Fourth Year Project Report

Communication Efficient Distributed Reinforcement Learning

Edward Gunn

Supervisor: Konstantinos Gatsis

May 17, 2023

Abstract

Distributed systems are ubiquitous in the modern, automated world. These systems collect large amounts of data from which it is possible to learn how to act more efficiently. Agents within the system can communicate in order to learn more efficiently than any individual and to overcome computational bottlenecks associated with working on the edge, where analysis is performed on device rather than transporting the data to the cloud for analysis. However, when distributed over a network the rate at which agents can communicate becomes a bottleneck for their rate of learning. It is therefore desirable to develop communication efficient algorithms for these use cases. A class of algorithms based on reinforcement learning show promising results in this domain, allowing for collaboration between agents to optimize a reward signal. I consider a number of existing algorithms of this nature, discussing their respective strengths and weaknesses and conducting numerical comparison between them. As well as this, I propose a new algorithm, evolution strategies with probabilistic communication (ESPC), to adjustably reduce the number of messages sent by evolution strategies (ES), one of the best performing and most versatile algorithms. I provide justification for this algorithm as well as a direct comparison between the ES and ESPC. I show that in some scenarios ESPC performs better than ES in terms of both quality of learning and amount of communication.

4YP Risk Assessment 2022

Description of 4YP task or aspect being risk assessed here: Distributed Reinforcement learning on the edge, office work project		4YP Project Number:
Site, Building & Room Number: None	Other Relevant risk Assessments: None	
Assessment undertaken by: Edward Gunn	Signed:	Date:03/11/2022
Assessment Supervisor: Konstantinos Gatsis	Signed:	Date:03/11/2022

Assessing the Risk*

You can do this for each hazard as follows:

- Consequences: Decide how severe the outcome for each hazard would be if something went wrong (i.e. what are the Consequences?) Death would be "Severe", a minor cut to a finger could be regarded as "Insignificant".
- Likelihood: How likely are these Consequences to actually happen? Highly likely? Remotely likely, or somewhere in between?
- Risk Rating: Start at the left of the coloured Matrix. On your chosen Consequences row, read across until you are in the correct Likelihood column for the hazard in question. For example, an outcome with Severe consequences but with a Low probability of actually happening equates to a Medium risk overall. In this case "Medium" is what should be written in the Risk.

RISK MATRIX

CONSEQUENCES	LIKELIHOOD (or probability)			
	High	**Medium**	**Low**	**Remote**
Severe	**High**	**High**	**Medium**	**Low**
Moderate	**High**	**Medium**	**Medium/Low**	**Effectively Zero**
Insignificant	**Medium/Low**	**Low**	**Low**	**Effectively Zero**
Negligible	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**

Overall statement of risk

- Carefully consider the risks associated with your project, the nature of the activity with which you will be engaged, and its location.
- Check the information from Health and Safety pages in the intranet including those specifically for the 4YP.

Students must discuss these risks with their supervisor.

☑ **Office work only**. My project involves only basic office work (paper and computers). It does not involve hands-on laboratory or field work of any kind. I am aware of the associated risks, including the health risks associated with the extended use of computers and display screens. No further assessment is required.

☐ **Low Risk**. I consider the health and safety risks associated with my project to be low, working in alignment with existing risk assessments, I have referenced relevant risk assessments above and have agreed with my supervisor that no further assessment is required. For example, collecting data from existing systems within a lab.

☐ **Medium Risk**. I consider there to be additional risks associated with my project as it requires risk assessment authorisation below:

Risk Assessments for Hazardous Substances & Biological Materials. The Biological & Chemical Safety Officer's (BSCO) signature is required for the final sign-off on Engineering Science COSHH Assessments. If the BCSO is unavailable the DSO can provide this signature. For IBME, the IBME Safety Officer can provide this signature. Reference E refers. The BCSO's signature is also required for risk assessments involving the use of biological materials.

Genetically Modified Organisms. Risk assessments involving genetically modified organisms require the BSCO's signature as well as approval from the Genetic Modification Safety Committee for the work to proceed. The department's Safety Policy refers.

Laser Risk Assessments: In addition to the supervisor of the laser equipment/experiment concerned, the Department Laser Safety Officer (DLSO) must also sign risk assessments involving lasers.

Where Specialist Safety Officers Originate Risk Assessments. Where the DSO or Specialist Safety Officers write, co-write or otherwise originate risk assessments they will be required to sign and authorize such risk assessments.

Requirements for review by specialists should be identfied within Safety Requirements section on https://fouryp.eng.ox.ac.uk/resourcetimepreview2.php

☐ **High Risk**. This is a high risk activity as identified by Specialist Safety Officers.

Please review with Specialist Safety Officers where projects are Medium Risk sign below, ask your supervisor to countersign and then submit to Sharepoint site.

Signature of student:	**Signature of supervisor:**
Date: 03/11/2022	**Date:03/11/2022**

Contents

Chapter 1

Introduction

1.1 Motivation

We are surrounded by distributed systems that make up a significant proportion of the infrastructure we have come to rely on. From social media to electrical power networks, it is essential we understand and improve their inner workings to allow them to cope with the ever-increasing demands of the modern world. Distributed systems provide great advantages over centralized ones. One advantage is scalability, by distributing a workload we are able to handle more data, more users, and more complex tasks. Distributed systems are also more resilient than centralized ones, they can continue to function when exposed to failures. They also provide a boost to performance as they can process data faster and more efficiently through parallelization. However, as the size of these systems grow, they are, often bottlenecked by their communication bandwidth. To take full advantage of the scalability of distributed systems and reduce the cost associated with communication we must develop communication efficient algorithms that maximize the rate of learning while minimizing the number and size of the messages that are exchanged.

The field of reinforcement learning (RL), recently highlighted by some very high profile algorithms such as AlphaGo [26] and OpenAI Five [3], aims to find policies that maximize the reward achieved in an environment. The problem setting it provides is incredibly versatile, with applications including robotics [1], games [29], control systems [33], natural language processing [20], and computer vision [22].

Inspired by this, distributed reinforcement learning (DRL) maintains the framework but dis-

tributes the learning process with the goal of speeding up training and allowing for the use of larger more complex models. However, as with all distributed systems, we may face a communication bottleneck leading to the desire to reduce how often agents communicate and how large the messages are when they do. As well as this, in many scenarios in which the data itself is distributed across devices and sharing this data is prohibited, for example for privacy or legal reasons, we must find a way to train a model while adhering to these constraints. This is the problem addressed in federated reinforcement learning (FRL).

In distributed systems the bandwidth available for communication between nodes varies widely. It is therefore desirable to develop algorithms where we can trade communication for performance where necessary to suit the limitations imposed by the system.

1.2 Current approaches

To achieve good communication efficiency there exists a number of different avenues for exploration, highlighted by recent work in this field. These include, thresholded communication strategies, gradient compression and black-box optimization. In thresholded communication we only communicate information which is sufficiently interesting or will provide a sufficient update to the system. Inspired by event triggered control (ETC) EBCDQL [19] is a Q-learning based method where agents only communicate samples where the difference between the estimated value of a state-action pair and the actual observed value, known as the temporal difference (TD) error, exceeds the threshold defined by a temporally discounted sum of past TD errors. This means only samples which provide a significant update to the Q value will be used resulting in good communication efficiency. However, due to the use of a tabular Q function this solution is not feasible for problems with large or continuous state or action spaces. Similar to this DAVIA [10] uses a linearly approximated value function where the approximate gradient of the objective function with respect to the weights is only communicated each episode if the approximate update to the objective function exceeds a threshold which increases over time. LAPG [7] builds on distributed policy gradient methods, reducing the amount of communication without reducing the quality of learning. It does this by only communicating approximate policy gradients if the squared norm of the increase in the difference between the last two samples of the policy gradient, known as innovation, from the last uploaded

policy gradient exceeds a threshold known as the LAPG condition.

Gradient compression increases communication efficiency by reducing the size of the messages sent by the system. This is demonstrated in [17], the TD-EF algorithm is proposed in which the system communicates a compressed version of the TD error and uses error feedback from the true TD error to update future communications. Under technical assumptions the authors give guarantees of convergence to the optimal policy.

Black-box optimization techniques benefit from a great advantage in terms of communication efficiency as they only need to communicate a single scalar after each episode. This is because they consider the system as a pure input output process where the input is the policy parameters and the output is the discounted cumulative reward. Using this they optimize directly in policy space. The Evolution Strategies (ES) [25] algorithm has show very strong results on a range of challenging reinforcement learning benchmarks such as Atari [2] and MuJoCo [28]. It approximates a policy gradient by taking samples of the total reward achieved in trajectories with parameters perturbed around the current parameters. This is highly parallelizable due to the minimal communication bandwidth used and lack of computationally expensive calculations such as calculating gradients using backpropagation. It is also very versatile, being effective on both discrete and continuous problems, however, in environments where all agents often receive the same reward it can perform poorly with little exploration. To address this NS-ES [8] optimizes the novelty of the behaviours the agent exhibits, thus the system learns behaviours that are not like any seen in past iterations. Combining this with the goal of maximizing reward NSR-ES weights the importance of optimizing reward and optimizing novelty leading to an agent which is less likely to get stuck in local optima. Finally, NSRA-ES uses an adaptive scheme to weight novelty and reward meaning the agent achieves very impressive rewards while also avoiding local minima. ES exhibits poor data efficiency as it discards large batches of reward samples immediately after use. In response to this IW-ES [6] reuses old reward samples through importance weighting. This leads to an acceleration of learning, however, with large learning rates the algorithm can become unstable.

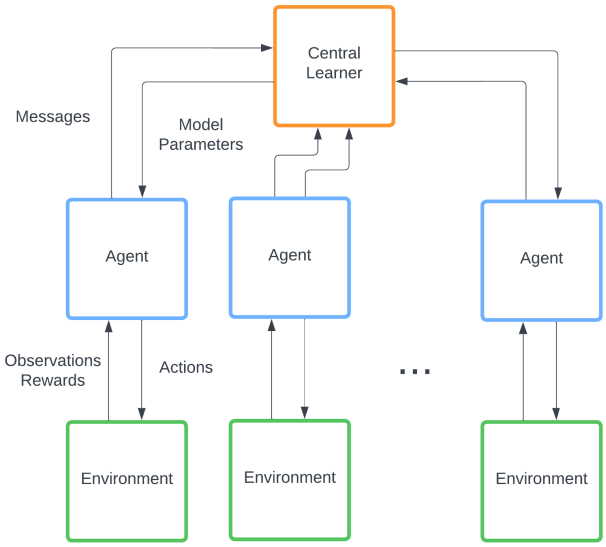


Figure 1.1: The distributed reinforcement architecture used for this project. Agents interact with a single environment taking actions and receiving an observation of the next state and a reward back. Agents also interact with the central learner sending messages and receiving updated model parameters.

1.3 Objectives of the project

The problem this project considers is that of a number of distributed agents acting in parallel instances of the same environment where they can communicate with a central learner to jointly solve a reinforcement learning problem. This architecture is shown in figure 1.1. There are two main objectives of this project, the first is to explore methods for communication efficient distributed reinforcement learning and compare them in simulation analysing their respective communication efficiency. The second is to develop a new algorithm for making distributed learning more efficient. To evaluate how well a specific algorithm performs in section 3.2 I define metrics based on the quality of learning and the amount of communication exhibited during training.

1.4 Contributions

I use the metrics defined in section 3.2 to compare algorithms from section 1.2 in a longitudinal study spanning multiple environments. I also perform a qualitative analysis of each of the algorithms highlighting their particular strengths and discussing what can be done to address their

weaknesses. As well as this analysis, in section 4.2, I propose a new general adjustable probabilistic communications scheme for ES, one of the most promising algorithms, and discuss the reasoning behind it. I then demonstrate in section 4.4 how a number of assumptions can be made to make the scheme implementable. Using this assumption, I show that if it were correct the expected number of agents communicating each episode is m out of n. Finally, I conduct experiments to compare this scheme with the original communication scheme and analyse the effect of varying its parameters.

I show that the evolution strategies algorithm with probabilistic communication (ESPC) has strong performance compared to ES, often achieving higher rewards while only communicating a fraction of the amount. I theorize that this is due to the bias it exhibits towards higher rewards causing ESPC to explore promising areas of parameter space more quickly than ES. As well as this I explore how varying ESPC's adjustable communication parameter affects its performance. I find that ESPC performs better with more communication and investigate unexpected communication behaviours at the beginning of training. Finally, I analyse how parameterizing the distribution I use to choose which samples are communicated affects the performance of ES. Specifically in the case of using a rolling mean and variance for this purpose I show that the horizon used for this has little effect on learning performance but causes closer tracking of reward by communication as the horizon length increases.

Chapter 2

Problem Background and Existing Algorithms

2.1 Reinforcement learning

Reinforcement learning (RL) is a subfield of machine learning that focuses on designing algorithms and models that allow an agent to learn through interacting with an environment. Agents are rewarded for achieving specific goals and punished for undesirable behaviours. RL has its roots in the field of psychology, where researchers studied the behaviour of animals and humans as they learned through positive and negative feedback. Due to its versatility RL is used in a number of fields such control theory, game theory and multiagent systems. It has been successfully applied to a wide range of problems, including game playing, robotics, and natural language processing. However, it remains a challenging and active area of research, particularly in complex and high-dimensional environments as well as environments in which there are multiple agents.

In RL, an agent learns by receiving feedback in the form of rewards or punishments for each action taken in the environment. The agent interacts with the environment through a series of discrete time steps. At each time step, the agent observes the current state of the environment and chooses an action based on its current policy. The environment then transitions to a new state and provides the agent with a reward based on the chosen action and the new state. The agent updates its policy based on the reward signal, with the goal of maximizing expected cumulative discounted

reward over time.

Formally, the problem of RL can be modelled as a Markov decision process (MDP), which can be described by the 5-tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is a set of action-dependent Markov transition kernels, $R : \mathcal{S} \to \mathbb{R}$ is a reward function and $\gamma \in [0, 1]$ is the discount factor. The actions an agent takes are determined by its policy $\pi : \mathcal{S} \to \mathcal{A}$. The policy can be alternatively defined as a distribution over actions $a \sim \pi(s)$. This can be advantageous as a non-deterministic policy can lead to greater exploration and can more easily avoid local optima.

We consider the episodic formulation of RL where the agent acts in the environment until it reaches some terminal state whereupon the episode ends. In order to ensure episodes are of finite length we require a non-empty set of states $\Omega \subseteq \mathcal{S}$ such that a state $s \in \Omega$ is reached, a final reward is issued, and the episode terminates. We also require that every policy π has a non-zero probability of reaching Ω at some point in the trajectory starting from every state.

The value of a particular policy in a particular state is defined as the expected cumulative discounted reward the agent will receive starting in that state

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s] \\ &= R(s_0) + \gamma \mathbb{E}[V^\pi(s_1)], \end{aligned}$$

where s_t is the state at time t starting from $s_0 = s$ and following policy π. The optimal value function is defined as that with the highest expected cumulative discounted reward

$$V^*(s) = \max_\pi V^\pi(s),$$

which yields the optimal policy $\pi^*(s) = \arg\max_\pi V^\pi(s)$. We can find the optimal value function using value iteration by repeating the update

$$V_t(s) \leftarrow \max_{a \in \mathcal{A}} R(s) + \mathbb{E} V_{t-1}(s'),$$

for all states $s \in \mathcal{S}$ until convergence is achieved. Value iteration is guaranteed to converge when the state and action spaces are finite, and reward values have a finite upper and lower bound.

It is common to come across environments where the state space is too large to use a tabular value functions as above. It will take too many iterations to have visited all the states to be able to have an accurate estimate of their value. Therefore, it is necessary to use a function approximator so that all states can be represented. A common method for this is linear value approximation

$$\hat{V}_{\boldsymbol{\theta}}(s) = \sum_{k=1}^{K} \theta_k \phi_k(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s),$$

where $\boldsymbol{\theta}$ is the parameter vector and $\boldsymbol{\phi}(s)$ is a feature vector for state s. It is also common to use a neural network for this purpose. To update a value function of this form towards the true value function for policy π we can use data to estimate the required update to the parameters $\boldsymbol{\theta}$. For each tuple (s, r, s') sampled following policy π we can calculate the approximate gradient as

$$\Delta\boldsymbol{\theta} = (r + \gamma \hat{V}_{\boldsymbol{\theta}}(s') - \hat{V}_{\boldsymbol{\theta}}(s)) \nabla_{\boldsymbol{\theta}} \hat{V}_{\boldsymbol{\theta}}(s)$$

repeating this for a whole trajectory. For linear value approximation, in the limit this is guaranteed to converge to a point close to the true value function with the least squares error $\varepsilon_{\hat{\boldsymbol{\theta}}} = \sum_{s \in \mathcal{S}} \mu(s)(V^\pi(s) - \hat{V}_{\hat{\boldsymbol{\theta}}}(s))^2$ bounded by $\varepsilon_{\hat{\boldsymbol{\theta}}} \leq \frac{1}{1-\gamma} \min_{\boldsymbol{\theta}} \varepsilon_{\boldsymbol{\theta}}$, where $\mu(s)$ is a state distribution such that $\sum_{s \in \mathcal{S}} \mu(s) = 1$.

To estimate the expected value of the state we need the transition probabilities between all states $p(s'|s, a),\ \forall s, s' \in \mathcal{S},\ \forall a \in \mathcal{A}$. In practice, we often do not have an accurate model of the environment, so these are not known. We can instead use an action value function defined as

$$Q_\pi(s, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s, a_1 = a].$$

With this we can use Q-learning to approximate the optimal action value function with no direct model of the environment and from the action value function we can approximate the optimal policy. The Q-learning iteration given sample (s, a, r, s') is

$$Q^{new}(s, a) \leftarrow Q^{old}(s, a) + \alpha_t (R(s) + \max_{a'} Q(s', a')),$$

where α is the learning rate. This is guaranteed to converge under the assumptions that the sum of

the learning rates α_t diverge, the sum of their squares converge and each state-action pair is visited infinitely often [31]. Similarly to using the state value function we can use function approximators to expand size of problem for which this method is feasible. For the linear case this converges with the same bound as before.

Alternatively to optimizing in value space we can optimize in policy space directly. One method for this is policy iteration where on each step we calculate the value function for the current policy then update the policy with

$$\pi^{new}(s) \leftarrow \arg\min_{a \in \mathcal{A}} R(s) + \gamma V^{\pi^{old}}(s').$$

Similarly to with value functions we can approximate policies to simplify optimization. We can say $\pi(s) = \pi_\theta(s)$ where θ is a vector of parameters. This approximated policy is often a neural network meaning θ represents the weights and biases in the network. We define the objective function as the discounted expected cumulative reward from the initail state s

$$J^\theta(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | s_0 = s] = V^{\pi_\theta}(s)$$

Since the policy is now parameterized we can take its gradient with respect to some objective function and perform gradient ascent to optimize it. A well-known algorithm for this is REINFORCE [32] shown in algorithm 1.

Algorithm 1 REINFORCE algorithm

Initialize θ arbitrarily
for each episode $\{s_0, a_1, r_1, \ldots, s_{T-1}, a_T, r_T\} \sim \pi_\theta$ **do**
  for $t = 0, \ldots, T-1$ **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_{t+1}) v_t$ ▷ $v_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$
  end for
end for

2.2 Federated learning

Federated learning (FL) [16] is a machine learning technique where training is performed in a decentralized manner, often on physically separate devices such as mobile phones, IoT devices or

other edge devices. The critical issue federated learning attempts to address is that of training a robust model while preserving data privacy, data security, and data access rights. In FL we aim leverage the computing power and data generated by these distributed devices to build a shared model, without sharing raw data between agents.

Federated learning can be broadly divided into four main phases, initialization, training, aggregation and testing. First during initialization phase the central server initializes a machine learning model, for example a deep neural network, and shares it with a set of devices. Next in the training phase, each device trains the model on its local training set, then evaluates the performance of the local model on the local evaluation set. The updated model parameters are sent back to the central server. The central server then aggregates these parameters and updates the shared model accordingly. The updated model is sent back to the devices and this training and aggregation process is repeated until the desired level of accuracy is achieved on the local evaluation sets. After this, in the testing phase the model is tested in a distributed manner on the local test sets. The performance of the model can then be aggregated to gain an overall performance metric for the model.

Federated learning has several advantages over traditional centralized machine learning approaches, including reduced data transfer costs, improved privacy and security, and the ability to learn from distributed and heterogeneous data sources. However, it also presents some challenges, such as communication and synchronization overhead, heterogeneity of devices, and model optimization across different data sources.

2.3 Distributed reinforcement learning

Distributed reinforcement learning (DRL) involves training an RL agent in a distributed manner across multiple machines, which may be physically separated, that work together to learn a single policy. DRL provides an advantage in large, complex environments or problems where single-agent RL might be too slow. There exists two main paradigms of DRL: centralized training and decentralized execution (CTDE) and decentralized training and decentralized execution (DTDE) [11].

In CTDE, a central learner is trained on the experience communicated from the agents. All

agents have identical state and action spaces and collect trajectories through acting in an environment. The central learner then updates a shared policy based on the communications it receives from agents.

In DTDE, each agent learns independently by acting in the environment. They maintain their own policy, updating it based on observations and rewards received from the environment. The agents communicate with each other directly to coordinate actions and improve performance. It is often more scalable and robust than CTDE, however, it can be less efficient in certain scenarios.

DRL is faced with issues when considering the communication and synchronization between agents. Centralized communication frameworks are often used, where a central server manages the communication and synchronization between the agents. An alternative to this is to use a decentralized communication framework, where agents communicate directly.

As well as this, DRL can take advantage of a number of existing RL algorithms, such as Q-learning, policy gradient methods, and actor-critic methods. These algorithms can be adapted for DRL by using mechanisms to handle distributed learning.

Federated reinforcement learning (FRL) [23] is a subfield of DRL which adopts the philosophies underpinning federated learning. Local devices each act as a reinforcement learning agent recoding observations and taking actions in its local environment. Agents can share data such as policy or value gradients to update the shared model allowing for collaborative learning without sharing raw data, in this case taking the form of agent trajectories, between agents.

2.4 Exisiting algorithms

2.4.1 Distributed Q learning

Distributed Q learning (DQL) adapts Q-learing to a distributed setting, increasing exploration and leading to an increase of the rate of convergence to the optimal policy. The agents communicate the trajectory collected after every episode. The central learner determines the Q update by iterating through the agents and at each time step $t = 1, \ldots, n$ taking sample $u = (s_{t-1}, a_t, r_t, s_t)$ and calculating $\Delta(u) = R(s_t) + \gamma \max_a Q(s_t, a) - Q(s_{t-1}, a_t)$ which is the TD error. Let the subsets

$\mathcal{U}_s = \{u : s_{t-1} = s\}$. The Q function is then updated by

$$Q^{updated}(s,a) = Q^{old}(s,a) + \alpha \frac{1}{|\mathcal{U}_s|} \sum_{u \in \mathcal{U}_s} \Delta(u).$$

This method provides good convergence behaviour on simple environments, although it is not guaranteed to converge. It reasonably assigns credit for a reward to a given action even if the reward is delayed. However, due to the use of a tabular value function this method can perform poorly on environments with many state-actions as even after many iterations it will come across states it has seldom visited and thus does not have an accurate estimate of their value. As well as this the build up of a large table of Q values can fill the memory of the computer the algorithm is running on leading to a reduction in performance and eventually can cause it to crash. Also, since the whole trajectory is communicated at each step the size of the messages sent can become very large for environments with long episode lengths or large state representations such as pixels on a screen meaning communication efficiency is poor.

Deep Q-learning is often used to address the performance issues where a neural network is used to approximate the Q function by estimating the value for each action in each state. This allows for use in larger, more complex environments. Data from a trajectory is used to approximate the gradient of the TD error with respect to the model parameters, allowing the use of gradient descent to minimize the error of the prediction. The model is often split into a prediction network, which approximates the value of the current state and a target network, which approximates the target from which we calculate the TD error.

2.4.2 Event based communication DQL

Event based communication distributed Q Learning (EBCDQL) [19] uses a communication scheme based on Event Triggered Control (ETC) [27] techniques to reduce the communication of DQL while maintaining good learning performance. The algorithm works similarly to vanilla DQL, however, agents only communicate the samples that have a TD error above the threshold defined as

$$|\hat{\Delta}(u_i)| \geq \max(\rho L_i(t), \epsilon)$$

where $L_i(t) = (1-\beta)L_i(t) + \beta|\hat{\Delta}(u_i)|$ and $L_i(0) = 0$.

This algorithm exhibits the same benefits and drawbacks as DQL, however, it has much better communication efficiency as it only sends samples to the central learner when they provide sufficient updates to the Q function.

2.4.3 Distributed approximate value iteration algorithm

In the distributed approximate value iteration algorithm (DAVIA) [10], agents communicate when a gradient step update provides a sufficient increase in the objective function, the threshold for which decreases over time. The objective function in this case is defined as the expected squared error between the updated value function and the current value function.

$$J(\theta) = \mathbb{E}[V^{updated}(s) - \sum_{i=1}^{n} \theta_i \phi_i(s)]^2$$

The algorithm uses a linearly approximated value function with an ϵ-greedy policy, meaning it takes a random action with probability ϵ and the estimated best action otherwise. The approximate gradient is calculated by

$$\hat{\nabla} J(\theta) = \frac{1}{T} \sum_{t=0}^{T} \phi(s_t)(\theta^T \phi(s_t) - r_t - \gamma V(s_{t+1}))$$

and the condition for communication is represented by

$$J(\theta_k - \epsilon \hat{\nabla} J(\theta_k)) \leq \frac{\lambda}{\rho^{N-1-k}}.$$

In the central learner the gradient is updated by the mean of the transmitted gradients. This algorithm is guaranteed to converge under technical assumptions and shows good communication efficiency as the gradients are only communicated when they provide a sufficient update and are of constant size. The approximated value function also means it is able to handle more complex problems than DQL with larger state spaces. However, this does require pre-defined feature vectors for each state. Extension of this method to non-linear approximations of the value function would create the opportunity apply the communication efficiency of this algorithm to more complex problems.

2.4.4 Evolution Strategies

Evolution Strategies (ES) [25] is a black-box optimization technique which approximates the policy gradient, or more specifically, the gradient of the expected cumulative discounted reward with respect to the parameters of a parameterized policy. The agents each run an episode with parameters, perturbed normally about the current parameters θ, before communicating the scalar cumulative discounted reward back to the central learner. We can use the fact that

$$\nabla_\theta \mathbb{E}_{\epsilon\sim N(0,I)}[F(\theta+\sigma\epsilon)] = \frac{1}{\sigma}\mathbb{E}_{\epsilon\sim N(0,I)}[F(\theta+\sigma\epsilon)\epsilon]$$

where $F(.)$ is the cumulative discounted reward for a given value of the parameters, and σ, chosen as a hyperparameter, is the standard deviation of the normally distributed perturbations ϵ. We calculate the approximate gradient by

$$\hat{\nabla}_\theta = \frac{1}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i. \tag{2.1}$$

It is useful to note that since $\mathbb{E}_{\epsilon\sim N(0,I)}[F(\theta)\epsilon/\sigma] = 0$ we get

$$\mathbb{E}_{\epsilon\sim N(0,I)}[F(\theta+\sigma\epsilon)\epsilon/\sigma] = \mathbb{E}_{\epsilon\sim N(0,I)}[(F(\theta+\sigma\epsilon) - F(\theta))\epsilon/\sigma]$$

from which we can see that ES is computing the finite difference estimate of the gradient in a random direction. This suggests that this method will scale poorly with the intrinsic dimension of optimization problem. However, in experiments this effect is not observed when considering the number parameters as a substitute for the number of intrinsic parameters of the problem, in fact larger neural networks tend to perform better. The authors hypothesize this is because larger networks have fewer local minima [14].

The main advantage of ES is its suitability for scalable parallelization. This is due to the minimal communication between agents and central learner as well as the absence of any backpropagation calculation. This means that ES can provide an order of magnitude speed up in training time. It also benefits from the fact that it only uses a parameterized policy so is suitable for any size of problem including those with continuous state and action spaces, however, by discretizing actions

ES exhibits better exploration performance on some environments.

ES performs excellently in terms of the size of the messages sent, communicating only a single scalar, the cumulative discounted reward for the episode, per agent. However, since this communication occurs after every episode, the performance in terms of number of messages sent is poor. It also exhibits poor data efficiency, only updating its parameters once from large batches of trajectories.

Chapter 3

Comparative Analysis

3.1 Problem setup

I consider a CTDE DRL problem with N agents acting in parallel instances of the same environment. The agents are able to episodically communicate with a central learner in order to jointly solve a Reinforcement Learning problem. They collect time series data consisting of state-action trajectories and rewards.

Consider the episodic MDP $\mathcal{M} = (\mathcal{S}^N, \mathcal{A}^N, \mathcal{P}, R, \gamma)$ where the terms are defined as in section 2.1. A trajectory following policy π through the environment is denoted as a sequence $\zeta = (s_0, a_1, r_1, s_1, a_2, \dots)$. At the end of episode k in trial l if some condition as a function of the current trajectory is satisfied i.e., $c_i^{k,l} = 1$ where sample $c_i^{k,l} \leftarrow \mathcal{C}(\zeta_i^{k,l})$, $c_i^{k,l} \in \{0, 1\}$ and $\mathcal{C}$ is some distribution representing a communication condition parameterized by the latest trajectory (this could be deterministic), then the agent i communicates information $z_i^{k,l} = Z(\zeta_i^{1,l}, \zeta_i^{2,l}, \dots, \zeta_i^{k,l})$, derived from the trajectories it has experienced, to the central learner such as select (s_{t-1}, a_t, r_t, s_t) tuples or value gradients. The central learner then updates the policy based on the information received and communicates this back to the agents. The event loop can be seen in algorithm 2.

3.2 Metrics

To evaluate the quality of DRL algorithms I establish metrics which can be used to compare them directly. These metrics broadly cover how well the systems learn and how much they communicate.

Algorithm 2 Distributed RL Event Loop

```
Initialize L                                   ▷ The number of trials to be run
Initialize N                                   ▷ The number of episodes to be run
for l = 1, ..., L do
    Initialize π
    for k = 1, ..., N do
        central learner communicates π to all agents
        for each agent i = 1, ... n do
            run episode to collect sample trajectory ζ_i^{k,l}
            compute z_i^{k,l} = Z(ζ_i^{1,l}, ζ_i^{2,l}, ..., ζ_i^{k,l})
            communicate z_i^{k,l} if c_i^{k,l} = 1
        end for
        update π from received information
    end for
end for
```

In terms of communication I exclusively focus on messages sent from the agents to the central learner as in this problem the communication from central learner to agents is fixed. To evaluate how well an agent learns we can average the discounted episodic reward achieved each episode across agents and trials then plot the average discounted episodic reward (μ^{reward}) against episode number. The μ^{reward} as a function of episode number is

$$\mu^{\text{reward}}(k) = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{n}\sum_{i=1}^{n}\sum_{t}\gamma^t r_i^{k,l}(t)$$

where $r_i^{k,l}(t)$ is the reward in episode k and trial l for agent i at time t or equivalent the reward in trajectory $\zeta_i^{k,l}$ at time t. We can say algorithm a learns better than algorithm b after N training episodes if $\mu_a^{\text{reward}}(N) > \mu_b^{\text{reward}}(N)$.

As well as maximizing reward we would like to minimize communication. For this we define two metrics. The first is the number of messages a system has sent from agents to the central learner averaged over the number of agents ($\mu^{\text{frequency}}$). The $\mu^{\text{frequency}}$ at episode k is

$$\mu^{\text{frequency}}(k) = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{n}\sum_{i=1}^{n} c_i^{k,l}$$

We say an algorithm a communicates less frequently than algorithm b on episode N if $\mu_a^{\text{frequency}}(N) < \mu_b^{\text{frequency}}(N)$. The second is size in bytes of messages sent from agents to the central learner aver-

aged over the number of agents (μ^{size}). The μ^{size} at episode k is

$$\mu^{\text{size}}(k) = \frac{1}{L}\sum_{l=1}^{L}\frac{1}{n}\sum_{i=1}^{n}\text{bytes}(z_i^{k,l})$$

We say algorithm a communicates less intensely than algorithm b on episode N if $\mu_a^{\text{size}}(N) < \mu_b^{\text{size}}(N)$.

3.3 Evaluation of existing algorithms

3.3.1 Computational setup

To conduct experiments it was necessary to implement each of the algorithms and to run them on various environments. To achieve this I built a framework in Python, specifying base classes for algorithms, environments and each of their respective components. The structure of the framework is shown in figure 3.1 Each of the algorithms were implemented according to the template laid out by the framework allowing for any algorithm to be easily paired with any environment. ES uses a neural network policy which was implemented in PyTorch [21] to enable acceleration of computation. Other algorithms mainly used NumPy [12] and SciPy [30] for their calculations. To pair algorithms with environments I implemented a Universe class which given a number of algorithms, environments, and parameters, appropriately pairs and evaluates them while recording details of the training via its logger. This means it was possible to run experiments in a large array of configurations using just a few lines of code in a Jupyter notebook. Once the experiments were complete the results were saved in a standardized data structure, shown in figure 3.2 from which I could analyse the performance of the algorithms. I used a number of Gymnasium (formerly OpenAI Gym [5]) environments to train on as it has many standard benchmarks for RL algorithms pre-implemented. To make them compatible with my framework I wrote a wrapper that mapped the inputs and outputs to an appropriate form. As a separate module I built a data analysis tool to extract the desired data from the results of an experiment and plot the results. This often involved extracting and combining data from multiple different files.

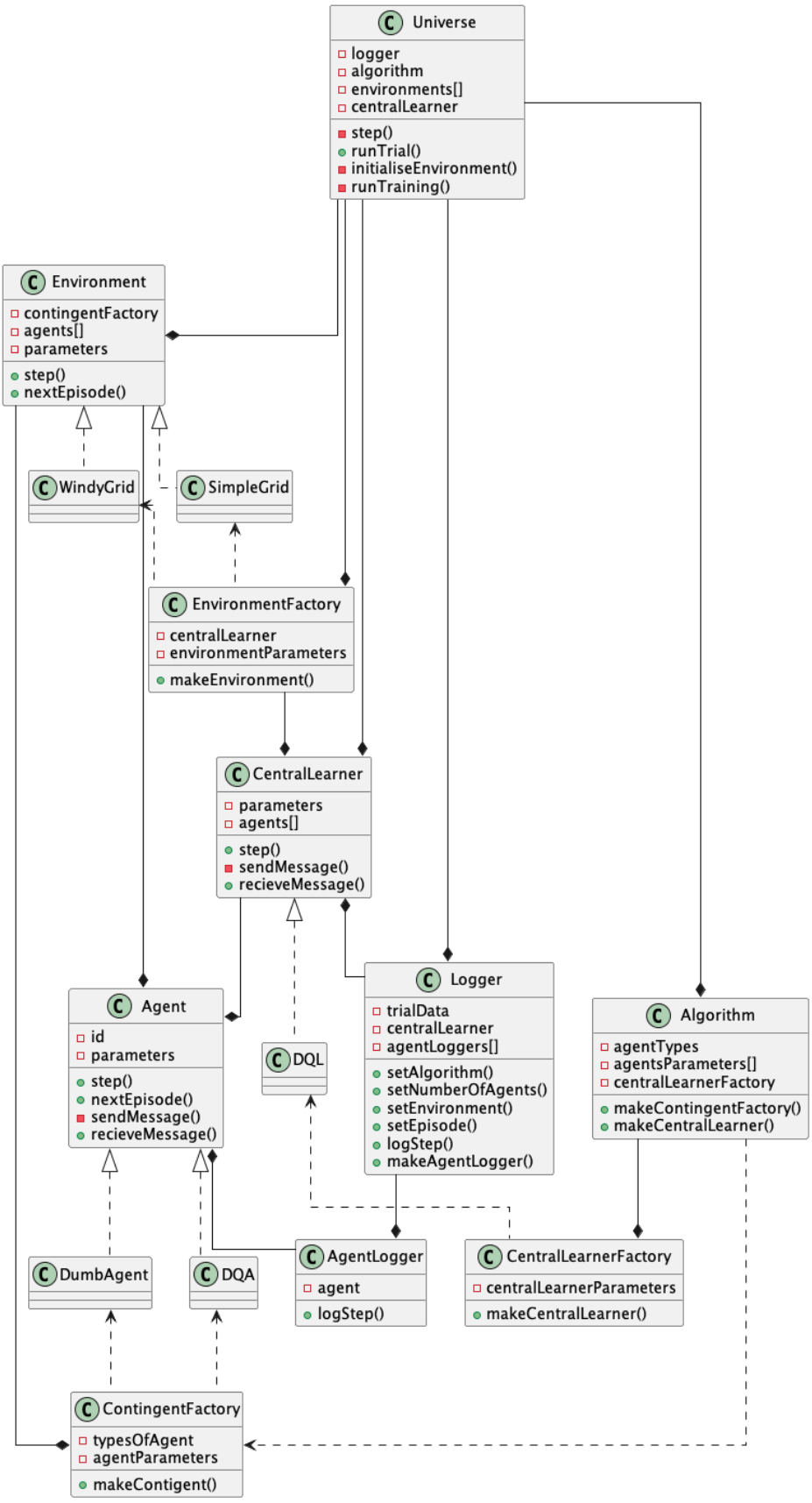


Figure 3.1: The structure of the framework used to run the experiments

```JSON
{
    "trialID": "id",
    "algorithms": {
        "algorithmName": {
            "algorithmParameters": {"hyperparameter object"},
            "numbersOfAgents": {
                "environmentName": {
                    "environmentParameters": {"Hyperparameter object"},
                    "episodeNumber": {
                        "stepNumber": {
                            "centralLearner": {
                                "?message": {},
                                "?valueFunction": {}
                            },
                            "agentId": {
                                "lastState": "State",
                                "lastAction": "Action",
                                "reward": "Number",
                                "currentState": "State",
                                "currentState": "Action",
                                "?message": {}
                            },
                        }
                    }
                }
            }
        }
    }
}
```

Figure 3.2: The data structure used to store the results of experiments

3.3.2 Experiments

I evaluated the DQL, EBCDQL, DAVIA, and ES algorithms on the Simple Grid and Frozen Lake environments. They are both grid worlds where the goal is to move from a fixed starting state to a fixed terminal state and the possible actions are to move left, right, up, down, or stay in the same place. Simple Grid is a custom 5x5 grid world where the goal is to move from the top left state to the bottom right state, agents receive a reward of -1 for every step apart from at the terminal state where they receive a reward of 0. Frozen lake (figure 3.3) is a 5x5 Grid environment from Gymnasium in which the goal is to move from the upper left to the lower right square upon which the agent receives a reward of 1. A number of squares in the grid are holes in the lake, where if reached by the agent the episode ends. These environments were chosen as they were simple enough for all the algorithms to feasibly run on them while still providing a clear picture of each

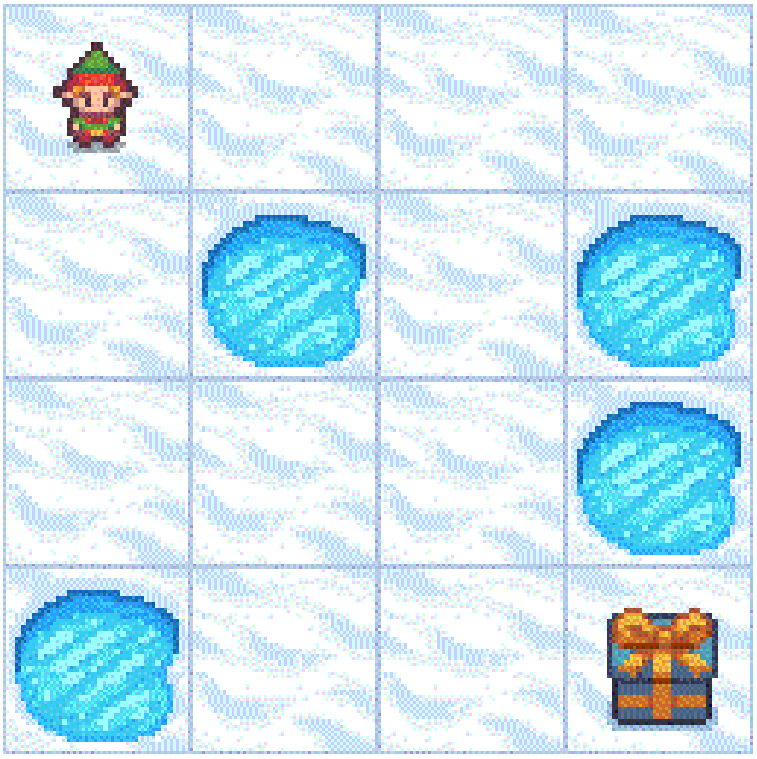


Figure 3.3: The Frozen Lake environment

of their performance.

Each experiment used $n = 5$ agents and a discount factor of $\gamma = 0.9$. Agents were trained over 1000 episodes and this was repeated across 100 trials. I analyse the average discounted episodic reward (μ^{reward}), the average number of messages ($\mu^{\text{frequency}}$), and the average size of messages (μ^{size}). I plot the mean as well as the 10^{th} to 90^{th} percentile across trials for each algorithm.

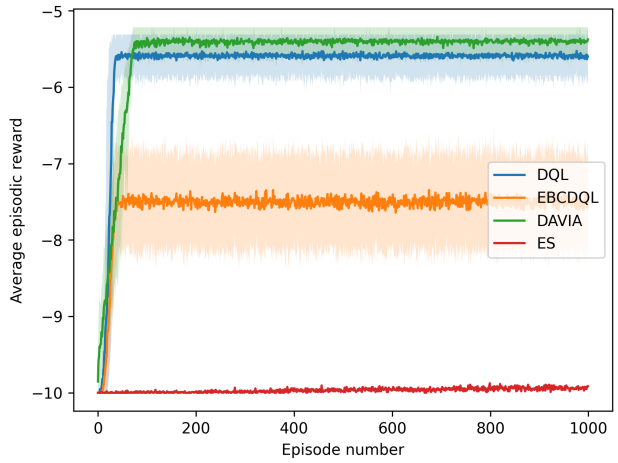
The results on the Simple Grid environment are shown in figure 3.4. It can be seen that DQL and DAVIA quickly converge to an optimal solution whereupon DAVIA rapidly reduces communication. EBCDQL reaches a suboptimal solution at which communication reduces significantly. This means the TD errors observed beyond this point are not large enough to justify communication even when improved performance is possible, resulting in a stagnation of learning. ES fails to learn an effective strategy for reaching the terminal, achieving near the minimum reward on every episode. This is likely due to a lack of exploration and low sensitivity to rewards when they are achieved. In this case the discount factor of $\gamma = 0.9$ means that rewards towards the end of the trajectory have little impact resulting in a very small difference in rewards between agents, so exploration is only down to the randomly chosen perturbations and the approximation of the policy gradient is poor.

For ES and DQL the number of messages sent is, by design, one per agent per episode which in this setting is the worst achievable. For EBCDQL and DAVIA the number of messages sent decreases rapidly around episode 50 meaning fewer agents are receiving significant updates to their value functions each episode.

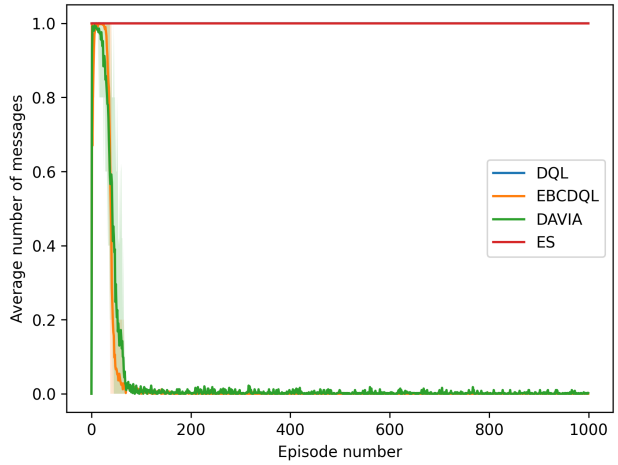
As DQL sends the whole trajectory after every episode, for long episode lengths the size of messages it sends is poor. This can be seen at the beginning of figure 3.4c. However, as the length of the episodes decrease, as faster routes to the terminal have been found, the size of the messages decreases dramatically. A similar behaviour is exhibited by EBCDQL, however, only significant subsections of the trajectory are sent, so the sizes of the messages are smaller. For DAVIA, since only the value gradient is communicated, the size of the messages sent are constant so the change in the average size of messages sent is only due to variations in the number of agents communicating. ES only communicates a single scalar per agent per episode, so the average size of messages is very small, but constant.

The results on the Frozen Lake environment are shown in figure 3.5. Figure 3.5a shows the cumulative episodic reward rather than the episodic reward as the results are particularly noisy meaning the cumulative plot more clearly shows the learning of the agents. By looking at the gradient we can determine how well the algorithms learned. EBCDQL performed best followed by ES, however, after around episode 200 their respective plots become linear meaning no learning occurs from this point on. DQL and DAVIA both failed to reach the terminal state a significant number of times.

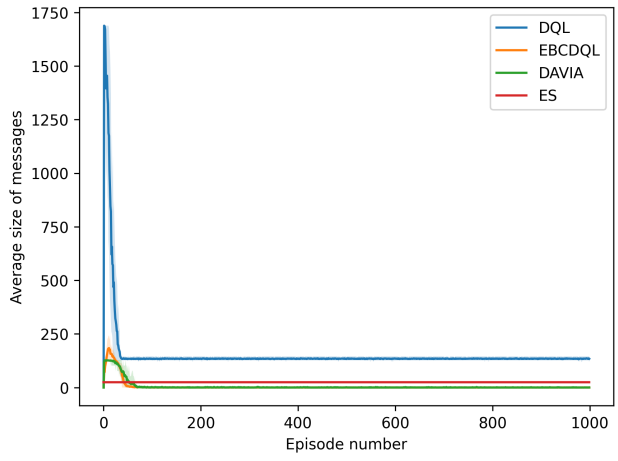
For EBCDQL and DAVIA in this environment communication was minimal. This is perhaps because achieving any reward takes a large amount of exploration and thus episodes that could provide a significant update to the value function are rare. The relationship between the number of messages and the size of the messages remained similar to that demonstrated in Simple Grid.



(a) The average episodic reward received by agents

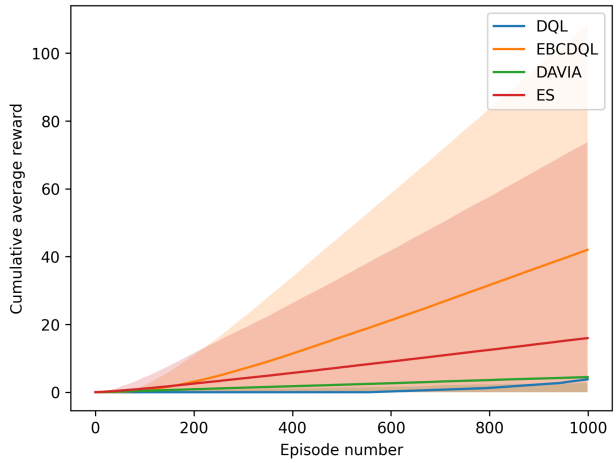


(b) The average number of messages sent from agents to central learner

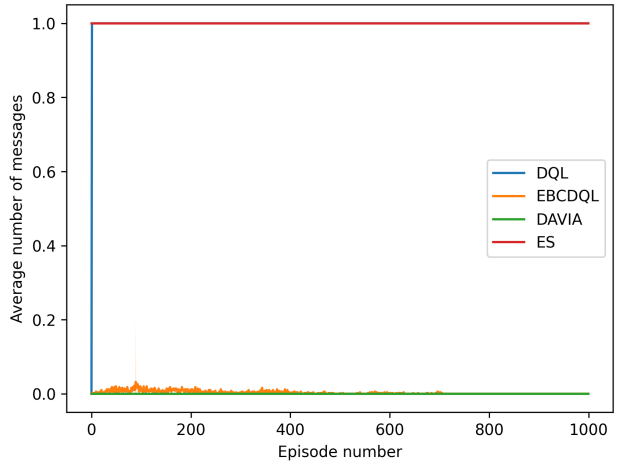


(c) The average size of messages sent from agents to central learner

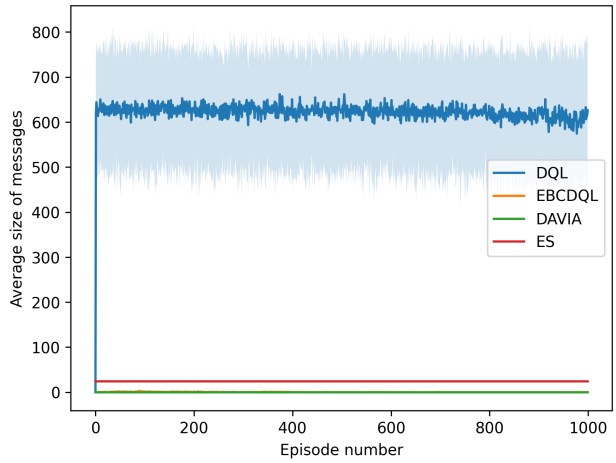
Figure 3.4: Results from the algorithm evaluation in the Simple Grid environment for DAVIA, DQL, EBCDQL, and ES with $n = 5$ over 1000 episodes and 100 trials. The shading represents the 10^{th} to the 90^{th} percentile across trials.



(a) The cumulative average reward received by agents



(b) The average number of messages sent from agents to central learner



(c) The average size of messages sent from agents to central learner

Figure 3.5: Results from the algorithm evaluation in the Frozen Lake environment for DAVIA, DQL, EBCDQL, and ES with $n = 5$ over 1000 episodes and 100 trials. The shading represents the 10^{th} to the 90^{th} percentile across trials.

3.4 Limitations

These experiments were heavily limited by the time it took to run them, this resulted in it being difficult to effectively tune the hyperparameters for each algorithm. Finer tuning could lead to a more accurate comparison of what these algorithms are capable of. In particular EBCDQL and DAVIA could be tuned to communicate more on Frozen Lake which could improve performance. In addition to this I only tested on two fairly similar environments. On other environments, performance could differ, therefore, using a larger set of environments with more variety between them could give a better picture of algorithm performance. Lastly the experiment consisted of 2 thresholded algorithms but only one black-box algorithm and no gradient compression algorithms. Using a wider variety of algorithms would provide a clearer picture of the relative performance of algorithms within each category and of the performance of each category relative to the others.

3.5 Summary

In summary, we have observed that

- EBCDQL and DAVIA are effective algorithms. They achieve mostly good rewards and their thresholded communication show a good trade off between performance and communication. They send messages only when it provides benefit to the learning process
- DQL learns well but has prohibitively poor communication properties. It sends a message every episode which is proportional to the length of the episode
- ES is not effective in these environments but has excellent performance in terms of the size of messages sent

ES performs poorly in these experiments proving to be brittle in simple environments. However, ES is considerably more flexible than any of the other algorithms tested here. Its effectiveness on environments infamous for their difficulty such as the MuJoCo humanoid shows the performance exhibited here does not well represent its strengths. The size of messages sent are very small, however, the frequency of the messages is poor. One way to improve this is to use a probabilistic communication scheme to reduce the number of messages sent. This is the focus of the subsequent chapter.

Chapter 4

Algorithm Proposal and Evaluation

4.1 Principle behind the new algorithm

When calculating the gradient using evolutionary strategies we first take n independent samples from a normal distribution by which we perturb the parameters θ of each agent $\epsilon_i \leftarrow \mathcal{N}(0, I),\ \forall i = 1, \ldots, n$. Then using these samples each agent runs an episode, sampling the cumulative discounted reward for each perturbation, $F_i \leftarrow \mathcal{F}(\theta + \sigma\epsilon_i),\ \forall i = 1, \ldots, n$ where $\mathcal{F}$ is some unknown distribution defined by the environment. Note that since the samples $(\epsilon_1, \ldots \epsilon_n)$ and independent and identically distributed (iid) and as sample F_i is conditional on ϵ_i then the samples $(F_1, \ldots, F_n)$ are also iid. These samples are then communicated from every agent to the central learner after every episode and the approximate policy gradient is calculated as in equation 2.1.

We are interested in finding a scheme that reduces the number of samples that are communicated after each episode. A possible approach to achieve this is for only m out of n agents to communicate or equivalently $n - m$ out of n agents to not communicate each episode on average. To do this we introduce the notion of the utility of a sample in calculating the final gradient, meaning how important is this particular sample relative to the others. Difficulty in determining this arises as a particular agent i only knows the values of its own sample pair (ϵ_i, F_i). However, since the samples are iid, if we assume some distribution over the utility of the samples, it is possible to estimate the probability that the sample we have drawn is greater than that of at least $n - m$ out of $n - 1$ other samples i.e., it is more useful in calculating the gradient than at least $n - m$ out of $n - 1$ other samples. We can then communicate the sample with this probability.

4.2 General Communication Scheme

To construct a communication scheme of this nature we must first specify some expected number of agents m that we wish to communicate each episode. We then define a deterministic utility function $U : \mathbb{R}^q \times S_F \to \mathbb{R}$ where q is the number of parameters and S_F is the sample space of the distribution from which the samples F_i are drawn. Since U is a function of random variables, it is itself a random variable with its own marginal distribution. However, since the distribution $\mathcal{F}$ is unknown then the distribution $\mathcal{U}$ of U is also unknown. We therefore assume some distribution of U based on information the agent knows. This includes all historical values for utility the agent has experienced. The procedure for determining whether a sample is communicated is then as follows

1. Draw sample $\epsilon \leftarrow \mathcal{N}(0, I)$
2. Draw sample $F \leftarrow \mathcal{F}(\theta + \sigma\epsilon)$
3. Calculate $u = U(\epsilon, F)$
4. Use assumed distribution $\mu \sim \mathcal{U}$ to calculate $p = \mathbb{P}(\mu < u)$
5. Calculate $p_{comm} = \mathbb{P}(M \geq n-m) = \sum_{k=n-m}^{n-1} \binom{n-1}{k} p^k (1-p)^{n-k-1}$, where $M \sim \text{Bi}(n-1, p)$
6. Generate a random number $\delta \leftarrow \text{Uniform}(0, 1)$
7. If $\delta < p_{comm}$ communicate F to the central learner

4.3 Determining the utility function

By discarding samples we will inevitably introduce some form of bias in the approximate policy gradient that we calculate. However, by careful choice of the utility function we can attempt to minimize the impact of this bias on the gradient ascent process. When performing gradient ascent introducing bias in the direction will be more impactful than if we introduce bias to the magnitude. It is much worse to go in completely the wrong direction than it is to step too far or not far enough in the right one. This is highlighted by the popularity of normalized gradient decent/ascent [18] in which the magnitude is discarded, entirely replaced by the manually assigned learning rate. To

Would like a better justification for this

calculate the gradient we use a sum of perturbation vectors weighted by the rewards they receive. We can alternatively think of this as a weighted sum of direction vectors.

$$\sum_{i=1}^{n} F_i \|\epsilon_i\|_2 \hat{\epsilon}_i = \sum_{i=1}^{n} w_i \hat{\epsilon}_i.$$

where $\hat{\epsilon_i} = \frac{\epsilon_i}{\|\epsilon_i\|_2}$ is a unit vector in the direction of ϵ_i. Since the directions $\hat{\epsilon}_i$ are uniformly distributed on a hypersphere [13] and each agent does not know what the others are, this direction $\hat{\epsilon}_i$ is not useful for an agent in determining how much a vector will contribute to the sum. We are therefore left with only useful factor in approximating the direction of the sum being the magnitude of individual samples, naturally leading to the utility function

$$U(F, \epsilon) = F\|\epsilon\|_2,$$

which can be easily calculated by the agent.

4.4 Approximating the utility function as a Gaussian random variable

To be able to implement the general communication scheme we must assume some distribution of utility. This will allow us to calculate the probability that an agent's sample has higher utility than another and thus how likely it is to be communicated. The distribution of utility $\mathcal{U}$ is dependent on both the distribution of perturbation $\mathcal{N}(0, I)$ and reward $\mathcal{F}(\theta + \sigma\epsilon)$. The distribution of reward is unknown, but we know is that it is conditional on the distribution of perturbation which is Gaussian. This gives us motivation to naively assume that utility is also distributed normally. To parameterize this normal distribution we can use a rolling mean and variance of the last k utility samples taken by an agent. Where k is a hyperparameter to be specified. Thus, we assume $\mathcal{U} = \mathcal{N}(\bar{u}_k, u_k^{\sigma^2})$ where $\bar{u}_k = \frac{1}{k}\sum_{t=T-k+1}^{T} u^t$, $u_k^{\sigma^2} = \frac{1}{k}\sum_{t=T-k+1}^{T}(u^t - \bar{u}_k)^2$ and T is the current time step. Alternatively to this we could take a weighted average of utilities weighting recent ones more highly. We refer to ES using this communication scheme as evolution strategies with probabilistic communication or ESPC.

We now show that given the assumption of Gaussian distributed utility, the expected probability

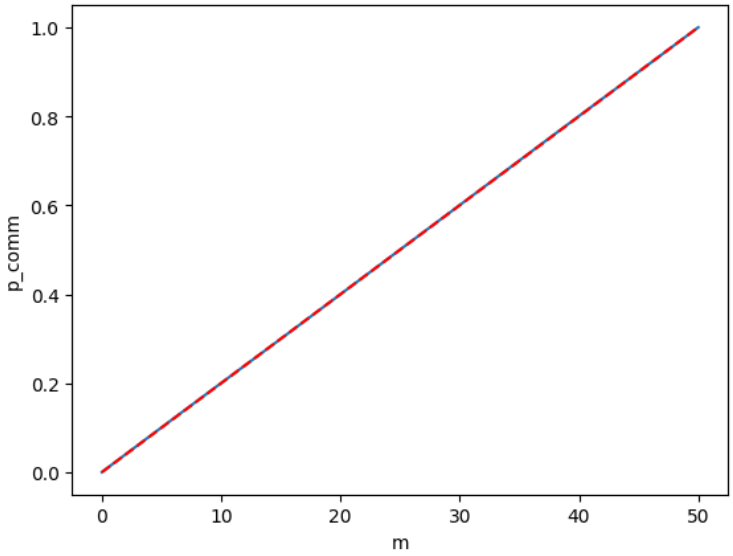


Figure 4.1: The expected probability of communication for different values of m with $n = 50$. The red dotted line shows the value of $\frac{m}{n}$ and the blue line shows the result of the Monte Carlo integration.

of communication is approximately $\frac{m}{n}$. The probability of communication p_{comm} is a direct function of the utility U. Therefore, the expectation of p_{comm} is

$$\mathbb{E}\left[p_{comm}\right] = \int_{-\infty}^{\infty} p_{comm}(z)\, dz$$

where

$$p_{comm} = \sum_{k=n-m}^{n-1} \binom{n-1}{k} p^k (1-p)^{n-k-1}$$

and

$$p = \int_0^z \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})\, dx$$

where $z = \frac{u-\bar{u}_k}{u_k^{\sigma^2}}$. We can calculate this expectation using Monte Carlo integration. We sample values of z from the standard normal distribution and calculate $p_{comm}(z)$ for each sample. Take the mean of these samples gives an approximation of the integral. Figure 4.1 shows the expected communication across values of m with $n = 50$. We can see that p_{comm} is practically equal to $\frac{m}{n}$ for all values of m with mean squared error on the order of 10^{-6} for 100,000 samples.

4.5 Computational setup

Running experiments on more complex environments and with larger numbers of agents using the framework developed in 3.3.1 proved infeasibly slow as it was largely implemented in native python. As well as this the data structure used record results produced very large file sizes meaning they took a long time to process and used up a significant proportion of the storage available. Therefore, for the ESPC, ES comparisons I improved my approach to experiments by taking advantage of libraries to speed up training. The main benefit came from moving from many instances of a single environment to Gymnasium vectorized environments. This allowed us to pass a vector of actions to the environment and for the environment updates to be carried out asynchronously in parallel, resulting in a significant improvement of performance. As well as this I used JAX [4] just in time compilation to accelerate the neural network responsible for policy calculations. I addressed the issue experiment file sizes by recording only the data needed to satisfy the metrics established in section 3.2. This significantly reduced the storage requirements, however, it also meant results were less flexible in analysis and harder to debug.

Even with the improved experiment setup, due to having very limited amounts of computation at my disposal I was not able to conduct experiments of the scale I desired. Ideally for an algorithm like ES I would use a number of agents at least an order of magnitude higher and train on an order of magnitude more episodes, meaning I would be able to compare with ESPC on the complex environment such as the more harder environments in the MuJoCo [28] and Atari [2] suits that ES proved particularly effective on. Comparison would also have benefitted from the use of larger models such as CNNs for use on the Atari suit. However, I still feel the experiments that were conducted give an accurate picture of the performance of ESPC compared to ES and hope for future work to test in more complex environments.

4.6 Comparison with existing algorithms

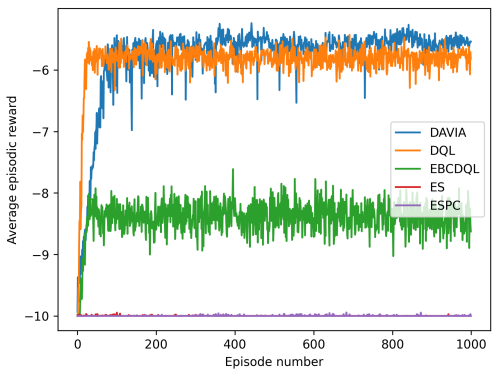
To gain an understanding performance of ESPC, I compared it to the DQL, EBCDQL, DAVIA and ES algorithms in the Frozen Lake environment from Gymnasium as well as the custom environment Windy Grid.

Windy Grid is a 5x5 gridworld where the objective is for the agent to move from the upper left

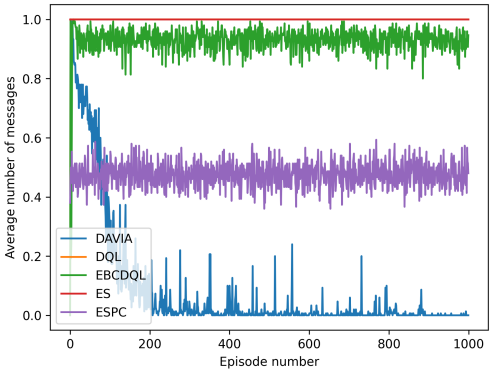
square to the lower right. The agent receives -1 reward every step except the final step in which it receives a reward of 0. The agent can move left, right, up, down or stay in the same place. At each step there is a 30% chance of being independently pushed one square in the directions up and/or left in addition to the movement from the action taken. For the ES and ESPC policy I used a feedforward neural network with 2 hidden layers of size 4 as well as a momentum optimizer. I carried out training with $n = 30$ agents over 1000 episodes and repeated the trial 5 times averaging the results over the trials.

Relative to the value based algorithms (DQL, EBCDQL, DAVIA) on Windy Grid ES and ESPC performed poorly in terms of quality of learning. Figure 4.2 shows that they both fail to learn an effective strategy achieving the near minimum reward on every episode. They performed better on the Frozen Lake environment shown in figure 4.3. However, all algorithms failed to find a consistent strategy to reach the goal. The size and frequency of messages sent by ESPC in Windy Grid were lower than all algorithms except DAVIA. On Frozen Lake, since both DAVIA and EBCDQL communicated minimally ESPC was outperformed.

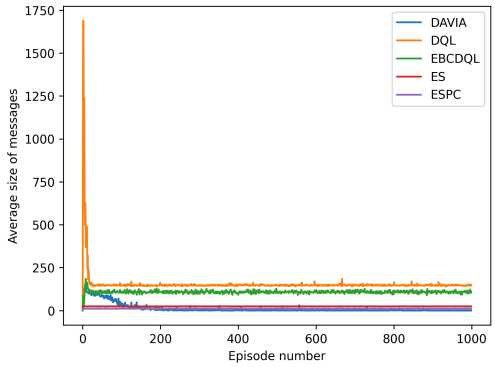
ES has showed strong performance in the past on much more complex environments than those tested here, and on which it is not feasible to run the other algorithms in their implemented state. The reason that ES and ESPC fail to learn effectively on these environments is due the rewards for all agents often being the same. Meaning the expected update to the parameters is zero. In the case of Windy Grid there will likely be some random update to the parameters as they will receive a negative reward. However, for Frozen Lake the update will often be zero as there is no reward until the terminal is reached. A potential remedy to this is to use novelty to encourage exploration as suggested in section [8] and discussed in 1.2. Alternatively virtual batch normalization as discussed in [24] could be introduced to improve exploration.



(a) The average episodic reward received by agents

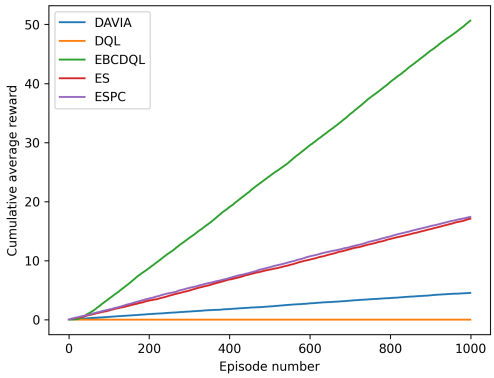


(b) The average number of messages sent from agents to central learner

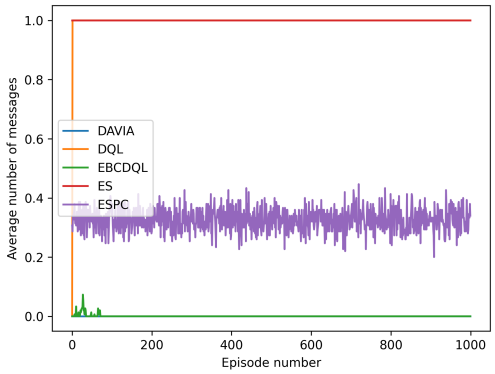


(c) The average size of messages sent from agents to central learner

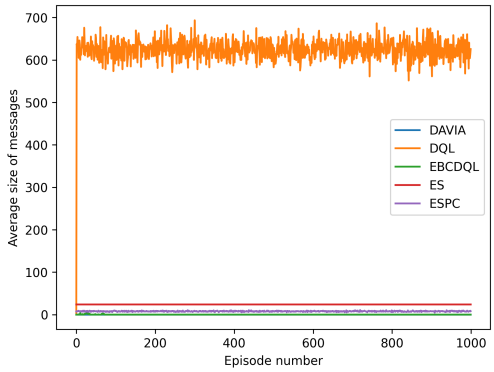
Figure 4.2: Results from the algorithm comparison in the Windy Grid environment for DAVIA, DQL, EBCDQL, ES, and ESPC with $n = 30$ over 1000 episodes and 5 trials.



(a) The cumulative average reward received by agents



(b) The average number of messages sent from agents to central learner

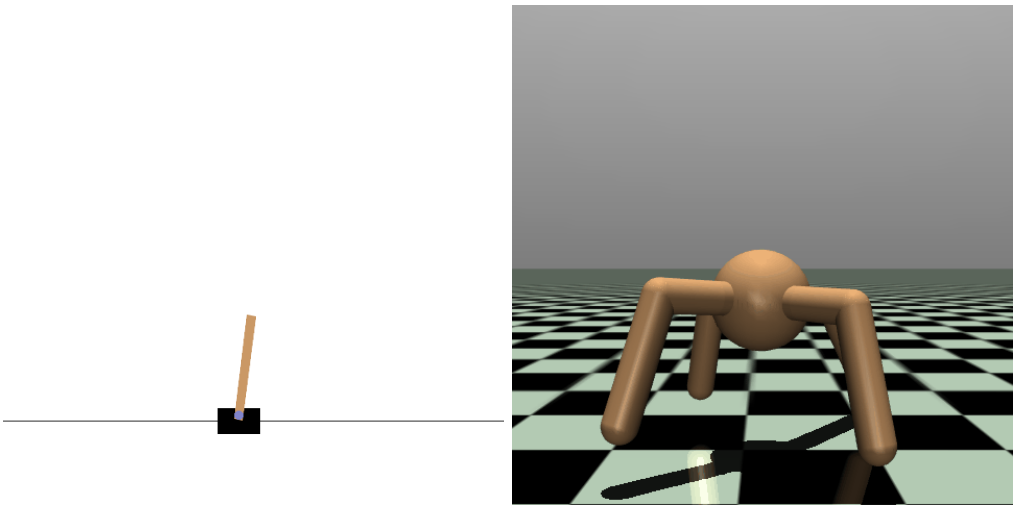


(c) The average size of messages sent from agents to central learner

Figure 4.3: Results from the algorithm comparison in the Frozen Lake environment for DAVIA, DQL, EBCDQL, ES, and ESPC with $n = 30$ over 1000 episodes and 5 trials.

4.7 Direct comparison between ESPC and standard ES

ESPC is designed to reduce the communication between the agents and the central learner to an amount equivalent to that of ES with m agents. We are therefore interested in evaluating an instance of ESPC which has n agents, but communicates like it has m agents, against an instance of ES with m agents. We note that this is roughly equivalent to using n agents with a fixed probability of communication of $\alpha = \frac{m}{n}$. Communicating with αn agents out of n agents with fixed α is exactly equivalent to n agents communicating with fixed probability α in the limit as $n \to \infty$. For ESPC to be effective it must learn better than ES with m agents, otherwise it would show it is wasting computational resources for no benefit. In the experiment I also include an instance of ES with n agents as a benchmark for performance. The bias ESPC introduces means the approximation of



(a) The Cart Pole environment (b) The Ant environment

the policy gradient will likely not be as accurate as ES with n agents. I therefore hypothesize that the performance of ESPC lies somewhere between that of ES with n and m agents.

To test this hypothesis I conducted an experiment of this nature on the Gymnasium Cart Pole environment, and the MuJoCo Ant environment using $n = 50$ and $m = 25$. I chose these environments as they are sufficiently complex, taking many iterations to converge to good solutions, meaning it is likely differences in performance across algorithms will be clear. They are also lightweight enough that it is computationally feasible to train agents on them with limited resources. The Cart Pole environment (figure 4.4a) is a physics based environment in which the aim is to control a cart such that the pole stays as close to vertical as possible for a long as possible. At each step the agent observes the cart position, the cart velocity, the pole angle, and the pole angular velocity. In response to this it can move left or right. It receives a reward of 1 for every timestep for which the pole is within a threshold of the vertical. If the pole leaves this region then the episode terminates. Ant (figure 4.4b) is an environment from the MuJoCo suit where the goal is to control a four legged robot such that it walks forward[1].

The experiments evaluate the average discounted reward (μ^{reward}) and the average number of messages sent ($\mu^{\text{frequency}}$). The size of messages is not measured as, for both algorithms, it is a constant multiple of the number of messages sent. I used the Gaussian assumption of utility distribution stated in section 4.4 and set the rolling average parameter $k = 5$. The policy for both

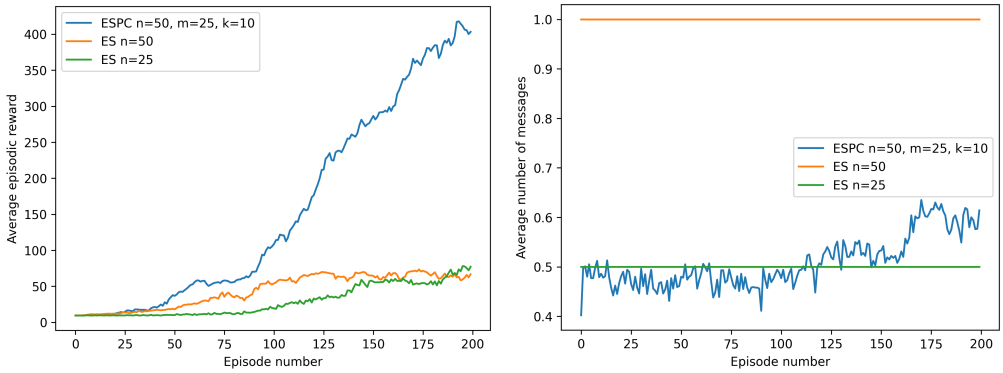
[1]Details of the action, observation and reward structure can be found at `https://gymnasium.farama.org/environments/mujoco/ant/`

ES and ESPC was a feedforward neural network with 2 hidden layers of size $8, 64$. I used Adam optimizer [15] with a learning rate of $\alpha = 0.01, 0.005$, an L_2 regularization coefficient of $0.05, 0.05$, a perturbation standard deviation of $\sigma = 0.05, 0.02$ and a discount factors of $\gamma = 1, 0.99$ where the parameters are for the Cart Pole and Ant environments respectively and are the same for both algorithms. Training was carried out over 200 episodes and repeated in 20 and 3 trials respectively, averaging over the results from the trials.

ESPC performed well in terms of both quality of learning and communication when compared to ES. Results on the cart pole environment are shown in figure 4.5. For every episode in training after episode 50 ESPC with $n = 50$ achieves a higher μ^{reward} than that of ES with both $n = 50$ and $n = 25$ and sends a number of messages similar to ES with $n = 25$. It can be seen in figure 4.5a that ESPC achieves significantly more episodic reward than ES with a continuing upward trend whereas ES is relatively stagnant at low levels of reward.

The results on the Ant environment are shown in figure 4.6 where ESPC achieved μ^{reward} greater than of ES with $n = 50$ and $n = 25$ for every episode in training and a $\mu^{\text{frequency}}$ in line with that of ES with $n = 25$. After episode 125 we can see that the μ^{reward} for ESPC begins to gradually increase whereas for ES with $n = 50$ it stays constant and ES with $n = 25$ it drops.

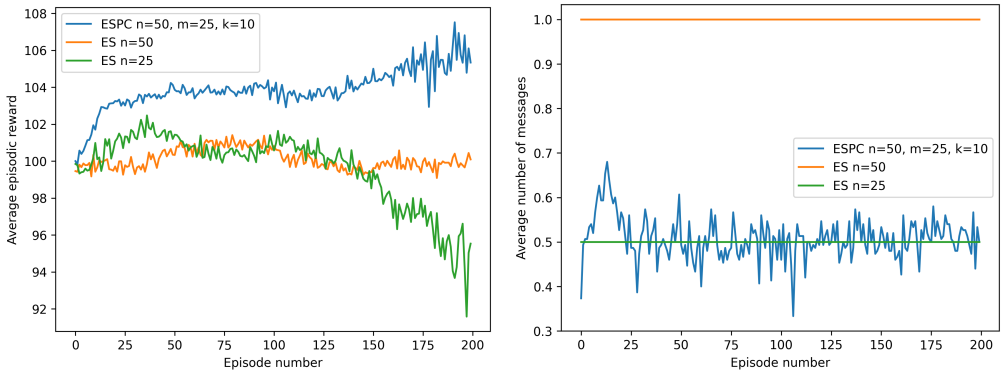
These results beat the expectation of a level of performance between the two ES instances. I theorize that this is due to greater exploration when using the probabilistic communication scheme as samples are biased towards higher rewards. It is also noticeable that ESPC exhibits a communication jump at the beginning of training, where communication converges to the expected amount of m out of n per episode, usually starting out significantly lower, this is further explored in the following sections.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to central learner

Figure 4.5: Results for the direct comparison of ES with $n = 25, 50$ and ESPC with $n = 50$, $m = 25$ and $k = 5$ on the Cart Pole environment over 200 episodes and 20 trials.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to central learner

Figure 4.6: Results for the direct comparison of ES with $n = 25, 50$ and ESPC with $n = 50$, $m = 25$ and $k = 5$ on the Ant environment over 200 episodes and 3 trials

4.8 Varying communication in ESPC

A key advantage of ESPC over ES is the ability to tune the expected amount of communication on each episode. However, reducing communication introduces bias to the estimate of the gradient and will thus affect the quality of learning achieved. I therefore conducted an experiment where I varied the parameter m hypothesizing that quality of learning will decrease with the amount of communication.

I used $n = 50$, $m = 5, 15, 30, 45$ and the Gaussian assumption stated in section 4.4 with $k = 10$.

The experiment was conducted on Cart Pole and Ant. I included an instance of ES with $n = 50$ as a benchmark for learning performance. The other details of the systems were the same as in section 4.7.

The results achieved by varying the communication parameter m on the Cart Pole environment are presented in figure 4.8. It can be seen that for all values of m ESPC outperforms ES in terms of μ^{reward} with $\mu^{\text{frequency}}$ roughly equal to m/n times that of ES per episode, this is approximately in line with the result from section 4.4. Noticeably, the learning performance of lower values of m is largely greater than that of higher values, however, the smoothness of the curve decreases. For $m = 5$ we can see fast but erratic increases in reward and for $m = 45$ there is smoother and more gradual increases. This is likely due to fact that for low values of m we are only taking into account the largest rewards meaning we take a large step in their direction in parameter space, and thus we get fast increases in reward. However, the direction of the steps is not consistently close to the true policy gradient meaning we often get rapid drops in reward as well. For high values of m the policy gradient is more accurate, however, we are averaging over more, smaller samples so the stepsize will be smaller. This leads to the smoother optimization seen in the figure. Over all the instances, ESPC with $m = 15$ performs the best indicating that there may exist an optimal amount of communication to maximize learning.

Figure 4.9 show the performance on the Ant environment. We can see for all values of m apart from $m = 5$, μ^{reward} is greater than that of ES for almost all episodes, however, for $m = 5$ around episode 120 the performance drops catastrophically. This collapse in performance is likely due to approximation of the policy gradient being too crude due to the low number of samples used to calculate it and the bias introduced by the probabilistic communication scheme.

These results show that there may exist a communication threshold, at which the approximation of the gradient by ESPC becomes too poor leading to significant degradation in learning performance, however, above this threshold performance will often match or exceed that of ES. As discussed in [25] and section 2.4.4 due to the fact that ES is effectively computing randomized finite difference estimates of the policy gradient, it will scale poorly with the number intrinsic parameters in the optimization problem. In the Ant environment the intrinsic dimension of the problem is larger than in the Cart Pole environment, so the approximation of the gradient with few samples will be much worse. This likely explains the presence of a performance collapse in Ant and not

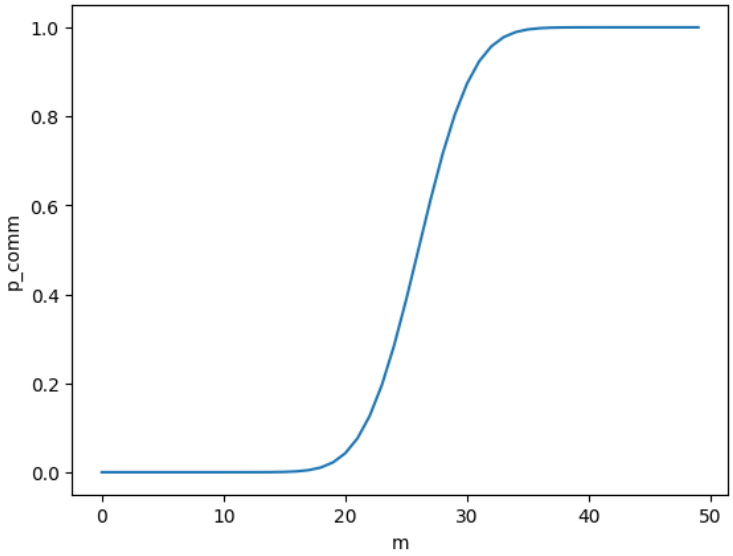
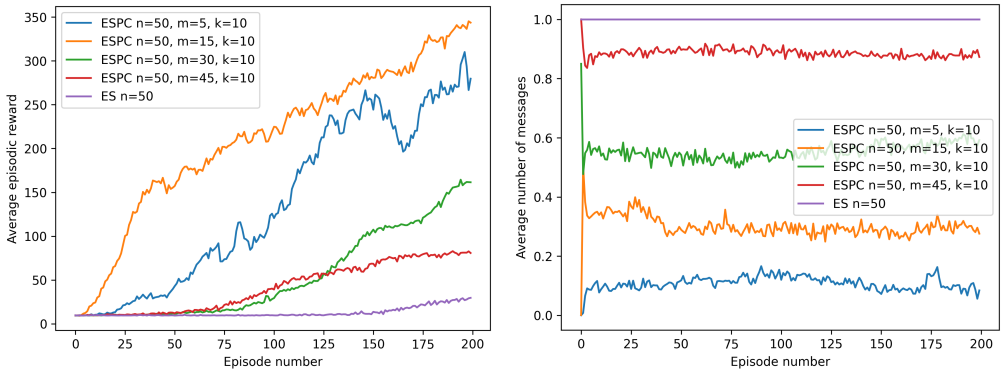


Figure 4.7: The probability of communication on the first episode with $n = 50$ for different values of m.

Cart Pole with the same number of samples. Further investigation is necessary to confirm this. Future work could focus on looking at this effect in greater detail as well as theoretically analysing the effect of varying the m parameter.

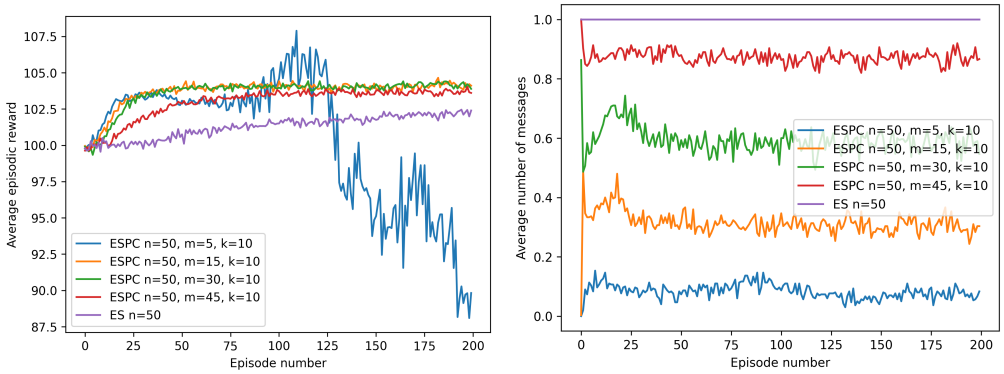
Additionally, we observe that the initial anomalous values of communication at the beginning of training are highly influenced by the value of m. This is because on the first episode we only have one utility sample, so the rolling mean is equal to the value of the sample. The variance is also zero but to avoid a division by zero error we set it to 1 on the first episode. Therefore, the probability that a sample is greater than another is 0.5. The probability of communication is thus only a function of m and n on the first episode, explaining the phenomenon observed. A plot of the probability of communication on the first episode as a function of m with $n = 50$ is shown in figure 4.7 we can see communication is rapidly pushed to the extremes away from 0.5 as m is perturbed around $m = 25$.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to central learner

Figure 4.8: Results for the communication comparison of ES and ESPC on the Cart Pole environment with $n = 50$ where the communication parameter is varied $m = 5, 10, 15, 30, 45$ and $k = 10$ over 200 episodes and 20 trials.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to central learner

Figure 4.9: Results for the communication comparison of ES and ESPC on the Ant environment with $n = 50$ where the communication parameter is varied $m = 5, 10, 15, 30, 45$ and $k = 10$ over 200 episodes and 6 trials.

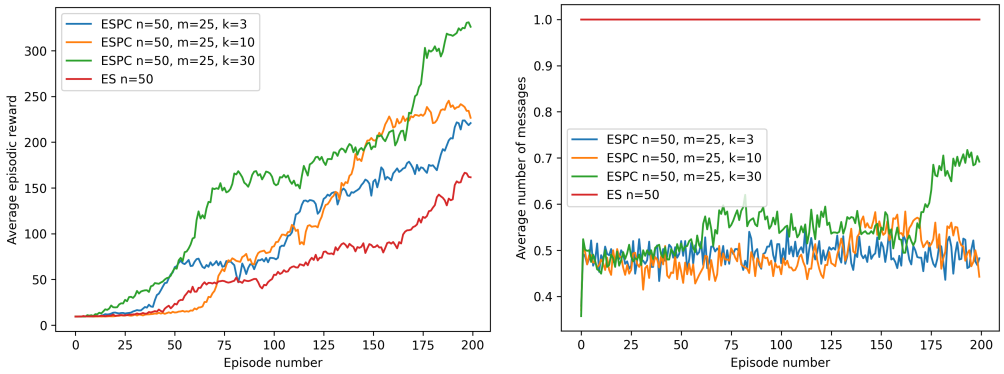
4.9 Effect of Gaussian utility distribution parameter

To evaluate how the length of the horizon for the rolling averages used to parameterize the assumed Gaussian distribution of utility from section 4.4 affects the performance of ESPC, I conducted an experiment where I varied the rolling average horizon k with all other parameters fixed. I used $k = 3, 10, 30$ with $n = 50$, $m = 25$ on the Cart Pole and Ant environments and compared against an instance of ES with $n = 50$. All other system details were the same as in section 4.7. I trained

over 200 and 300 episodes and averaged the results across 10 and 5 trials respectively.

Figures 4.10 and 4.11 show the results of varying the horizon k of the samples with which we estimate the utility distribution on the Cart Pole environment. All instances of ESPC attain rewards that exceed that of ES for most training episodes in both Cart Pole and Ant. However, there is no obvious effect of the horizon length on learning performance.

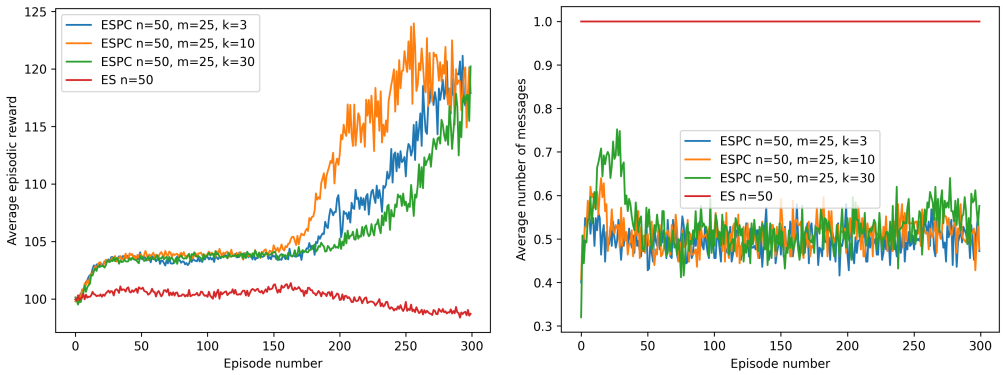
The effect of the horizon length on communication is clear. We can observe that communication more closely tracks the learning as k increases. This is particularly obvious at the beginning of figure 4.11b where we see different jumps in communication for different values of k that last for a period proportional to k when the reward achieved by the instances is increasing by roughly the same amount. We also see this in 4.10b. For $k = 30$ there are two distinctive jumps in communication at episode 60 and 170 corresponding to rapid increases in reward around these times. Similarly, we see a smaller increase in communication for $k = 10$ around episode 130. This effect occurs because, since a sample is included in the calculation of the rolling mean and variance to determine its own probability of communication, with larger k a new reward sample has less of an effect on these parameters as the number of samples used to calculate them is higher. Therefore, an extreme sample has a larger difference from the updated mean and is thus more likely to be communicated. Therefore, when reward is increasing fast the amount of communication also increases.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to the central learner

Figure 4.10: Results for the horizon comparison of ES and ESPC on the Cart Pole environment with $n = 50$ and $m = 25$ where the rolling average horizon parameter is varied $k = 3, 10, 30$ over 200 episodes and 10 trials.



(a) The episodic average reward received by agents

(b) The average number of messages sent from agents to the central learner

Figure 4.11: Results for the comparison of ES and ESPC on the Ant environment with $n = 50$ and $m = 25$ where the rolling average horizon parameter is varied $k = 3, 10, 30$ over 300 episodes and 5 trials.

4.10 Limitations and omitted experiments

In these experiments we tested ES and ESPC with the same parameters, this allows us to observe only the changes in performance that are a direct consequence of the change in algorithm. However, an alternative method would be to tune each algorithm individually and compare the thier peak performance as this will not necessarily occur for both when their parameters are the same. Doing this in addition to the experiments performed would be informative.

I only tested on two environments with few trials and low number of episodes due to long runtimes for experiments. Increasing the number of environments would demonstrate the difference in performance between the algorithms more clearly and repeating experiments for longer training runs in many more trials would allow us to give bounds in performance on each of them and observe their asymptotic performance.

Additionally, we did not employ some techniques to combat the brittleness of ES such as virtual batch normalization that have been effective in the past. This is because in the original publication they only became necessary when using CNNs on the Atari environments where enhanced exploration was required. However, in the simple gridworlds these techniques could have potentially improved performance.

I also did not explore in great detail the quality of the Gaussian assumption made in 4.4.

This analysis could provide insight into why ESPC performed as it did and suggest ways in which we could improve performance further. Additionally, a comparison of the Gaussian against other distributions, especially those which could exhibit skew, would provide insight into the quality of the assumption. Similar to this a comparison of different utility functions would shed light on the quality of justification made in section 4.3.

Chapter 5

Conclusion

I have explored the current methods in distributed reinforcement learning and numerically compared them. Specifically I focused on evaluating, DQL, EBCDQL, DAVIA, and ES. I discussed how I implemented the algorithms by developing a framework to standardize their interfaces leading to a system where I could easily pair algorithms with environments in experiments. In the experiments I found that the value based methods are more suited to the Simple Grid environment than ES, finding effective solutions where ES failed to get any significant reward. However, on the Frozen Lake environment ES performed better where DQL and DAVIA were unable to find effective solutions. Furthermore, I discussed the drawbacks of the value based algorithms including the limitations of tabular methods and ways in which the algorithms could be amended to make them more suitable for complex environments. I discussed the communication performance of each of the algorithms highlighting the adaptive communication of EBCDQL and DAVIA as well as the small message sizes of ES. I concluded that despite its performance on Simple Grid, ES was the most versatile and promising of the algorithms, this was heavily influenced by past performance on famously challenging environments. In terms of communication efficiency the main drawback of ES is the frequency at which it sends messages.

To improve on this I proposed a probabilistic communication scheme in which samples with the highest magnitude were prioritized. I showed numerically that this scheme communicates less than the original communication scheme for evolution strategies while often performing better in terms of quality of learning. Specifically in the Cart Pole and Ant environments ESPC outperformed ES with the same number of agents while communicating half the number of messages. I explored the

effect of varying the communication parameter and the horizon length used to parameterize the utility distribution of ESPC. I found that increasing the amount of communication led to slower but more steady learning and decreasing it lead to fast erratic learning. In the case of very low amounts of communication I discussed how the approximation of the gradient can break down leading to a catastrophic collapse in learning performance. I also found that the horizon length had little effect on the learning performance of the algorithm but lead to closer tracking of reward by communication for longer horizons.

Future work could explore further choices of the utility function for ESPC and the assumed distribution induced by the function. In addition to this a mathematical analysis of the scheme could look to find how distributions of utility are induced by the choice of the utility function. Alternatively an approach could be to parameterize the utility function and use meta-learning for example MAML [9] or a second layer of ES to optimize it. Another possibility would be to find a scheme in which the expected number of agents communicating at each episode m varies adaptably during training, a way to achieve this could be to include the communication parameter in the model and update it as part of the ES update process. Finally, my work in this project has exclusively focused on the communication between agents and the central learner, however, with the efficiency of modern algorithms in this regard we are now limited by the communication of the new model parameters to from the central learner to the agents. Work exploring ways to compress these weights or only adaptably communicate them could lead to further increases in overall communication efficiency.

Bibliography

[1] Saminda Abeyruwan, Laura Graesser, David B. D'Ambrosio, Avi Singh, Anish Shankar, Alex Bewley, Deepali Jain, Krzysztof Choromanski, and Pannag R. Sanketi. i-sim2real: Reinforcement learning of robotic policies in tight human-robot interaction loops, 2022.

[2] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Int. Res.*, 47(1):253–279, may 2013.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Víctor Campos, Xavier Giro i Nieto, and Jordi Torres. Importance weighted evolution strategies, 2018.

[7] Tianyi Chen, Kaiqing Zhang, Georgios B. Giannakis, and Tamer Başar. Communication-efficient policy gradient methods for distributed reinforcement learning. *IEEE Transactions on Control of Network Systems*, 9(2):917–929, 2022.

[8] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 5032–5043, Red Hook, NY, USA, 2018. Curran Associates Inc.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017.

[10] Konstantinos Gatsis. Federated reinforcement learning at the edge, 2021.

[11] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

[12] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[13] Asaf Shachar (https://math.stackexchange.com/users/104576/asaf shachar). Normalized vector of gaussian variables is uniformly distributed on the sphere. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/1864519 (version: 2020-05-18).

[14] Kenji Kawaguchi. Deep learning without poor local minima, 2016.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[17] Aritra Mitra, George J. Pappas, and Hamed Hassani. Temporal difference learning with compressed updates: Error-feedback meets reinforcement learning, 2023.

[18] Ryan W. Murray, Brian Swenson, and Soummya Kar. Revisiting normalized gradient descent: Fast evasion of saddle points. *IEEE Transactions on Automatic Control*, 64:4818–4824, 2017.

[19] Daniel Jarne Ornia and Manuel Mazo. Event-based communication in distributed q-learning, 2021.

[20] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[22] André Susano Pinto, Alexander Kolesnikov, Yuge Shi, Lucas Beyer, and Xiaohua Zhai. Tuning computer vision models with task rewards, 2023.

[23] Jiaju Qi, Qihao Zhou, Lei Lei, and Kan Zheng. Federated Reinforcement Learning: Techniques, Applications, and Open Challenges. *arXiv e-prints*, page arXiv:2108.11887, August 2021.

[24] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[25] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017.

[26] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[27] Paulo Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52(9):1680–1685, 2007.

[28] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[29] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[30] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew

R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[31] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

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

[33] Qingxiang Wu, Ning Sun, Tong Yang, and Yongchun Fang. Deep reinforcement learning-based control for asynchronous motor-actuated triple pendulum crane systems with distributed mass payloads. *IEEE Transactions on Industrial Electronics*, pages 1–10, 2023.