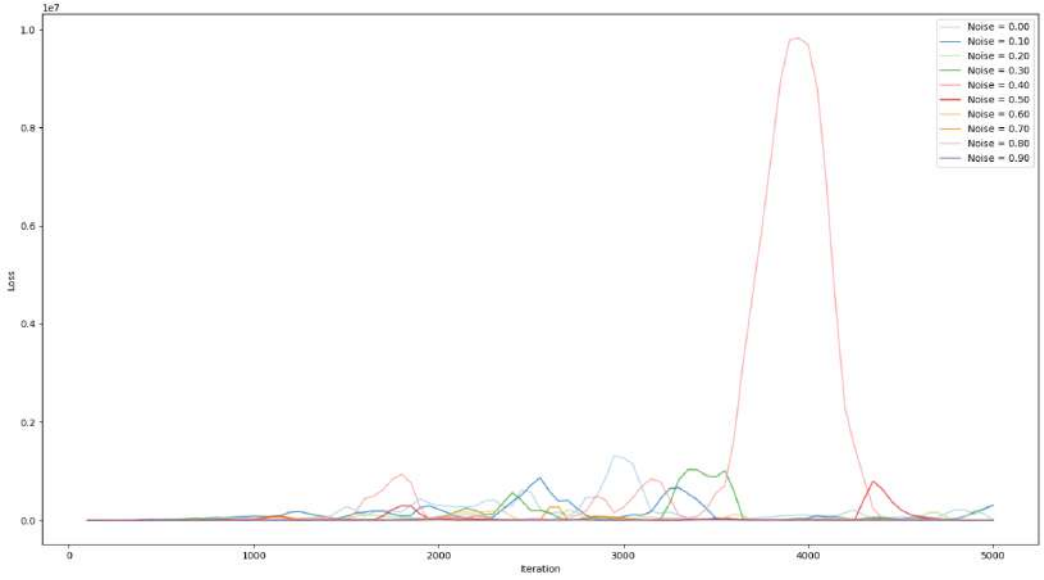

FIGURE A.15: DQN. Average Loss for 10 runs per Noise with Random Exchange.

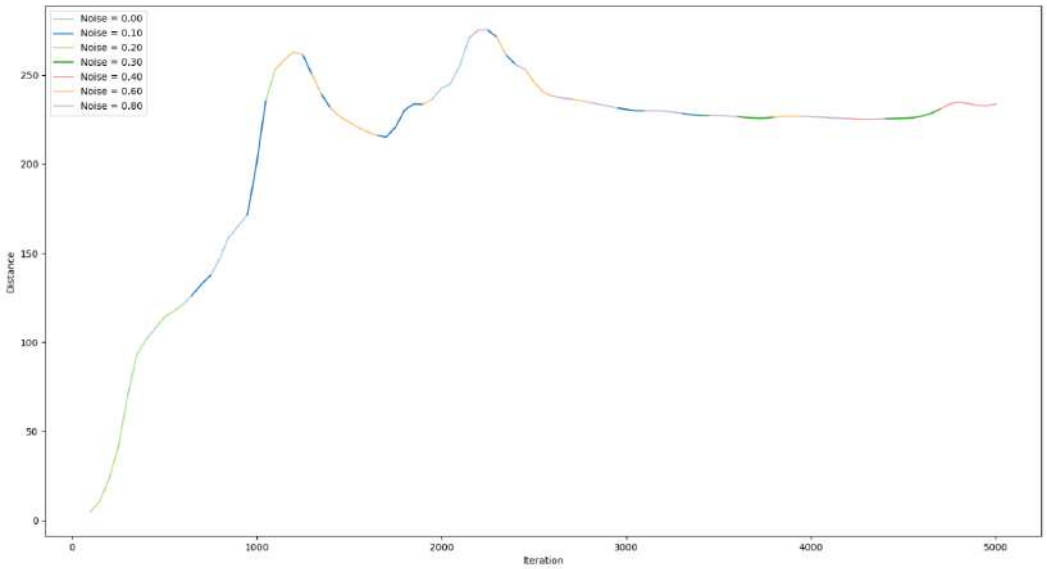

FIGURE A.16: DQN. Average Distance for 1 run for agent 1 with Random Exchange.

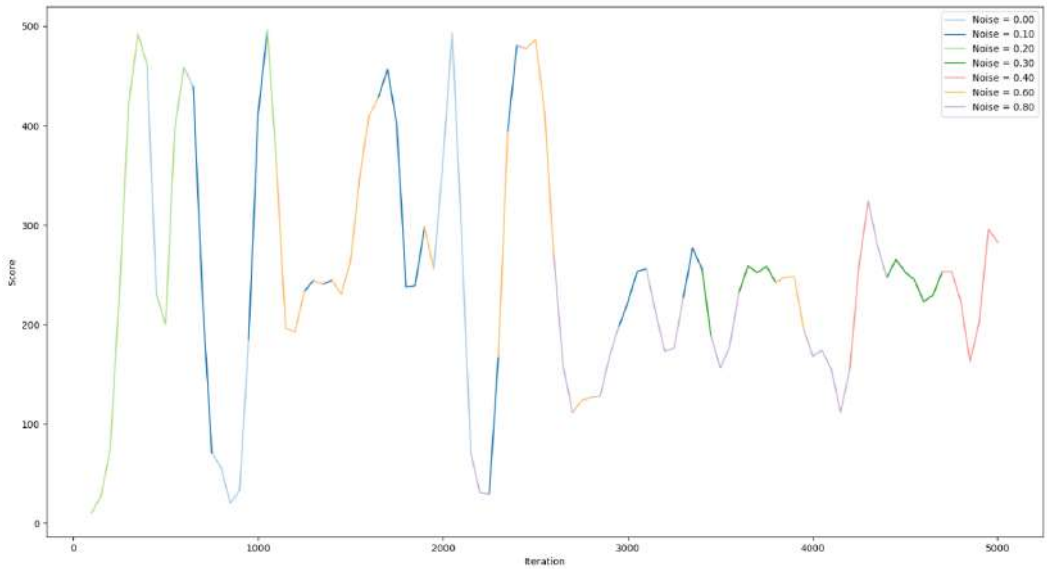

FIGURE A.17: DQN. Average Score for 1 run for agent 1 with Random Exchange.

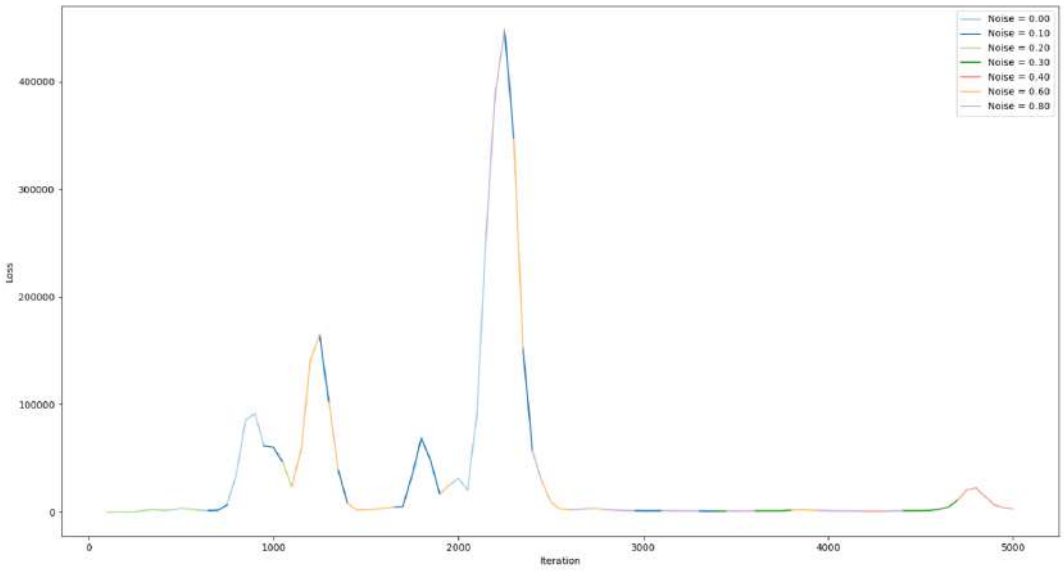

FIGURE A.18: DQN. Average Loss for 1 run for agent 1 with Random Exchange.

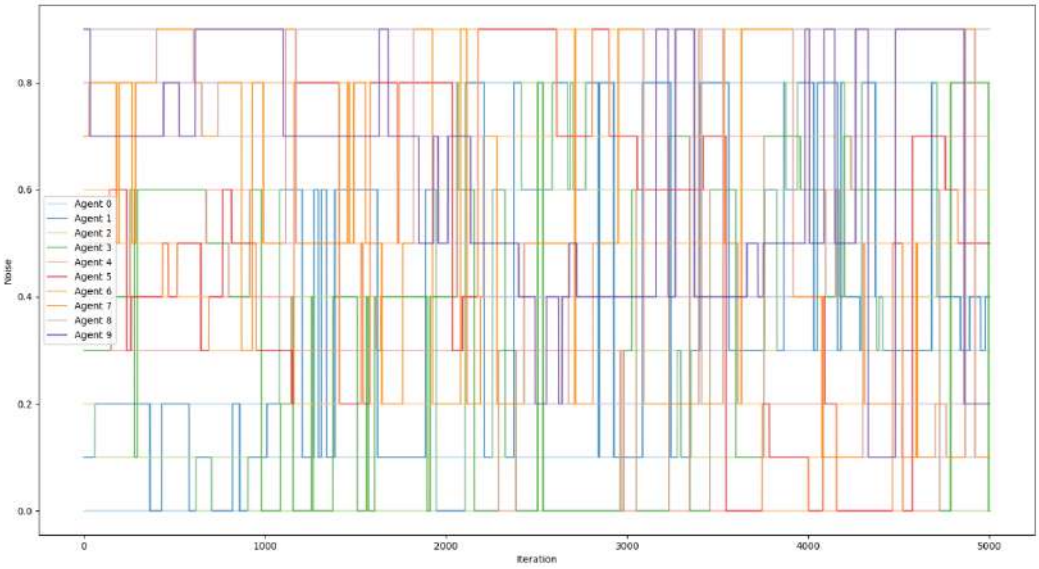

FIGURE A.19: DQN. Noise exchanges for 1 run per agent with Random Exchange.

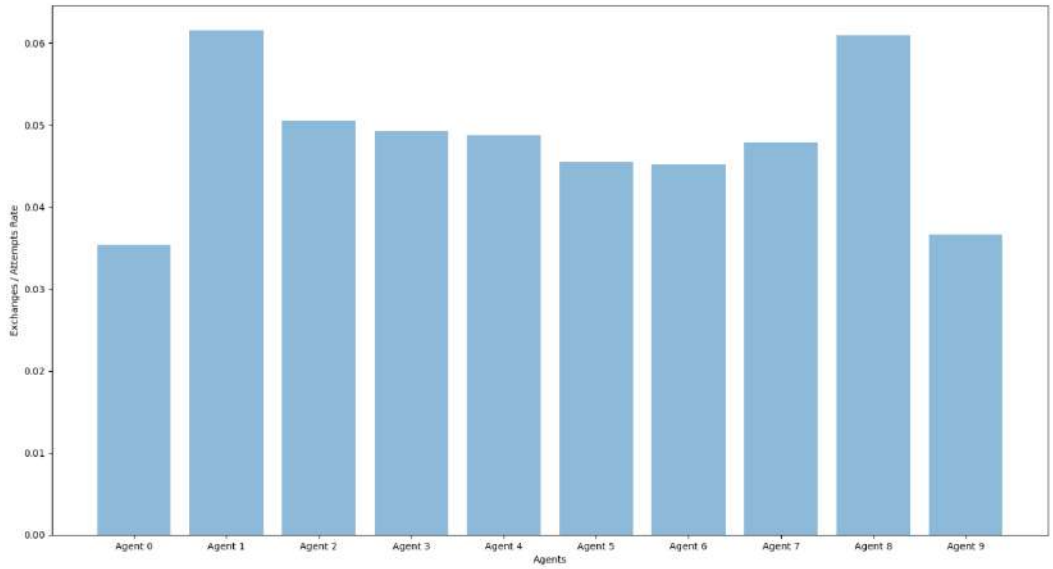

FIGURE A.20: DQN. Exchange rates for 10 runs for each agent with Random Exchange.

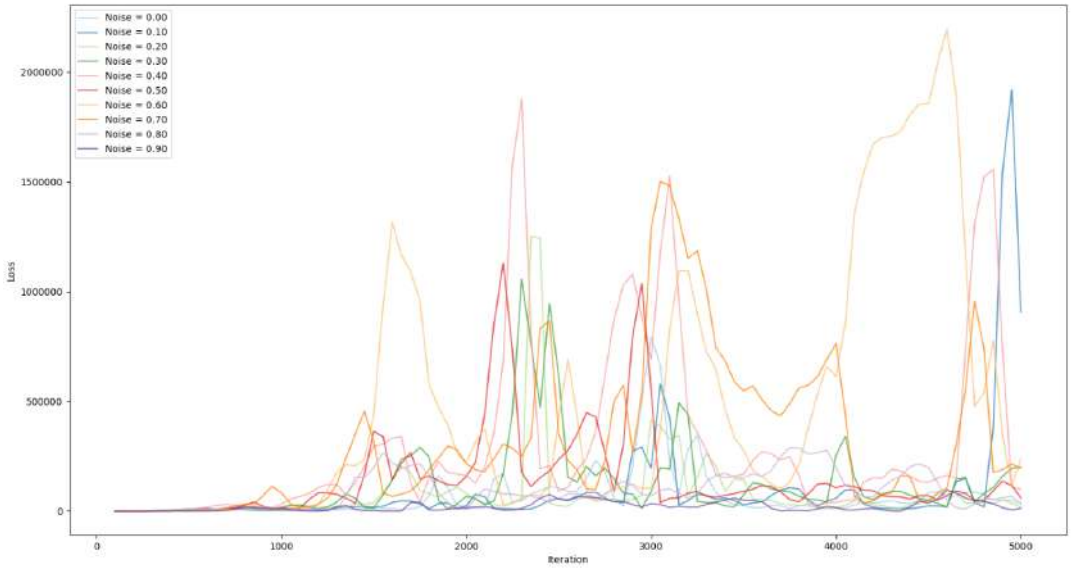

FIGURE A.21: DQN. Average Loss for 10 runs per Noise with Smart Exchange.

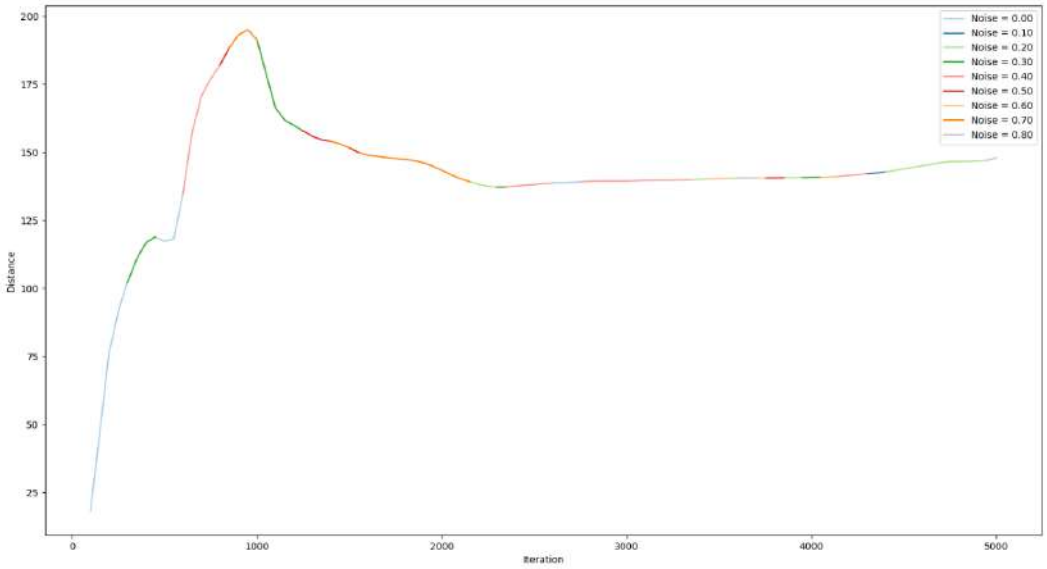

FIGURE A.22: DQN. Average Distance for 1 run for agent 1 with Smart Exchange.

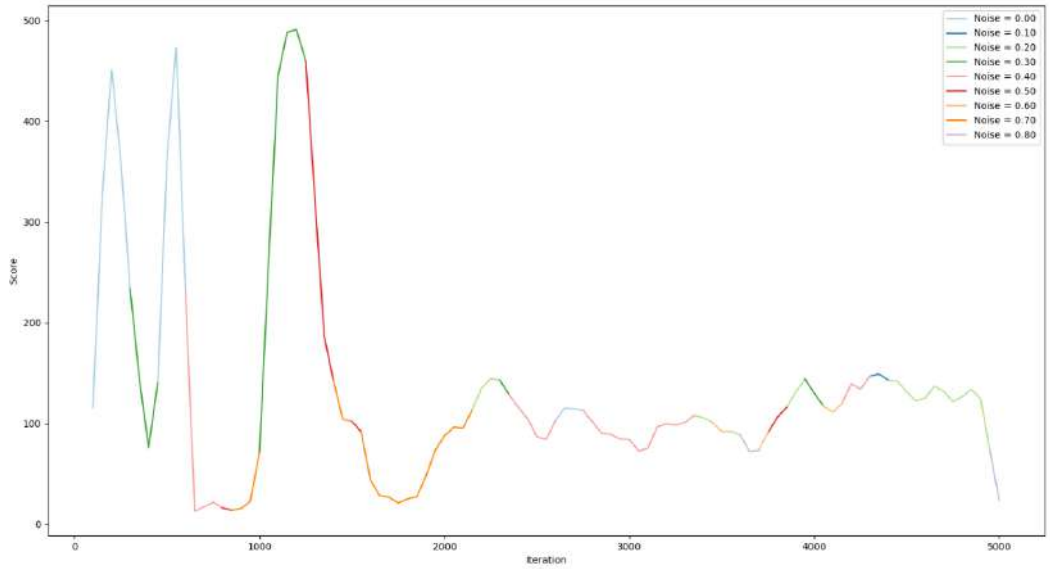

FIGURE A.23: DQN. Average Score for 1 run for agent 1 with Smart Exchange.

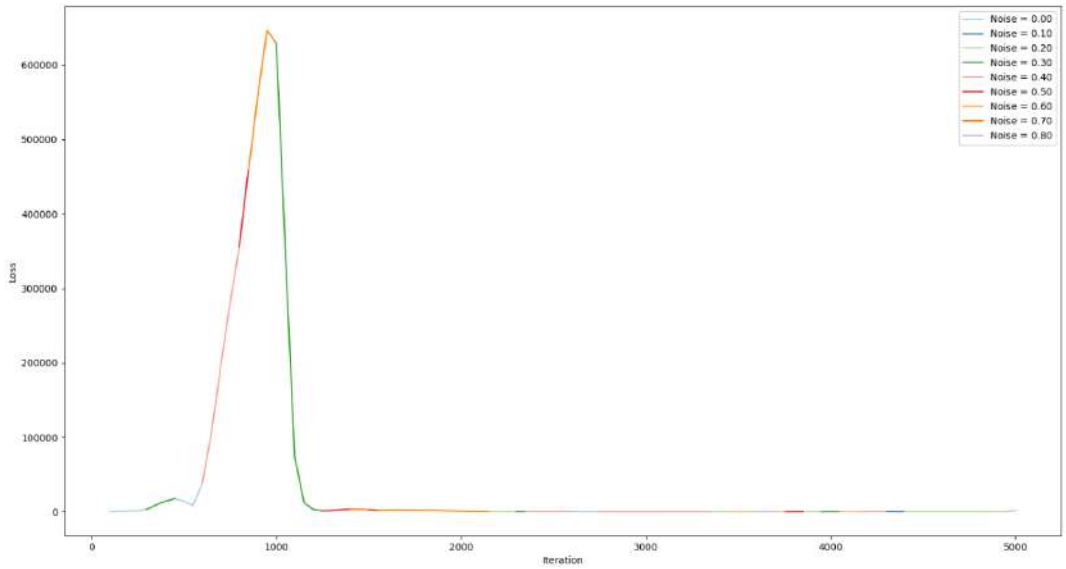

FIGURE A.24: DQN. Average Loss for 1 run for agent 1 with Smart Exchange.

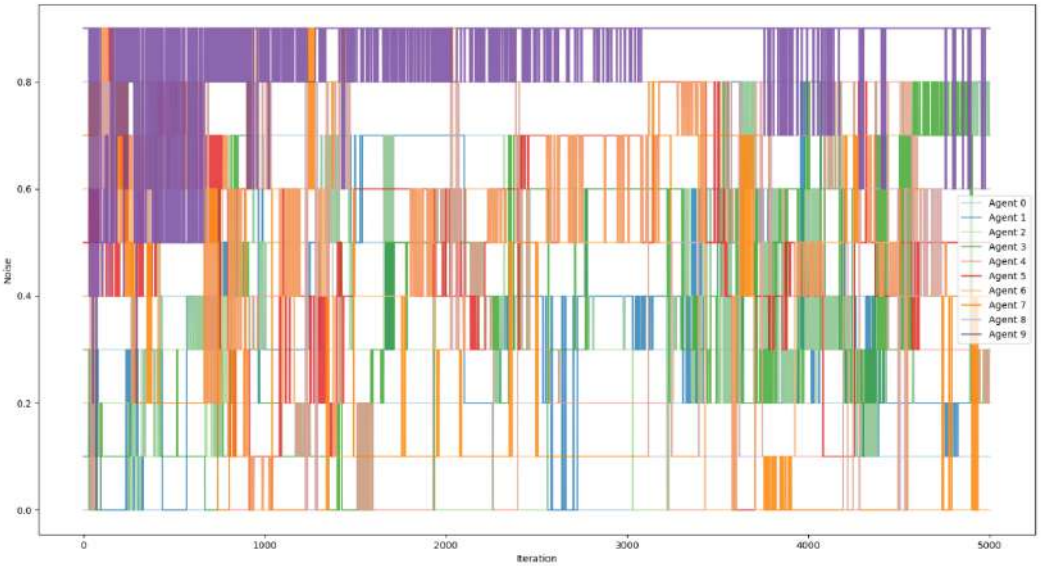

FIGURE A.25: DQN. Noise exchanges for 1 run per agent with Smart Exchange.

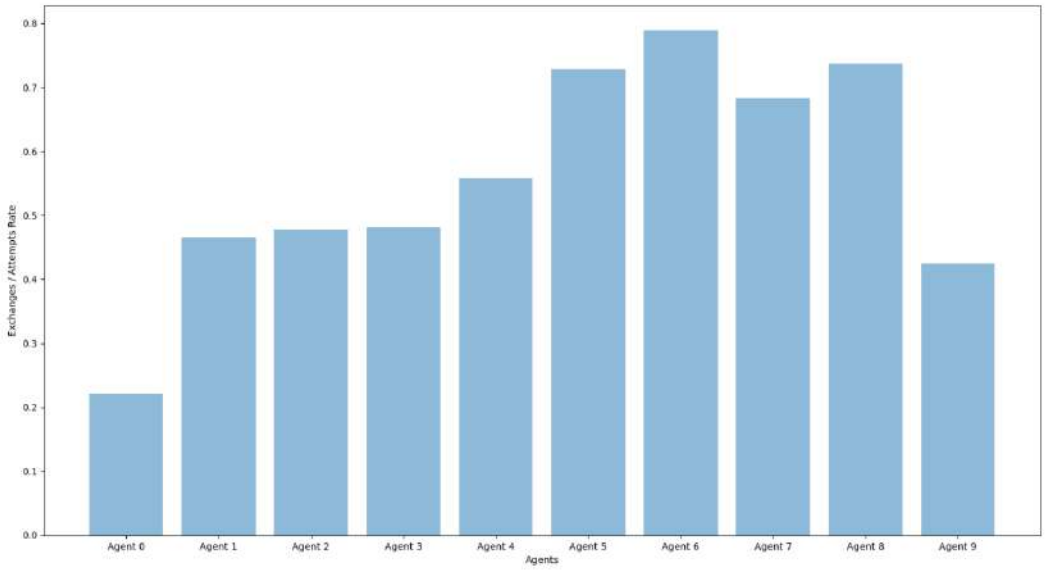

FIGURE A.26: DQN. Exchange rates for 10 runs for each agent with Smart Exchange.

Bibliography

Arel, I et al. (June 2010). "Reinforcement learning-based multi-agent system for network traffic signal control". In: *IET Intel. Transport Syst.* 4.2, pp. 128–135.

Aubret, Arthur, Laetitia Matignon, and Salima Hassas (Aug. 2019). "A survey on intrinsic motivation in reinforcement learning". In: arXiv: 1908.06976 [cs.LG].

Barto, Andrew G (2013). "Intrinsic Motivation and Reinforcement Learning". In: *Intrinsically Motivated Learning in Natural and Artificial Systems*. Ed. by Gianluca Baldassarre and Marco Mirolli. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 17–47.

Bishop, Christopher M (2006). *Pattern recognition and machine learning*. springer.

Cobbe, Karl et al. (Dec. 2018). "Quantifying Generalization in Reinforcement Learning". In: arXiv: 1812.02341 [cs.LG].

DeepMind. *Reinforcement learning lectures by DeepMind*. URL: https://www.youtube.com/playlist?list=PL7-jPKtc4r78-wCZcQn5IqyuWhBZ8fOxT.

Dmitri Glusco, Mykola Maksymenko (2019). *Replica Exchange For Multiple-Environment Reinforcement Learning*. URL: github.com/d4glushko/ReinforcementLearning.

Jaderberg, Max et al. (Nov. 2017). "Population Based Training of Neural Networks". In: arXiv: 1711.09846 [cs.LG].

Jin, Junqi et al. (Feb. 2018). "Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising". In: arXiv: 1802.09756 [stat.ML].

Kober, Jens, J Andrew Bagnell, and Jan Peters (Sept. 2013). "Reinforcement learning in robotics: A survey". In: *Int. J. Rob. Res.* 32.11, pp. 1238–1274.

Kulkarni, Tejas D et al. (2016). "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation". In: *Advances in neural information processing systems*, pp. 3675–3683.

Mao, Hongzi et al. (2016). "Resource management with deep reinforcement learning". In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56.

Minar, Matiur Rahman and Jibon Naher (July 2018). "Recent Advances in Deep Learning: An Overview". In: arXiv: 1807.08169 [cs.LG].

Mnih, Volodymyr et al. (June 2016). "Asynchronous Methods for Deep Reinforcement Learning". en. In: *International Conference on Machine Learning*, pp. 1928–1937.

OpenAI. *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. https://gym.openai.com/. Accessed: 2019-12-15.

OpenAI. *OpenAI Spinning Up: An educational resource produced by OpenAI that makes it easier to learn about deep reinforcement learning*. URL: https://spinningup.openai.com/en/latest/index.html.

Packer, Charles et al. (Oct. 2018). "Assessing Generalization in Deep Reinforcement Learning". In: arXiv: 1810.12282 [cs.LG].

Pushkarov, Vlad et al. (Sept. 2019). "Training Deep Neural Networks by optimizing over nonlocal paths in hyperparameter space". In: arXiv: 1909.04013 [cs.LG].

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, Andrew G Barto, and Others (1998). *Introduction to reinforcement learning*. Vol. 2. MIT press Cambridge.

Vinyals, Oriol et al. (Nov. 2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". en. In: *Nature* 575.7782, pp. 350–354.

Zheng, Guanjie et al. (2018). "DRN: A deep reinforcement learning framework for news recommendation". In: *Proceedings of the 2018 World Wide Web Conference*, pp. 167–176.

Zhou, Zhenpeng, Xiaocheng Li, and Richard N Zare (Dec. 2017). "Optimizing Chemical Reactions with Deep Reinforcement Learning". en. In: *ACS Cent Sci* 3.12, pp. 1337–1344.