

École polytechnique de Louvain

Deep Reinforcement Learning for the Expert Advice Multi-Armed Bandit

Author: Nikraftar Khiabani ARMAN

Supervisor: Macq BENOÎT

Readers: Ghislain MÉLANIE, Bary TIM, De Vleeschouwer CHRISTOPHE

Academic year 2023–2024

Master [120] in Electro-mechanical Engineering

Acknowledgements

I would like to express my gratitude and thank those who helped make this work possible, as such, special thanks to:

Prof. Benoit Macq: For his support, encouragement and mentorship throughout the year.

Melanie Ghislain and Tim Bary: For their guidance, mentoring and exceptional help with all aspects of this thesis. Their time and expertise have helped in the making and the improvement of this work.

Christophe De Vleeschouwer: For agreeing to read my thesis and to participate in my defense.

My mother: For my upbringing and the unwavering support I have received my whole life.

I would also like to thank all of those who have contributed to this work indirectly.

Abstract:

The expert-advice Multi-Armed Bandit (MAB) is a framework in which we rely on the advice gathered from the experts to perform a task. In this thesis, we discuss a classification problem where the experts serve as classifiers. The goal is to optimize an objective function describing the correct classification rate as well as the cost associated with consulting the experts over a series of turns. We study the case where a series of stochastic experts, each with different costs and classification capabilities, must be used to perform binary classification on a large group of patients.

The expert-advice MAB problem is modeled as a Reinforcement Learning problem (RL), where a deep-learning agent is responsible for making the decision of which experts to consult at each turn based on the data available on the experts. The RL algorithms are trained to maximize the cumulative reward signal, thus maximizing the objective function.

Different implementations of the MAB as RL environments are designed to exploit the symmetries of the problem and to incorporate some RL tricks, such as reward shaping, to improve the performance of RL based algorithms. The performance of these RL based approaches is then compared to that of a pre-existing algorithm for this type of problem.

The results indicate that while RL-based approaches can effectively optimize decision-making in small-scale problems under strong assumptions, they encounter challenges related to scalability and robustness. The RL methods successfully identified local optima with higher expected rewards, but this came at the cost of increased computational demands. These findings contribute to the understanding of how RL can be applied to expert-advice MAB problems and suggest areas for future research to address these limitations and improve the practical applicability of RL-based solutions.

Table of Contents

1 Introduction	3
2 Theoretical Background	4
2.1 The Multi-Armed Bandit (MAB) Problem	4
2.2 The Concept of Regret	5
2.3 MAB Algorithms	5
2.4 Problem Statement	8
2.5 The General RL Problem	9
2.6 Generalized Policy Iteration	13
2.7 Policy Evaluation	13
2.8 Policy Improvement	14
2.9 Monte-Carlo Methods $TD(\infty)$	14
2.10 Temporal Difference-Learning $TD(n)$	15
2.11 Function Approximation	16
2.12 Policy-Gradient Methods	17
3 State of the Art	21
3.1 Recent Advances In RL	21
3.2 Existing Methods	21
4 Material and Methods	23
4.1 The Patients and the Experts	23
4.2 The Need for A Forecaster	25
4.3 Weighted Majority Forecasters	25
5 Experiment Setup	34
5.1 Approach 1 (All Experts In One Step)	34
5.2 Approach 2 (All Experts In One Step + Forecaster)	36
5.3 Approach 3 (One Expert Per Step + Forecaster)	36
5.4 Ordering	37
5.5 The Extended Softmax Method:	40
5.6 The Policies	40
6 Results	42
6.1 Choice of Metrics	42
7 Conclusion	49
Bibliography	51
8 Annexes	53
8.1 Experiment 2 (Stochastic Bandit With RL)	53
8.2 Code	56

1 Introduction

It has been observed in recent years that AI based systems are capable of outperforming medical experts when it comes to cancer screening programs [1], [2]. Namely, deep learning-based agents are able to attain better classification (labeling) when compared to individual experts in the medical imaging fields. An example of this is breast cancer screening where AI based methods had better classification accuracy [3]–[5].

It has however also been observed that coalitions of medical experts, by communicating and trying to convince each other, still outperform the best deep learning (DL) based models in terms of classification accuracy [6]. In terms of brain imaging accuracy, the state-of-the-art DL based models get outperformed by a double expert view (2 expert coalition) [7], [6].

The thesis focuses on strategies that leverage a coalition of experts to increase classification accuracy while minimizing consultancy costs. More specifically, we aim to know if deep Reinforcement-Learning (RL) based agents can exploit a coalition of experts effectively.

We begin by discussing the Multi-Armed Bandit (MAB) framework (Section 2.1), which serves as a foundation for modeling the thesis problem as an expert advice MAB scenario (Section 2.4). Following this, we implement RL based strategies to surpass the performance of existing methods (Section 5). A detailed introduction to RL is also provided (Section 2.5).

The main challenge faced in this problem is the lack of information regarding the confusion matrices of the experts at the start of the classification, as well as the biased and correlated advice which the different experts give. The experts are consulted not only to obtain classifications, but also to understand how competent they are when it comes to their classifications. All of this has to be done while aiming to minimize both misclassifications and consultancy costs.

2 Theoretical Background

2.1 The Multi-Armed Bandit (MAB) Problem

Originally termed from the many arms of a slot machine, the term MAB is used to describe a wide array of problems [8]. A base example MAB problem is described here:

The problem is as follows [9]:

- The agent has K arms a in its disposition who each have reward distribution D_a associated to them.
- A total number of T rounds are played where an arm a_t is chosen by the agent for each round t . With $T > K$
- Upon playing arm a_t , the agent is given a reward r_t sampled independently from the distribution D_{a_t} of arm a_t .
- The reward distributions D_a for each arm a are stationary, meaning that they are the same regardless of the round t and the past arms taken by the agent.
- After each round, the agent only observes the reward r_t with no information revealed about the distributions of the other arms.

The goal of the agent is to maximize the cumulative rewards collected over the T rounds. If the underlying reward distributions of each arm D_a were known in advance, then this problem would be trivial. Simply taking the arm with the highest expected reward at each turn would yield the highest expected cumulative reward. The agent in this scenario is said to be exploiting the best arm. The agent would ideally like to exploit all of the time, but the imperfect knowledge of the reward distributions makes it more complicated.

If the agent mistakes a suboptimal arm to be the best and decides to exploit that arm every single turn, it would miss out on the opportunity of obtaining more rewards associated to playing a better arm. Knowledge about the reward distributions of the arms is thus important to avoid this unideal scenario. The only way to obtain such knowledge however, is by performing exploratory rounds. The exploratory rounds are rounds where the agent decides to play an arm thought to be inferior to another. Exploratory rounds are done with the hope of finding a better arm to exploit than the one currently thought to be the best.

Exploration is essential for exploitation as a strategy to be viable, but performing an exploratory round means the agent is not exploiting. As such, it is crucial to find the right balance between the two for obtaining the highest expected reward.

2.1.1 Real-World Applications

The MAB framework can be used to model various real-world scenarios. Below are examples of real-world problems that can be modelled as MAB problems.

1. Healthcare:

Medications which medical experts prescribe to patients might be considered as actions in an MAB problem, with the reward signal being the efficiency of that medication on dealing with the patient's illness.

2. Clinical Trials:

In the case of emergency outbreaks, scientists need to rapidly come up with a series of medications to counter the newly emergent outbreak. There might be an urgent clinical study where scientists pit different medications against each other by giving them to ill patients and recording the effectiveness of said medications on the patient's health. Now if we want to not only discover the best medication but also to minimize the number of losses among the experiment subject patients, we might want to use some form of MAB algorithm that would aim to maximize the number of patients cured over the experiment, while also providing a high chance of identifying the most effective medication for mass production and large-scale application.

Notice that the algorithm facing the problem above might also have to wait a certain amount of time before observing how the patient's health evolves after taking the medication. For instance, one might compare the patient's health to its state two weeks before taking the medication [10]. This is delayed feedback which complexifies the original problem even further [11].

3. Search Engine Optimization:

In search engine optimization, MABs are used to optimize the presentation of search results. By monitoring user interactions, the algorithm can learn which search results are more relevant and engaging to specific users, thereby improving user experience and increasing user engagement [12].

4. Financial Investments:

In financial investments, MABs are used to optimize the allocation of capital across different assets and investment strategies. By monitoring the performance of assets and investment strategies, the algorithm can learn which ones are more profitable and less risky, thereby improving investment performance and reducing risk [13], [14].

2.2 The Concept of Regret

In order to be able to compare MAB algorithms, we define the metric of regret. The agent is said to accumulate regret at each turn whenever it plays a sub-optimal arm a_t . Depending on how severe the difference between the best arm and the one that was played, the more regret is accumulated. Regret R is essentially the same as the cumulative reward except that this quantity is independent of the average reward of the arms [15].

$$R(T) = \mu^* * T - \sum_{t=1}^T \mu(a_t)$$

Where the regret $R(T)$ at time T represents the difference between the maximum amount of reward which could have been obtained in expected value compared to the expected value of the reward obtained for the given series of actions.

$\mu(a) = \mathbb{E}D_a$ represents the expected rewards of each arm, and $\mu^* = \operatorname{argmax}_a \mu(a)$ is simply the best arm.

The regret metric being independent of the baseline level of the rewards —such as the mean of arm rewards or the DC level of rewards— makes it convenient to compare algorithms on different MAB problems.

It is important to realize that the agents will base their decisions on the sampled rewards they get from the arms. Given the fact that the rewards are sampled according to distributions D_a , the arms taken by an agent a_t are themselves random variables.

Therefore the metric that interests us more is technically the expected value of cumulative regret, $\mathbb{E}R(T)$, simply called regret from now on.

2.3 MAB Algorithms

2.3.1 Explore First, Commit Later

This algorithm will operate in two distinct phases, phase one will constitute a phase of exploration where the algorithm tries each and every single arm for a set number of times (m), gathering information on the rewards obtained from each of the actions to produce estimators

$$\begin{aligned}\tilde{\mu}_a(t) &= \frac{1}{T_a(t)} \sum_{s=1}^t \mathbb{I}\{A_s = a\} r_s \\ T_a(t) &= \sum_{s=1}^t \mathbb{I}\{A_s = a\}\end{aligned}$$

for the underlying $\mu(a)$ of each a . The second phase (exploitation) is a phase where the arm with the highest estimated expected reward is played for the rest of the play [15].

So the algorithm can be implemented as follows:

$$A_t = \begin{cases} (t \bmod k) + 1 & \text{if } t \leq mk \\ \operatorname{argmax}_i \tilde{\mu}_i(mk) & \text{if } t > mk \end{cases}$$

m is going to be a parameter which the engineer can tune for this algorithm to obtain better results. It can be shown in the case where $K = 2$ that the optimal $m = \max\left(1, \frac{4}{\Delta^2} \log\left(\frac{T\Delta^2}{4}\right)\right)$. with $\Delta = \mu^* - \mu_{\text{other}}$. [9]

It can be shown that the expected regret of this simple algorithm has a complexity of $R(T) \leq O\left(T^{\frac{2}{3}}(\log T)^{\frac{1}{3}}\right)$. This reflects the expected regret in the favorable scenario where the algorithm successfully identifies the best arm or another arm sufficiently close to the best one. However, there is a scenario where the algorithm fails to

find the best arm and achieves a regret of $O(T)$. Fortunately, this less favorable outcome occurs only in rare circumstances with a probability of $O(T^{-4})$, meaning that the expected regret is primarily influenced by the favorable scenario. Despite this, the regret level achieved, even in the favorable case, is not the optimal level of achievable regret complexity for this type of problem, primarily because the exploration phase is quite costly. A significant portion of resources is allocated to testing sub-optimal arms, which the next few algorithms aim to address [15], [16], [9].

2.3.2 Epsilon-greedy

The Epsilon-greedy algorithm was designed to more evenly spread the exploration-exploitation steps of the agent. At turn t , it exploits the best estimated action with $(1 - \varepsilon_t)$ probability or performs an exploratory step (with ε_t probability). On the exploratory turns, the arm to play is chosen randomly with a uniform distribution over all arms.

So the algorithm can be implemented in the following fashion:

$$A_t = \begin{cases} \text{Uniform}(A) & \text{with probability } \varepsilon_t \\ \text{argmax}_i \tilde{\mu}_i(t-1) & \text{with probability } 1 - \varepsilon_t \end{cases}$$

The actual choice of how ε_t is modeled with respect to t is called exploration-annealing. And it plays an important role in the performance of the algorithm.

- Choosing a constant ε_t : makes sure that all the actions are taken and continually taken by the agent with a given least probability of $\frac{\varepsilon_t}{K}$. This makes the constant ε epsilon-greedy algorithm the least favorable in terms of expected reward complexity guarantees (achieving $O(T)$ regret). It does not however mean that this algorithm is useless as the fact that it never stops to explore can be beneficial in applications where the underlying reward distributions of the arms may vary over time such as non-stationary bandits but also as an exploration driver for RL training. Nonetheless, if the T is known in advance and one is able to run experiments on the environment for more than just a single play, the ε value can be tuned to make this algorithm competitive with the other algorithms.
- Choosing $\varepsilon_t = t^{-\frac{1}{3}}(K \log t)^{\frac{1}{3}}$ is known to have an expected regret complexity of $O(K \log t)^{\frac{1}{3}}$.

The Explore First, Commit Later algorithm and the Epsilon-greedy, do not use the reward signals obtained from their play to adapt their exploration. The following algorithms have adaptive exploration, and perform better in many practical applications [9], [17], [16].

2.3.3 Softmax (Boltzmann Exploration)

Softmax differs from Epsilon-greedy in that it does not perform exploratory steps uniformly across all arms. Arms with higher estimated expected rewards are explored more often. Both the exploration and the exploitation of the Softmax algorithm is driven by the Boltzmann distribution:

$$p_i(t+1) = \frac{e^{\frac{\tilde{\mu}_i(t)}{\tau}}}{\sum_{j=1}^K e^{\frac{\tilde{\mu}_j(t)}{\tau}}}, \forall i = 1, \dots, K$$

The parameter τ is a parameter that determines how exploratory the algorithm is. Higher values of τ yield a more uniform distribution of probabilities over the actions (pure exploration) while a τ value close to 0 makes the algorithm behave more greedily (pure exploitation). Just as in the Epsilon-greedy algorithms, τ can be modelled to be decreasing over time just like how ε was being lowered over time with exploration annealing. And this annealing process applied to τ also gives the softmax exploration algorithm poly-logarithmic ($O(\log(T) + \dots + \log^n(T))$) regret bounds as in the epsilon-greedy case.

But just as in the epsilon-greedy case, theoretical regret bounds often don't lead to significant gains in performance in practical cases when the parameter T is known in advance, and when multiple rounds of experimentation on the environment can be performed to fine tune a constant τ value [18].

2.3.4 Optimism With Respect to Uncertainty

A quick overarching improvement that can be made to the softmax exploration algorithm, the epsilon-greedy algorithms and many other algorithms is that of estimating yet unexplored or not-well explored actions highly.

The fact of saying “think highly of actions who are estimated to be at a lower performance than the others if you haven't yet explored them enough” might appear contradictory to the theme of adaptive exploration who

aims to explore further the actions who are estimated at higher values and explore less the actions which are estimated to be of poorer quality. In a sense, this remark is correct because we are trying to perform trade-offs between exploration and exploitation to maximize returns (minimize regret). But it is a bit more subtle than that because the notion of uncertainty is added to the equation.

The way we were evaluating and comparing different actions was comparing our estimates of how fit we thought they were in expectation (their $\tilde{\mu}_a(t)$). But optimism with respect to uncertainty says that we should rather evaluate options based on our estimates of how fit they potentially could be instead. Meaning that the uncertainty we have about how well an action might be in the best case scenario should be taken into consideration when deciding which actions to go for.

An artificial way of introducing this optimism towards exploration in the epsilon-greedy and softmax exploration functions is to add an additional term $\frac{\mu_{\text{init}}}{T_a(t)}$ to the definition of $\tilde{\mu}_a(t)$, such that:

$$\tilde{\mu}_a(t) = \frac{\mu_{\text{init}} + \sum_{s=1}^t \mathbb{I}\{A_s = a\} r_s}{T_a(t)}$$

It has been shown in [19], [20] that in the cases where $T \gg K$, higher and higher values of μ_{init} yield better expected cumulative rewards. In fact, using μ_{init} values that accurately represent the correct expected rewards of the different actions yields subpar performance when compared to higher μ_{init} values.

In fact, even if we had access to the distribution $\mathcal{D}$ from which the rewards distributions $D_a(r)$ are independently sampled at each restart of the game, and developed an $\mu_{\text{init}}^* = \mathbb{E}_{\mathcal{D}} \max_i \mu_i$ for $i = 1 \dots K$, to be used as μ_{init} , epsilon-greedy would still perform worse than with slightly higher values of μ_{init} .

This is because of how “cheap” exploration in the beginning of the game should be considered as opposed to exploitation. But there definitely is a balance to strike when choosing an optimal μ_{init} .

The next algorithms implement uncertainty measures into their decision making process explicitly and do not use such artificial constructs to further exploration. [19]

2.3.5 Upper-Confidence Bounds (UCB)

The UCB algorithms implement an optimism with respect to uncertainty into the equation by building an uncertainty model. This model helps quantify how much the actual expected reward of an action might deviate from the current estimate. By considering the action with the highest estimated potential (i.e. highest upper-confidence bound) to be the next action to take. UCB achieves a better form of exploration than epsilon-greedy especially if the estimated reward distribution variance is taken into account [21]. The next step explores the actions with high degrees of uncertainty, yielding more information than a sample from an action where all the sampled rewards appear to be clumped up together with less variance. It is also likely that the chosen action is among the best in terms of expected reward as the upper-bound is a sum of expected reward estimator and an uncertainty estimator.

The most widely used form of UCB is the UCB1 algorithm, which defines uncertainty $v_a(t)$ for action a at time t as:

$$v_a(t) = \sqrt{c \frac{\log(t)}{T_a(t)}}$$

The parameter c is a constant parameter that can be tuned to make the algorithm more exploratory. UCB1 has been shown to have regret $R(T) \leq O(\log T)$ [9], [22]. However, the theoretical regret bound does not guarantee strong performance in all practical applications. As it was shown that UCB1 performs worse than other algorithms when the number of actions K increases and the level of variance in the rewards obtained from samples of the same action decreases [18].

There are other UCB algorithms that follow similar, potential based reasoning for evaluating actions. Another UCB is the successive elimination algorithm which explores actions until it has certainty that an action will perform poorer in its best case scenario than some other action in its worst case scenario (i.e. lower-confidence bound). In this case, the poor action is discarded from the pool of actions and will never be considered again.

Another variation of the UCB1 algorithm is the UCB-v algorithm which actually takes into account the variance of the rewards obtained from sampled rewards, using this to build a better Upper-Confidence model. It has been shown to perform better than UCB1 in practical experimentations [9].

2.3.6 Thomson-Sampling (Posterior-Sampling):

This algorithm can be considered as a state-of-the-art algorithm, as it is used by a number of huge tech giants to optimize some of their processes. Netflix uses Thomson-Sampling (TS) in their recommendation systems , Amazon uses TS to evaluate and optimize their site layout, Facebook uses a modified version of TS to optimize the quality of videos in the video uploading process [23].

TS is a random Bayesian algorithm. In TS, we want to build distributions for the expected rewards $\mu(a)$ for each action a . The more information we have from more and more samples of rewards from a given action, the more these distributions are going to concentrate around the actual values of $\mu(a)$. These distributions thus innately model the uncertainty that was mentioned in the UCB algorithms [24].

Suppose the underlying rewards distributions of the actions follow a Bernoulli (θ chance to get a reward of 1 and 0 otherwise) distribution $D_i \sim \text{Bernoulli}(\theta_i)$.

Playing the action with D_i reward distribution will have a cumulative reward y_i distribution that follows a Binomial distribution $\text{Binomial}(T_i(t), \theta_i)$.

Now let us suppose that the distribution over θ_i (which corresponds to μ_i) is a Beta distribution $\text{Beta}(\alpha, \beta)$. This leads to θ_i following a Beta posterior distribution (Bayes). This is because the Beta distribution happens to be the Conjugate prior of the Binomial distribution [25].

Formally:

$$\begin{aligned} P(\theta_i | Y_i) &\propto P(y_i | \theta_i) P(\theta_i) \\ &= \binom{T_i}{y_i} \theta_i^{y_i} (1 - \theta_i)^{T_i - y_i} \frac{1}{B(\alpha, \beta)} \theta_i^{\alpha-1} (1 - \theta_i)^{\beta-1} \\ &\propto \text{Beta}(y_i + \alpha, T_i - y_i + \beta) \end{aligned}$$

So this means that if the sample from D_i yielded a reward, the first parameter of the Beta distribution is increased by 1. And if the reward sampled was 0, we should increment the second parameter. Doing so for all actions and keeping track of the α and β coefficients for each of them allows us to have a distribution that is an estimate of μ_i but one that takes uncertainty into account [16].

The next step is to decide which action to take. In TS, we perform a random sampling from all the estimate distributions we have for each of the actions and then pick the action that got the highest sample. Thus, TS is not a deterministic algorithm. However, it tends to prioritize actions with higher estimated means, while the variance of these distributions, representing the inherent uncertainty about the true values of the expected rewards, also influences the decision-making process through random sampling [24].

For other distributions of D_a , other families of distributions functions can be used according to whether or not they are Conjugate priors to the D_a distributions. Rewards sampled based on a Normal distribution give rise to Normal distribution cumulative rewards whose Conjugate priors also happen to be Normal distributions [25].

TS achieves expected regret bounds of $\leq O(\log(T))$ which is known to be optimal for the problem at hand just like UCB1 [9].

2.4 Problem Statement

Having seen the MAB problem and some algorithms, the expert advice MAB can now be defined. The definition of the expert advice MAB has been tailored to our particular case, the general definition is slightly different.

The problem consists of a group of T patients. Each patient is either healthy or unhealthy making it a binary classification problem. It is our job to accurately classify these patients based on the information provided by experts, referred to as the advice or vote of each expert. There are K medical experts, each of whom gives their advice on the patient’s health when consulted. For each patient t (in order, from 1 to T), we may decide the set of experts to consult for determining that patient’s health. The goal is to find an effective strategy for deciding which experts to consult at each turn (i.e. for each patient).

- Each of the T patients’ health condition is represented by a real number $h \in [-1; 1]$ following a uniform distribution. The patients whose health condition h is negative are considered to be in poor health and require medical treatment (unhealthy). The remaining patients are classified as healthy. The distribution is balanced such that the number of healthy and unhealthy patients is equal.
- The advice provided by the experts about the patients is given in the form of a binary vote. At the end of each turn, the algorithm must assign a classification to the patient.
- The experts are modeled as stochastic estimators of the patient’s health condition with a bias and variance. The modelling and analysis of these experts are explained in Section 4.1. Additionally, each expert is assigned a cost, representing the reward that must be paid to access the expert’s vote.
- The three parameters defining an expert —bias, variance, and cost— are assigned at the start and are stationary. Only the cost is visible to the agent.
- The reward signal received by the agent depends on the correctness of the final classification, as well as the cost incurred from the expert consultancies.

The usual interpretation of the expert advice MAB is that of a contextual MAB with paid observations. In this interpretation, the arms are the two classes of the binary classification. The experts serve as tools where reward could be exchanged to potentially increase the classification accuracy by providing better context, which may lead to higher rewards. This interpretation of the expert advice MAB focuses more on the two classes and how the context (expert advice) could be used to increase classification accuracy.

Another interpretation of the expert advice MAB is to consider 2^K arms, where each arm corresponds to a unique combination of experts to consult. A strategy for combining the expert votes should be implemented for the MAB. In this interpretation, the underlying reward distributions of the arms are not entirely independent. Two similar groups of experts, with many coinciding experts, are likely to perform the same classifications. This interpretation can be useful if the action-space of the problem is modeled as an appropriate hypercube graph with the appropriate connections.

The problem can also be interpreted as a combinatorial bandit, meaning that there are many machines, and the agent, at each turn, needs to decide what to play on each of those machines. The machines in this interpretation would correspond to experts and the two arms available on each machine correspond to the decision of consulting that expert or not. The action-space Section 2.5 for this interpretation of the problem is thus $\{0, 1\}^K$. A strategy for combining the expert votes should be defined implemented separately in this interpretation.

The agent may choose to consult all experts, but this strategy is dubious due to consultancy costs exploding.

2.5 The General RL Problem

Reinforcement learning (RL) has been used in the last decade to train agents that could solve a wide variety of tasks. It has been used by Google to train AlphaGo and AlphaGo zero [26], who attained superhuman performance in the game of Go. The same OpenAI team also trained agents capable of beating human players in the games of StarCraft 2 as well as Dota 2, even though these last two agents were imposed limitations to ensure fairness for human competitors, who have reaction time and Actions Per Minute (APM) constraints. RL has also been used to teach agents how to manipulate robotic arms for the object grasping task. In recent years, researchers from DeepMind leveraged RL, to discover a faster way to multiply two matrices than the best available method we have today . Additionally, RL has been used to advance self-driving car technology [27].

So in short, RL has proven to be powerful in a range of tasks, each presenting their own complexity. It can manage large state-action spaces. It is able to learn given a model of the environment as well as through experience in cases where such a model is absent. Some of the tasks mentioned above demonstrate RL’s ability to handle imperfect information situations. The examples also demonstrate RL’s ability to make long term plans.

The goal of this chapter is to have an understanding of the fundamental concepts of RL as well as to better understand the calculations used to train the agents. It will closely follow [19] but will not be a comprehensive guide.

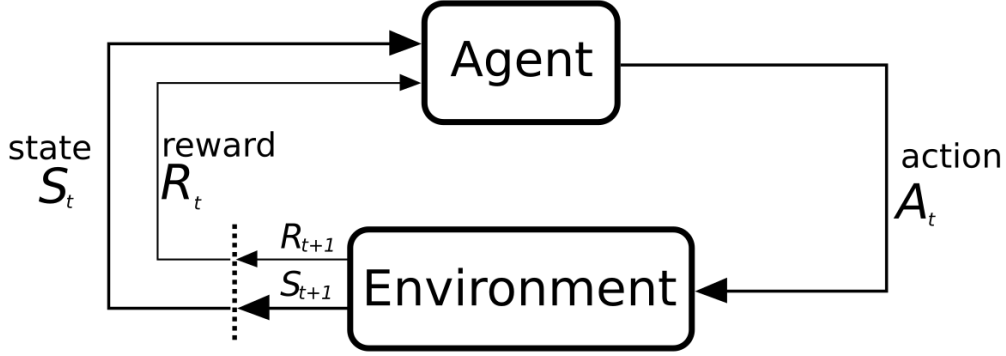


Figure 1: The general RL problem

The setting of RL consists of an agent —often referred to as the policy, actor, algorithm, or player— takes actions within an environment. The actions the agent takes are based on the current state of the environment, which it receives from the environment. These actions influence the environment, putting it in a potentially new state who is then passed onto the agent, forming a cycle.

The agent also has goals to accomplish within the environment. These goals are defined by a reward signal provided by the environment which the agent uses during training. The goal of RL is to train agents that collect high amounts of reward over the course of their play. The word “collect” indicates that what interests us is not the maximization of reward over a single step but over a consecutive sequence of steps that defines the entire game. The sequence of states, actions and the rewards collected from start to finish is henceforth called an episode (game, play).

In general, the state of an environment s at time t is said to come from a finite state-space $\mathcal{S}$. Such that $s \in \mathcal{S}$.

As for the actions the agent is able to take, these depend on the state of the environment. The set of all valid actions depending on the state also belong to a finite set $a \in \mathcal{A}(s)$. A detail to keep in mind is that the terminal state (the state which announces the end of an episode) is not included in $\mathcal{S}$ but only in the larger set $\mathcal{S}^+$. This is because no actions or rewards may occur after the terminal state. It is thus sufficient to limit our study to $\mathcal{S}$.

It is important to note that the states, actions and the rewards can be stochastic signals. Therefore, capital letters are used to indicate the random variables, while their corresponding samples are represented with lowercase letters.

And lastly, the reward signal r has been limited to be a real valued signal. $r \in \mathcal{R} \subseteq \mathbb{R}$

2.5.1 The Environment

The environment is defined with a probability distribution function p . This p , also called the model of the environment determines the state transitions as well as the rewards.

$$p(s', r \mid s, a) = \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$$

The above equation is very important because it states that the next state of the environment as well as the reward signal depend only on the current state of the environment and the action taken by the agent. This hypothesis is called the Markov decision process (MDP) hypothesis and is essential for the theoretical convergence of many RL algorithms to optimality.

2.5.2 The Agent

The agent in the environment behaves according to a policy. The policy π determines the agent’s action probabilities depending on the state. The policy is written as $\pi(s)$ in cases where it is deterministic, or as $\pi(a|s)$ when it is stochastic.

The stochastic aspect of the agent's policy in RL is essential for exploration. In the absence of the environment model p , stochastic policies can be used to drive the exploration of the state-action space. Additionally, stochasticity can be beneficial in certain environments, such as in poker or rock-paper-scissors, where it is important to act unpredictably against adapting opponents.

2.5.3 The Reward Signal

As stated earlier, the goal is to implement an agent to find a policy π that gathers high cumulative rewards. This idea is captured by the return random variable G_t .

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots + \gamma^{T-t-1} R_T = \sum_{k=t+1}^T \gamma^{k-t-1} R_k$$

Where the γ , $0 \leq \gamma \leq 1$ is the discounting factor, emphasizes the relative importance of immediate rewards compared to future ones. The T variable indicates the final step before the episode terminates. In episodic tasks, such as tic-tac-toe, the episode is expected to conclude after a finite number of steps. However, in continuous tasks, such as controlling the speed of a motor with a low-level controller, the episode might never end $T = \infty$. In the latter case, the addition of this γ factor may be essential to the convergence of G_t . Only Episodic tasks are studied here.

Our goal is to find a policy that maximizes the expected return, referred to as the optimal policy π_* .

$$\pi_* = \max_{\pi} \mathbb{E}[G_t]$$

2.5.4 An example

An example of a MDP is an agent navigating a grid world labyrinth to find the exit. In this situation, the state of the environment is the agent's position, and the actions are the set of movements up, down, left or right who do not lead the agent into a wall.

We aim to design a reward signal that allows the agent to solve the labyrinth. In the un-discounted case ($\gamma = 1$), a naive approach might be to give the agent a reward of 1 whenever it walks into the exit square and 0 otherwise.

However, this approach does not work because all episodes that end in the terminal state (the exit square) get a return of 1. This reward signal doesn't allow to distinguish between policies that competently walk towards the exit and the policies who stumble onto the exit by chance after wandering for a long time.

A better reward signal would be to give the agent a constant reward of -1 for each step spent inside the labyrinth. This reward signal establishes the necessary hierarchy between policies to define the task. The optimal policy under this reward signal would walk the shortest path from start to finish.

Notice how the two different reward signals only differ from each other by 1 reward at each step. But they do not result in the same behavior. This is because the behavior of the agent actually played a role in determining the number of steps within each episode.

The second reward signal was equal to the first reward signal shifted down by 1. This made a difference in the induced return signal because episode lengths may vary. In Section 5.1, a reward signal for our expert advice MAB is defined, but since the episode lengths are constant in our problem, the baseline level of the reward is not important. Additionally, this baseline level is irrelevant because the algorithm used to train the policy takes measures to make the baseline level of the reward irrelevant (Section 2.12).

2.5.5 The Hierarchy of Policies

2.5.6 The Bellman-Ford Equations

As demonstrated previously, given an environment, certain policies perform better than others. To express this concept, the following random variable is defined:

$$V_{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right]$$

This expression represents the expected return following policy π and starting from state s . The expected return depends on the current state, and $V_{\pi}(s)$ is called the state value function for the given policy π .

A related idea to the state value function is the action value function $q_\pi(s, a)$, also known as the state-action value function. It is a random variable depicting the expected return G_t achieved by starting from state s , and taking action a , and then following policy π thereafter.

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

It should be noted that episodic tasks can be considered as a subcase of continuous tasks, except that they are guaranteed to reach a terminal state, which loops infinitely upon itself with certainty and yields a reward of 0. This allows for the use of the same notation to handle both continuous and episodic tasks.

The state value function can be trivially expressed in terms of the action value function as follows:

$$V_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

And by using the identity that $\mathbb{E}[G_t] = \mathbb{E}[R_t + \gamma G_{t+1}]$, the action value function can be expressed in terms of the state value function as follows:

$$q_\pi(s, a) = \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')]$$

Thus, the two functions are strongly related, and knowing one is sufficient to find the other.

The double application of the above equations yields the two Bellman-Ford equations:

$$\begin{aligned} V_\pi(s) &= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_\pi(s')] \\ q_\pi(s, a) &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \right] \end{aligned}$$

The value functions satisfy affine equations for any given policy, with as many equations as there are states. Similarly, there are an equal number of affine action value equations corresponding to each state-action pair. Moreover, there being as many equations as variables and the equations being affine means that the value functions are the unique solutions to their respective Bellman-ford equations.

2.5.7 The Optimal Policy

The optimal policy yields optimal returns when followed from any given state.

$$\begin{aligned} \max_{\pi} (v_\pi(s)) &= v_{\pi_*}(s) = v_*(s) \\ \max_{\pi} (q_\pi(s, a)) &= q_{\pi_*}(s, a) = q_*(s, a) \end{aligned}$$

The optimal value functions $v_*(s)$ and $q_*(s, a)$ are of interest because they enable the creation of the optimal policy. The two value functions can be expressed in terms of the other with the Bellman optimality equations. The quantity $q_*(s, a)$ can be trivially expressed as:

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

The state value function might be calculated by considering the policy that is greedy over the action value function in that state.

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

Double application of these equations yields the Bellman optimality equations

$$\begin{aligned} v_*(s) &= \max_a \sum_{s', r} p(s', r \mid s, a) [r + \gamma v_*(s')] \\ q_*(s, a) &= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \end{aligned}$$

RL algorithms can use these equations in their search for optimal policies.

2.6 Generalized Policy Iteration

The main idea behind Generalized Policy Iteration (GPI) is to iteratively transition from a sub-optimal policy to an optimal one using the Bellman-ford equations. This process involves two key steps: Policy evaluation, which is the calculation of the state or action value function for a given policy, and Policy improvement, which is the making of a policy greedy with respect to a given state or action value function.

This approach of evaluating a given policy by obtaining the corresponding state or action value function, and then using that evaluation to develop a better policy is at the heart of almost all RL algorithms. But what distinguishes GPI is some of the strong assumptions made with regard to the problem.

1. The Markov property:

We assume that the environment is a MDP where the reward and next state depend only on the current state and the action taken from that state. While this process can be stochastic, the underlying distributions rely solely on these two variables and are independent of all the past actions and states.

The Markov property is satisfied when the agent is able to observe the state, meaning no relevant information is omitted. This ensures that the agent is interacting directly with the MDP.

2. Perfect information over the underlying pdf p of the MDP:

The underlying $p(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)$ is used extensively in the Bellman-Ford equations to evaluate policies. In Section 2.9, it is shown how optimal policies can be learned without knowledge of p .

3. All the states and actions can be listed:

This is just to say that the underlying state and action spaces of the problem are finite, and ideally small. This assumption is essential for the convergence of GPI methods. In Section 2.11, we see how good policies can be found even if the state and action spaces are large or enumerable.

Under such strong conditions, which are difficult to fulfill for most real world problems, the results achieved are also robust strong. However, it's important to note that the violation of these conditions leads to weaker RL algorithms in practice than what the theoretical predictions for the optimal scenario.

2.7 Policy Evaluation

As mentioned earlier, policy evaluation is the act of extracting the state or action value function of a given policy. This will be done by using the Bellman-Ford equations as an assignment rule iteratively, in order to converge to the actual state values of the problem. The policy to evaluate is going to be $\pi(a|s)$ and the corresponding state value function is $v_\pi(s)$.

Start by arbitrarily assigning values to all the states, denoted as $V_0(s)$. These values having been arbitrarily assigned and likely do not satisfy the Bellman-Ford equations for the policy π , meaning they aren't the correct values. Nonetheless, update the arbitrary values as follows with the Bellman equations:

$$V_{k+1}(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) [r + \gamma V_k(s')]$$

This process of updating the faulty values with the previous iteration is guaranteed to converge in the limit ($k \rightarrow \infty$) and the only fixed point of the Bellman-Ford equations of the given policy being those of v_π means that true state value function can be found in this manner.

A simple improvement to this method would be to, instead of relying solely on the values from the k^{th} step to calculate the $(k+1)^{\text{th}}$ step, incorporate some of the updated values from the $(k+1)^{\text{th}}$ step during the iteration. This approach has been shown to converge faster in practical applications while still guaranteeing to converge to the same state value function.

In practice, it is impossible to iterate a process indefinitely in order to find the exact limit values. Therefore, the process is stopped once the iterations change the values by a smaller amount than a predefined accuracy parameter. This ensures that the final values $V(s) \approx v_\pi(s)$.

A similar process can be done by using the action value function $Q_0(s, a)$ to obtain $Q(s, a) \approx q_\pi(s)$ with the following iteration.

$$Q_{k+1}(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \gamma \sum_{a'} \pi(a' | s') Q_k(s', a') \right]$$

2.8 Policy Improvement

Given the state or action value function $v_\pi(s)$ or $q_\pi(s, a)$ of a policy $\pi(a|s)$, the goal is to develop a new policy $\pi'(s)$ that performs better than the initial policy. This is achieved by updating the policy so that, in each state, it chooses the action with the highest expected return, assuming the initial policy π will be followed thereafter.

$$\begin{aligned} \pi'(s) &= \operatorname{argmax}_a q_\pi(s, a) \\ &= \operatorname{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \end{aligned}$$

There is a proof in [19] that shows policy improvement always yields a better policy $\pi'(s)$ than the policy used to generate the value function $\pi(s)$. The new policy $\pi'(s)$ is strictly better than $\pi(s)$ unless $\pi(s)$ was an optimal policy, meaning $\pi(s) = \pi_*(s)$.

2.8.1 GPI Methods

GPI methods intertwine policy evaluation and policy iteration to refine policies until convergence is achieved. Convergence is guaranteed because the state and action spaces are finite, meaning that there are finitely many deterministic policies that may exist for this environment.

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

This is how the simplest of GPI algorithms, known as policy iteration, operates, as elegantly demonstrated in Figure 2:

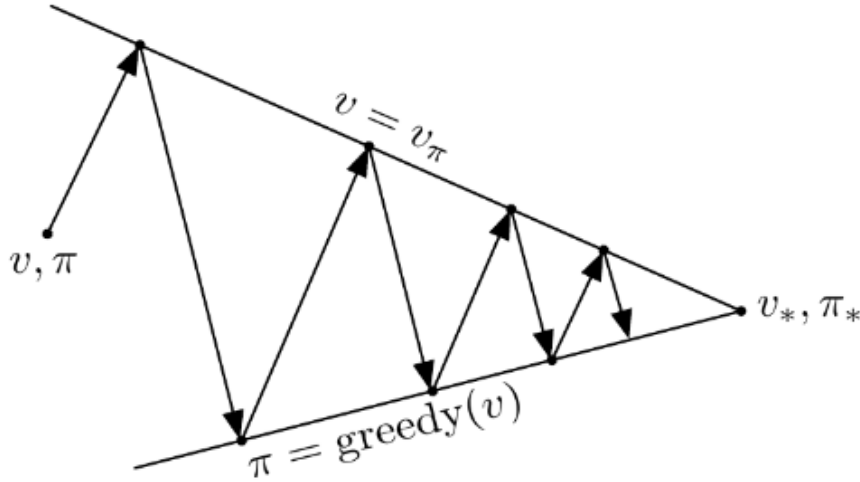


Figure 2: The convergence process of policy iteration [19]

Other GPI methods attempt to introduce shortcuts to to accelerate convergence to optimal policies in most practical cases, although this comes at the cost of losing guaranteed convergence. These methods are beyond the scope of this thesis.

2.9 Monte-Carlo Methods TD(∞)

The goal of Monte-Carlo (MC) methods is to find optimal policies without relying on the underlying model $p(s', r | s, a)$. Instead, the agent learns through experience, using trial and error.

As with GPI methods, the goal is to perform successive iterations of policy evaluation and policy improvement to move closer to the optimal policy. The focus here is on modifications to policy evaluation as policy improvement remains mostly unchanged.

The primary change in policy improvement is that the state value function can no longer be used, as this would require knowledge of p .

$$\operatorname{argmax}_a q_\pi(s, a) = \operatorname{argmax}_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s)]$$

We are thus limited to the use of $q_\pi(s, a)$.

2.9.1 MC Policy Evaluation

The idea is to hold an estimate $Q(s, a)$ of the true value of an action $q_\pi(s, a)$. This estimate is obtained by averaging the samples of the return G_t where the state-action pair s, a was achieved. By the law of large numbers, this estimate converges to $\mathbb{E}_\pi[G_t | S_t = s, A_t = a]$.

MC methods primarily differ in how they perform the trial and error exploration and can be categorized into three main types.

2.9.2 Exploration Starts

Below is an algorithm from [19] that performs both policy evaluation and improvement using exploration starts to find q_* . his approach ensures that all the state-action pairs are visited infinitely many times in the limit, which is necessary for convergence to the true values of q_π . The episodes are started at random states, following random actions, after which the the policy is continues to explore according to its $\pi(a|s)$ values from there on.

```

1. Initialize:
     $\pi(s) \in \mathcal{A}(s)$  arbitrarily  $\forall s \in \mathcal{S}$ 
     $Q(s, a)$  arbitrarily  $\forall s \in \mathcal{S}$  and  $\forall a \in \mathcal{A}(s)$ 
     $N(s, a)$  as all 0s  $\forall s \in \mathcal{S}$  and  $\forall a \in \mathcal{A}(s)$ 

1. Loop forever (for each episode):
    Choose  $S_0 \in \mathcal{S}, A_0 \in \mathcal{A}(S_0)$  randomly s.t. all pairs have probability  $> 0$ 
    Generate an episode according to  $\pi$ :  $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \dots, S_{T-1}, A_{T-1}, R_T$ 
     $G \leftarrow 0$ 
    Loop for each step  $t$  of episode,  $t = T - 1, T - 2, \dots, 1, 0$  :
         $G \leftarrow \gamma G + R_{t+1}$ 
        Unless  $S_t, A_t$  appears in  $S_0, A_0, S_1, A_1, S_2, A_2, \dots, S_{t-1}, A_{t-1}$ :
             $N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$ 
             $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G - Q(S_t, A_t))$ 
         $\pi(S_t) \leftarrow \operatorname{argmax}_a Q(S_t, a)$ 

```

2.9.3 On-Policy Methods

On-policy methods are similar to the exploration start methods discussed earlier, with the key difference being that instead of randomly generating an initial state and action for the beginning of an episode—which may not always be feasible—the agent is kept ε -soft. This means the agent always has a non-zero chance of taking a random action at any given state. As a result, in the limit, it explores all state-action pairs sufficiently to ensure that $Q(s, a)$ converge to q^* .

2.9.4 Off-Policy Methods

Off-policy methods aim to decouple the policy being evaluated from the policy used to generate the training data. This is especially useful when the training must be conducted on a pre-sampled dataset of trajectories, and no new data can be generated.

The principle is similar to that of on-policy methods, but the update rules must be adjusted to account for the fact that the exploration policy does not have the same action probabilities as the policy being evaluated.

2.10 Temporal Difference-Learning TD(n)

Although MC methods are sound in principle, they face slow learning problems (low sample efficiency) because they require waiting until the end of an episode before updating the value function. Instead, if the state, ac-

tion, reward triplets that immediately follow a state-action pair are considered relevant enough for an earlier update, learning can be accelerated by allowing policy evaluation to occur within an episode. This approach can potentially increase sampling efficiency [19].

The only change with respect to MC methods is in the policy evaluation step, where another target is used.

$$\text{MC} : Q(s_t^m, a_t^m) \leftarrow Q(s_t^m, a_t^m) + \alpha(g_t^m - Q(s_t^m, a_t^m))$$

$$\text{TD}(1) : Q(s_t^m, a_t^m) \leftarrow Q(s_t^m, a_t^m) + \alpha(r_{t+1}^m + \gamma Q(s_{t+1}^m, a_{t+1}^m) - Q(s_t^m, a_t^m))$$

Instead of waiting for the episode to conclude, TD methods use the current estimate of the value function as an educated guess about the episode return, allowing updates to occur before the episode ends. This technique is known as bootstrapping in the domain of RL. TD algorithms differ by how many steps n they are willing to wait before bootstrapping if the episode hasn't yet ended. This is why MC methods are referred to TD(∞), as they are willing to wait indefinitely for the episode to end and employ bootstrapping [19].

Another point of difference for TD algorithms is the choice of the target. The most well-known TD algorithm, called Q-learning, uses a greedy target in its update rule, aiming to accelerate learning even more than TD(1).

$$\text{Q-Learning} : Q(s_t^m, a_t^m) \leftarrow Q(s_t^m, a_t^m) + \alpha \left(r_{t+1}^m + \max_a Q(s_{t+1}^m, a) - Q(s_t^m, a_t^m) \right)$$

2.11 Function Approximation

2.11.1 State Innumerability Problem

So far, methods capable of learning from an environment in both model-based and model-free formats have been explored, with Generalized Policy Optimization being the central theme. A fundamental step in these methods involves storing the value functions $v(s)$ and $q(s, a)$ in a table for policy-making and bootstrapping. However, this approach isn't always practical due to real-world problems, where state action spaces can be extremely large or even uncountable, as in the case of continuous state variables like velocity or position. The naive approaches struggle to learn effectively in these environments due to memory constraints and the need for unreasonable amounts of data.

2.11.2 The Solution

To address the data-intensive nature of RL, the value function estimates are constrained to a restricted function class, allowing us to store them with a set of parameters w . If the continuity imposed by the new class function closely resembles the true value function, learning speed is drastically improved.

$$v_\pi(s) \approx \hat{v}_\pi(s, w), w \in \mathbb{R}^d$$

w corresponds to a set of parameters that define $\hat{v}_\pi$ and $\hat{v}_\pi(_, w)$ replaces the state value function.

Examples of such functions could be linear functions for feature vectors such as position and angular velocity for a pendulum $x, \dot{x}$. Leading to

$$\hat{v}_\pi(s, w) = x(s)^T w$$

Our goal now shifts to finding a $w \in \mathbb{R}^2$ such that the policy $\pi(a|s)$ obtained from greedily acting over the value function achieves a high level of reward. This approach can similarly be applied to the action value function q .

With the value function now defined by the vector w and because we limit ourselves to a class of value functions, the entire space of possible value functions is no longer attainable. As a result, altering the w vector is likely to change the value functions for many states rather than just one. Which is exactly the trade-off sought from the beginning in order to improve memory, data efficiency, and computation costs.

2.11.3 Policy Evaluation

Policy evaluation is challenging because the true value function $v_\pi(s)$ likely does not belong to the model class. However, this does not prevent the goal of obtaining a sufficiently close approximation. This notion of closeness is measured by the value error VE committed by the approximation $\hat{v}_\pi$.

$$\overline{\text{VE}}(w) = \sum_{s \in S} \mu_\pi(s) [v_\pi(s) - \hat{v}_\pi(s, w)]^2$$

$\mu(s)$ is a term that quantifies the idea of “not all mistakes committed by the value function are equally relevant”. It is less problematic if the value approximation deviates from the true value function for a rarely state, as long as it allows for better approximations in states that are encountered more frequently under the policy π . $\mu(s)$ is called the importance function [28], [19].

However, it is important to acknowledge that computing the value error $\overline{VE}$ is impractical in real-world scenarios, as it requires knowledge of the true state-value function v_π , which is the very function being estimated in policy evaluation, and necessitates to perform a summation over large state spaces.

2.11.4 Gradient Monte-Carlo

An example of a RL algorithm that performs updates to reduce the value error through stochastic gradient descent is the Gradient Monte-Carlo method, which uses the following update rule:

$$w \leftarrow w + \alpha[G_t - \hat{v}_\pi(S_t, w)]\nabla\hat{v}(S_t, w)$$

The name “Gradient MC” is fitting for this method because the target used in the update rule is the full return G_t . However, unlike regular MC methods, where there is no bootstrapping, Gradient MC must bootstrap its estimate of the value function to predict the gradient of the value error. If the model class is differentiable with respect to w , Gradient MC is guaranteed to converge to a local minima of the value error for sufficiently small α .

2.11.5 Model Free Gradient-Descent

One must keep in mind however, because policy improvement cannot be done without a model p of the environment with the state value function alone, it is once again necessary to consider working with the action value function instead.

$$w \leftarrow w + \alpha[U_t - \hat{q}_\pi(S_t, A_t, w)]\nabla\hat{q}(S_t, A_t, w)$$

The update rule of model free gradient-descent is similar to that of gradient MC except that any target U_t can be used instead of the full return. The target used in this update is what differentiates the RL algorithms in function approximation. Q-learning and other TD learning targets could be used as the U_t to make RL algorithms.

2.12 Policy-Gradient Methods

The Reinforce algorithm described in this section is an on-policy method. It has one major difference when compared to the algorithms seen in Section 2.11, namely that it forgoes trying to figure out value functions but rather models the policy $\pi(a|s, w)$ directly. This policy gives us the probabilities of taking any action as a function of the state and a parameter vector w . All that is required of the policy function is that the action probabilities given by the policy be differentiable with respect to the parameters, allowing the computation of a gradient [19].

The Reinforce algorithm strongly resembles MC algorithms, it samples episodes from the environment and then uses the return obtained at the end of an episode to perform an update, not on state or action values, but on the policy directly (by updating its parameters). Technically, an update is performed for every action the policy took during that episode. By performing this process across multiple episodes with the appropriate update rule, the aim is to develop effective policies.

A preliminary assumption regarding the appropriate update rule could be as follows (resembling the update procedure employed for function approximation techniques):

$$w \leftarrow w + \alpha\gamma^t g_t^m \nabla\pi(a_t^m | s_t^m, w)$$

For each episode m this update rule is applied to each time step, starting from 0 up until the end of the episode. This update rule is logical because the higher the return g_t^m obtained after following action a_t^m in state s_t^m , the more likely it is for this action to be taken on that state after the update. The policy is updated in a direction that locally increases action’s probability based on the return. The higher the return, the larger the step in that direction.

If there are multiple actions in a given state, with some yielding more expected returns than others, the update rule will more aggressively adjust w to favor these actions. Conversely, in the case of negative returns, this update rule will make the actions less likely to be taken. Herein lies the issue with this update rule. As discussed in earlier chapters, adding a constant amount of reward to all actions in every state should not alter the optimal

policy for an environment with fixed-length episodes (as is the case for our Expert-Advice MAB). However, the convergence properties of this update rule are heavily influenced by the average reward level (baseline), as demonstrated by the one-state, one-step environment illustrated in Figure 3, a basic example of a 2-armed MAB.

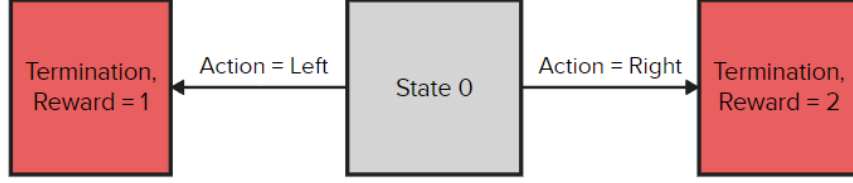


Figure 3: Left, Right environment

For the 1 step environment described above, the return is just the reward for that single step. Now suppose we currently have that $\pi(\text{Left} | s, w) = 80\%$ and $\pi(\text{Right} | s, w) = 20\%$ for a given policy π .

As episodes are sampled and the update rule is applied, episodes ending in the Right action will increase the probability of the policy selecting the Right action in future iterations. Similarly, episodes terminating in the left state after taking the Left action will increase the probability of the policy selecting the Left action. Notably, the updates corresponding to the Right action will be twice as aggressive as those for the Left action due to the return being twice as large, just as desired. However, because the Left action is taken 4 times as often as that of the Right action, the overall direction in which the policy will move with this update rule above is towards the Left action being more likely to be taken. So the update rule of Reinforce takes this phenomenon into account

$$w \leftarrow w + \alpha \gamma^t g_t^m \frac{\nabla \pi(a_t^m | s_t^m, w)}{\pi(a_t^m | s_t^m, w)}$$

or more often written as:

$$w \leftarrow w + \alpha \gamma^t g_t^m \nabla \log \pi(a_t^m | s_t^m, w)$$

Where the additional quotient makes the updates oriented towards higher reward actions on average. And makes the update rule at any given state, less dependent on the baseline of that state.

The full first visit Reinforce algorithm is thus:

Reinforce [19]

1. Initialize:
 - Functional $\pi(a|s, w)$ of predefined function class
 - Initial parameters $w \in \mathbb{R}^d$
 - γ, α
2. Loop forever (Or for a given number of episodes):
 - Generate an episode according to $\pi(a|s, w)$: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$
 - $G \leftarrow 0$
 - Loop for each step t of episode, $t = T - 1, T - 2, \dots, 1, 0$:
 - $G \leftarrow \gamma G + R_{t+1}$
 - Unless S_t, A_t appears in $S_0, A_0, S_1, A_1, S_2, A_2, \dots, S_{t-1}, A_{t-1}$:
 - $w \leftarrow w + \alpha \gamma^t G \nabla \log \pi(a_t^m | s_t^m, w)$

The convergence properties of Reinforce were thus made independent of the baseline reward level by adding that additional term however, Reinforce still suffers from slower learning times when the baseline reward level is too high or too low. To demonstrate this, we analyze the environment in Figure 4:

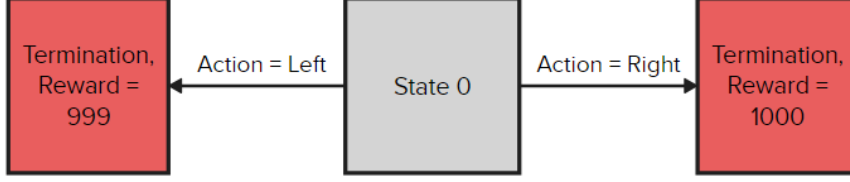


Figure 4: Left, Right environment with increased baseline

The update steps corresponding to samples where the policy took the Left and Right actions will be much larger than those for the environment in Figure 3 (for the same α). Despite these two environments being identical from an optimal policy perspective, this dependency exists. Therefore, an enhancement through function approximation can be applied to Reinforce to facilitate faster learning, albeit at the expense of increased computational resources.

Reinforce with baseline

1. Initialize:

Functional $\pi(a|s, w)$ of predefined function class

Initial parameters $w \in \mathbb{R}^{d_w}$

Functional $\hat{v}(s, \theta)$ of predefined function class

Initial parameters $w \in \mathbb{R}^{d_\theta}$

$\gamma, \alpha_w, \alpha_\theta$

2. Loop forever (Or for a given number of episodes):

Generate an episode according to $\pi(a|s, w)$: $S_0, A_0, R_1, S_1, A_1, R_2, \dots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step t of episode, $t = T - 1, T - 2, \dots, 1, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Target = $\delta \leftarrow G - \hat{v}(s_t, \theta)$

Unless S_t, A_t appears in $S_0, A_0, S_1, A_1, S_2, A_2, \dots, S_{t-1}, A_{t-1}$:

$$w \leftarrow w + \alpha_w \gamma^t \delta \nabla \log \pi(a_t^m | s_t^m, w)$$

$$\theta \leftarrow \theta + \alpha_\theta \delta \nabla \log \pi(a_t^m | s_t^m, w)$$

In this approach, both an estimation of the state value function and an optimal policy are learned. The value function ensures that the target (used in the update rule) remains independent of the mean value of the rewards for that state. Essentially, the value of each action for a given state is determined by how much higher or lower it is than the average return the policy expects to achieve from that state. Since it is crucial to have an accurate state value function before refining the policy, typically $\alpha_w < \alpha_\theta$. The $\hat{v}$ function used above is technically not the state value function but the baseline function, which is essentially the same as the state value function but offset by a constant that represents the baseline of that state.

All of the RL agents used in this thesis have enough similarities to Reinforce with Baselines to be considered essentially the same. They utilize different estimators for finding the same gradient direction as Reinforce. And as long as their update rule aligns with a gradient that has an expected value equivalent to that of Reinforce, they exhibit good convergence properties. This is because the update rule of Reinforce is equal in expected value to the direction which increases the state value of a policy, the steepest. Its update rule is derived from the Policy-Gradient Theorem. [19]

The expert advice MAB algorithms employed in this thesis are all based on the Proximal Policy Optimization (PPO) method, which belongs to the class of policy gradient methods. Composed of one function representing

the policy, and another called the advantage $A(s, a) = Q(s, a) - V(s)$. Advantage measures how good an action is compared to the baseline value of that state and serves the same purpose as the baseline [29].

One key feature of PPO is the clipping of policy update steps. If an update is too large, it is clipped, thereby limiting the extent to which a single update can alter the policy. This clipping mechanism ensures that the policy does not deviate significantly from the region where the baseline function accurately approximates the policy’s value function, thereby enhancing the stability of the learning process.

The policy and advantage functions are modeled using neural networks, which serve as universal function approximators. These networks process input observations through layers of linear combinations followed by non-linear activation functions to generate outputs. The internal parameters of these neural networks are adjusted according to the PPO update rule to improve the policy [29].

3 State of the Art

3.1 Recent Advances In RL

When using RL to solve any problem, it is important to supply the RL agent with appropriate observations. Recent studies have shown that providing the same input data to RL agent albeit in different representations can impact performance.

The impact of observation representations on performance is illustrated in a study that trained a pathfinding agent on a 2D plane. The agent received observations of the plane in different bases, such as Fourier, polynomial, and others [30]. The study found that agents trained with different representations exhibited varying levels of performance.

Further illustrating the importance of state representations for RL agents, the study “Graph-based State Representation for Deep Reinforcement Learning” compares the performance of RL agents tasked with navigating in a 2D grid-world. A grid-world essentially functions as a graph where nodes represent positions and edges indicate possible movements. This research investigates the efficacy of representing the grid-world environment both in matrix form and through four distinct graph embeddings. The findings reveal that agents trained using any of the graph-based representations outperform those trained with the matrix representation, despite an increase in the dimensionality of the observation space [31].

Other recent DRL studies have implemented techniques to solve one of the major limitations of RL-based methods, that of sample efficiency. These techniques are particularly relevant in scenarios where the decision-making process must consider a series of objects of the same type, such as multiple particles, agents, or financial instruments. In such situations, the underlying symmetry of the problem—specifically permutation symmetry—demands that the performance of the algorithm should not depend on the ordering of the input data [32], [33].

Two innovative approaches to addressing this challenge involve the use of specialized neural network architectures that respect the aforementioned symmetries. The first study introduces an attention-based mechanism that projects neural network inputs into an efficient representation space, which remains invariant regardless of changes in input ordering. This method significantly reduces the dimensionality of the input space and enhances the agent’s ability to handle variable numbers of objects, leading to marked improvements in sample efficiency across various tasks [32].

Conversely, the second work explores the use of an equivariant neural network architecture, akin to convolutional layers used in image processing. This permutation-invariant layer is designed to effectively encode and utilize the symmetry present in domains involving multiple interacting entities. By learning to predict the dynamics of interacting hard discs in 2D, this approach not only mimics the generalization capabilities of convolutional layers to different image sizes but also extends it to varying numbers of objects in the context of reinforcement learning [33].

These two studies are particularly relevant to our expert advice MAB problem, where agents are tasked with using a coalition of experts to classify patients. Each expert, viewed as an object, shares several characteristics with others, such as cost, bias, and variance. Given that the experts’ parameters are initialized independently of their identities, the expert advice MAB problem exhibits symmetries similar to those addressed in [32], namely that of the observations containing objects of the same type. These parallels are further elaborated in Section 5.4, where the thesis also introduces a novel approach. This approach constrains the agent from observing equivalent permutations of the same data, effectively reducing the observation space by a factor of $K!$, thereby improving sample efficiency.

3.2 Existing Methods

There are many application problems closely resemble our expert advice MAB problem, and there are existing solutions to the question of which experts to consult at each iteration. These solutions are often extensions of algorithms used in the MAB framework. One such algorithm is the Normalized Exponentiated Gradient (EG) [34]. For a coalition of K experts, this algorithm specifies how often w_i each expert i should be consulted at time t according to a softmax distribution given by:

$$w_i = \frac{e^{-\eta g_{1:t,i}}}{\sum_{j=1}^K e^{-\eta g_{1:t,j}}}$$

The term $g_{1:t,i}$ represents the loss suffered by expert i up until time t . The higher the loss an expert suffers, the less likely they are to be consulted at the next iteration. η is a parameter determining how greedy the algorithm is. EG is an extension of the softmax algorithm because, in the context of an MAB problem, setting $g_{1:t,i}$ as the negative of the current estimated reward of an arm effectively reproduces the softmax algorithm.

The nuance in this algorithm lies in the definition of $g_{1:t,i}$, which can be adapted to suit the needs of the problem. If the experts are non-stationary, an exponentially weighted moving average can be a good strategy, as it places more emphasis on loss terms that occurred at a later time compared to earlier ones. However, since the experts in our problem are stationary, a different, more suitable solution can be applied.

The paper [35] presents a framework that closely resembles ours. In their study, a coalition of experts can be consulted to help them choose the arm to be played in an MAB problem. The experts in their study give estimations for the expected reward of each arm of the MAB. In our context, this MAB corresponds to the binary classification of each patient. The full algorithm is outlined in the box below:

Prediction with limited advice:

Input η_t the greediness parameter and M_t the expert budget $\forall t \in \{1, \dots, T\}$.

$$g_{1:1,i} = 0, \forall i \in \{1, \dots, K\}$$

For each $t = 1, 2, \dots$ do:

$$w_i = \frac{e^{-\eta g_{1:t,i}}}{\sum_{j=1}^K e^{-\eta g_{1:t,j}}}$$

sample M_t experts according to the distribution w without replacement to consult.

Play the advice of one of the experts consulted chosen according to the distribution.

Observe the loss $g_{t,i}$ suffered by each consulted expert.

Update $g_{1:t,i}$ for all the experts i that were consulted as:

$$g_{1:t,i} = g_{1:t-1,i} + \frac{g_{t,i}}{w_i + (1 - w_i) * \frac{M_t - 1}{K - 1}}$$

The losses accumulated by each expert enable the algorithm to develop a preference for better-performing experts over the course of an episode. This learning is not to be interpreted as the learning of an RL algorithm, but rather that of a MAB algorithm.

The update rule in the prediction with limited advice algorithm is more aggressive when a rarely consulted expert performs well compared to when a frequently consulted expert does. This approach is similar to the rationale behind the update rule in Reinforce algorithm. The prediction with limited advice serves as our baseline comparison for evaluating RL-based techniques.

4 Material and Methods

4.1 The Patients and the Experts

The expert advice MAB we construct involves a series of experts and patients. The health of each patient in this problem is generated uniformly over the interval $[-1 : 1]$.

$$\forall j \in \{1, \dots, N\}, h_j \sim \text{Uniform}([-1, 1])$$

Where h_j is the innate health level of patient j , and they are sampled independently from the interval.

The patients corresponding to positive numbers are called healthy and the negative ones unhealthy. The patients are thus categorized with respect to the sign of their h .

$$\forall j, H_j = \text{sign}(h_j)$$

Where H_j is the correct classification regarding the health of the patient.

The experts are binary classifiers that attempt to predict the health level H_j for each patient. Each expert $i \in \{1, \dots, K\}$ is characterized by their bias μ_i and variance σ_i^2 , which are the variables used to calculate their prediction regarding the patient's health. These two parameters are sufficient to fully describe our experts.

The classification of each expert is stochastic. For a given h_j , expert i classifies the patient in the following manner:

$$\begin{aligned} \hat{H}_i(h_j) &\sim \text{sign}(h_j + \mu_i + V_i) \\ V_i &\sim \text{Uniform}\left(\left[-\frac{\sigma_i}{\sqrt{3}}, +\frac{\sigma_i}{\sqrt{3}}\right]\right) \end{aligned}$$

Such that $\text{Var}[V_i] = \sigma_i^2$.

This process models the imperfect nature of expert classifications. The bias shifts the expert's classification threshold, while the variance introduces randomness to the classifications around the threshold. Figure 5 illustrates intuitively how these experts differ in their classifications, depending on their bias and variance parameters.

- Figure 5 Top left: This illustration shows the misclassification of an expert who is fully biased to always classify patients as healthy (bias $\mu = 1$). This expert correctly classifies healthy patients but consistently misclassifies unhealthy patients. Since the two classes are balanced, the misclassification rate is 50%, as indicated by the area under the curve.
- Figure 5 Top right: This illustration shows the misclassification rate of an unbiased expert who still has misclassifications due to the stochastic nature of the classification process. Patients who are very ill and or very healthy are correctly classified most of the time. The zone between the two is where most misclassifications occur. The experts are practically incapable of classifying patients whose health condition hovers close to 0.
- Figure 5 Bottom left: This illustration depicts the misclassifications for a biased expert. This expert is biased to classify patients as ill. A detail to point out is that because the variance of the expert is small enough compared to its bias, this variance can never overcome the bias of the expert, essentially making it so that the expert never fails to identify an unhealthy patient when he sees one.
- Figure 5 Bottom right: This illustration demonstrates that when an expert's variance is large enough, it can occasionally override their bias. This expert tends to make fewer mistakes when classifying a patient as ill rather than healthy. However, a significant number of ill patients are still misclassified as healthy due to the high variance.

Misclassification Probability of Experts [Bias, Variance]

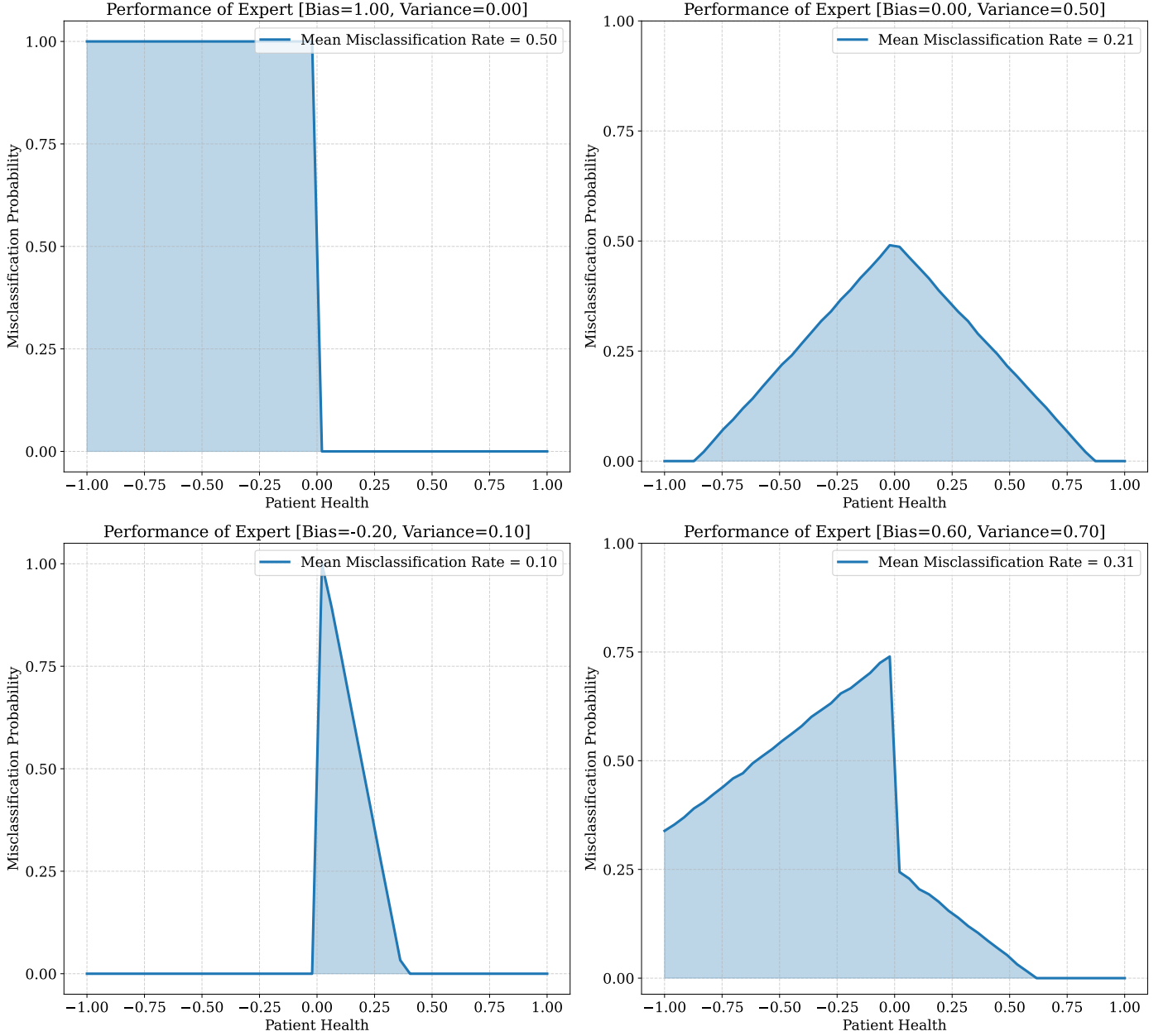


Figure 5: The misclassification rate of various experts depending on the patient health. The experts differ in their bias and variance. The area under each curve represents the average misclassification rate (the lower, the better).

It is important to notice how the experts in this problem are correlated with one another. This is easy to see with an extreme example:

Consider two experts i, i' who are fully deterministic ($V_i = V_{i'} = 0$). However, expert i is more biased than expert i' to classify patients as healthy ($\mu_i > \mu_{i'}$).

In the scenario where expert i' classifies patient j with health h_j as healthy ($h_j + \mu_{i'} > 0$), it is guaranteed that expert i will perform the same classification ($h_j + \mu_i > 0$). The same holds true when expert i classifies a patient as unhealthy.

The correlation between the experts means that the advice of an additional expert doesn't always yield more information and increase classification rate.

4.2 The Need for A Forecaster

In the context of the expert advice MAB defined earlier, the agent is responsible not only for deciding of which experts to consult, but also for determining the final classification to assign to the patient based on the expert advice. To simplify the learning process for RL agents and enable them to develop an effective policy for handling the expert advice MAB, it is useful to split these tasks apart. Specifically, RL should focus solely on deciding which experts to consult for each patient (in each round or turn), rather than also combining the expert advice into a final classification. It is necessary, therefore, to have a method for combining the advice provided by the experts into a single classification. The forecaster is responsible for aggregating the advice from the M consulted experts out of the K available to produce the final classification.

4.3 Weighted Majority Forecasters

The forecasters considered are functions of the form $F : \{-1, 1\}^M \times W^M \rightarrow \{-1, 1\}$. Where the $\{-1, 1\}^M$ represents classifications provided by the M consulted experts, and the output $\{-1, 1\}$ is the final classification determined by the forecaster. The space W^M contains information related to the experts, W can contain values such as such as their accuracy and sensitivity. In our forecasters, W is taken to represent the confusion matrix, which includes the true/false $\times$ positive/negative rates for each expert and helps assess the weight of each expert's opinion.

In total, there are 2^{2^M} possible mappings from $\{-1, 1\}^M$ to $\{-1, 1\}$, making it impractical to find the optimal one. Therefore, we restrict ourselves to a voting system with multiplicative voting weights assigned to each expert.

The forecaster used is a weighted majority voting of the form:

$$F(v, w) = \text{sign} \left(\sum_{i=1}^M \text{weight}(v_i, w_i) \right)$$

$$\forall v = \begin{bmatrix} v_1 \\ \vdots \\ v_M \end{bmatrix} \in \{-1, 1\}^M, \forall w = \begin{bmatrix} w_1 \\ \vdots \\ w_M \end{bmatrix} \in W^M$$

The idea here is to assign greater importance to the classifications made by more competent experts through the use of a $\text{weight}(v, w)$ function, or confidence function.

Taking $\text{weight}(v_i, w_i) = v_i$ would assign equal weights to all experts.

4.3.1 Confidence Matrix Weight Function

In this context, we define a weight function that assigns a weight to each vote cast by an expert based on the expert's performance, specifically how often their classifications are correct. The weight function is given by the following equation:

$$\text{weight}(v_i, w_i) = \begin{cases} \frac{p(\hat{H}_i=1|H=1) - p(\hat{H}_i=1|H=-1)}{p(\hat{H}_i=1|H=1) + p(\hat{H}_i=1|H=-1)} & \text{if } v_i = 1 \\ \frac{p(\hat{H}_i=-1|H=1) - p(\hat{H}_i=-1|H=-1)}{p(\hat{H}_i=-1|H=1) + p(\hat{H}_i=-1|H=-1)} & \text{if } v_i = -1 \end{cases}$$

where the H represents the true class of the patient and the $\hat{H}_i$ is the classification made by expert i . The equation is easier to read as as:

$$\text{weight}(v_i, w_i) = \begin{cases} \frac{\text{TP} - \text{FP}}{\text{TP} + \text{FP}} & \text{if } v_i = 1 \\ \frac{\text{FN} - \text{TN}}{\text{TN} + \text{FN}} & \text{if } v_i = -1 \end{cases}$$

How this function behaves for experts with different levels of bias and variance can be seen on Figure 6.

This function is completely symmetrical for a reversal of the confusion matrix in a binary classification task. Therefore, if an "evil" expert intentionally misclassifies patients, the reversal of its confusion matrix would cause the forecaster to distrust this expert as seen in Figure 7.

The final point to note is that experts are fully trusted on the first visit. This choice is motivated by the assumption that there are no evil experts in our coalition and all experts perform better than random chance.

Even though the case of a fully biased expert is statistically impossible, it is a common scenario for the experts in our MAB to be casting their first vote.

It can be observed in Figure 6 that for large bias values μ_i (in absolute value), and sufficiently small variance values, the stochastic part V_i in $\text{sign}(h_j + \mu_i + V_i)$ is unable to dominate μ_i . As a result, experts with large bias and low variance exhibit either a positive-predictive-value or a negative-predictive-value of 100%. Their votes are assigned the value of 1 or -1 depending on the classification sign.

The arbitrary choice of trusting an expert who has not been seen before can be justified by the assumption that, in general, experts perform better than random and are unlikely to be “evil.”

4.3.2 Entropy Reduction Weight Function

This function assigns weights to votes according to the entropy reduction induced by the vote on the patients.

$$\text{weight}(v_i, w_i) = \begin{cases} \text{sign}(\text{TP} - \text{FP}) \left(1 - \text{Entropy}\left(\frac{\text{TP}}{\text{TP} + \text{FP}}\right)\right) & \text{if } v_i = 1 \\ \text{sign}(\text{TN} - \text{FN}) \left(1 - \text{Entropy}\left(\frac{\text{TN}}{\text{TN} + \text{FN}}\right)\right) & \text{if } v_i = -1 \end{cases}$$

Figure 6 shows the profile of this function for different expert characteristics. Notably, this weight function is steeper than the confidence matrix weight function. It assigns greater weights to near-perfect votes and lower weights to poor votes when compared with the confidence matrix based weights.

Figure 7 again depicts the symmetry of this weight function with evil experts.

Vote Weight Values for Confidence and Entropy Forecasters

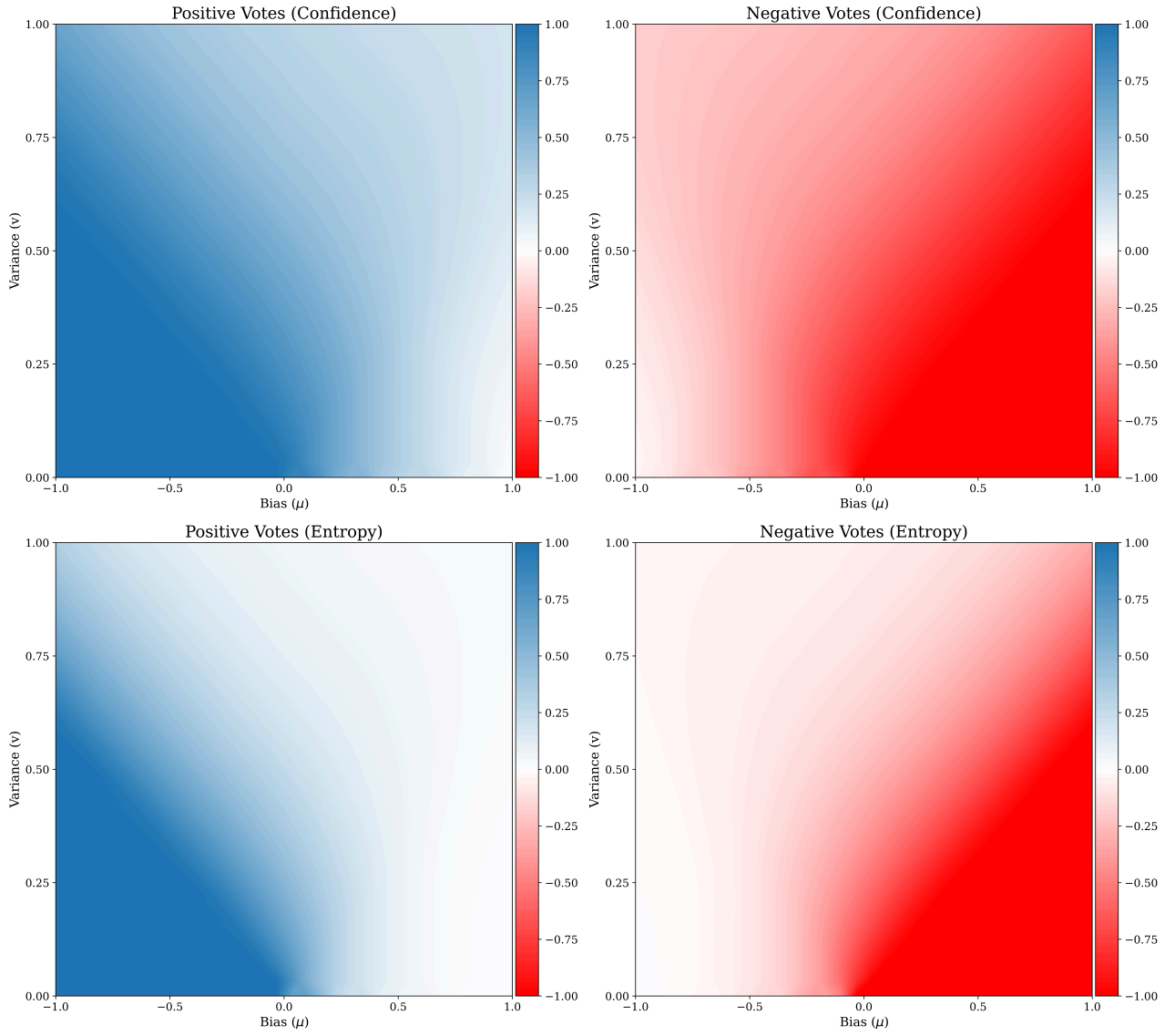


Figure 6: Confidence matrix (above) and Entropy (below) weight function values for positive and negative votes for experts of different bias and variance values.

Vote Weight Values for Confidence and Entropy Forecasters With Evil Experts

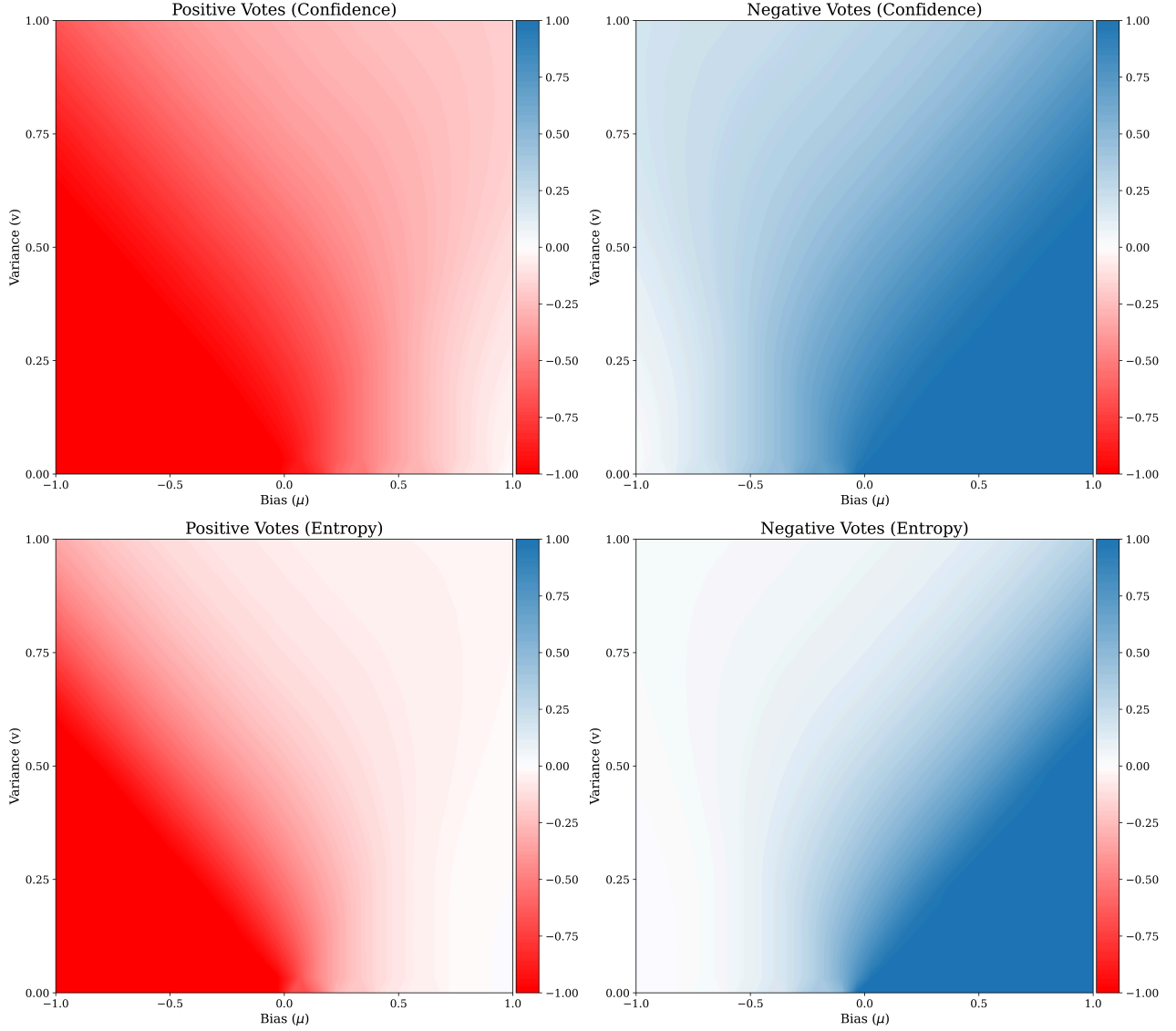


Figure 7: Forecaster weight values for positive and negative votes for “evil” experts of different bias and variance. These experts always classify the patients as the opposite to their true belief, resulting in a reversal of the confusion matrix.

4.3.3 Forecaster results:

Figure 8 shows how the votes of two experts are combined to make a final classification. Since one of the experts is better than the other at identifying patients, the vote of the better expert dominates the forecasting. Figure 9 shows how the votes of four experts are combined to achieve greater accuracy than the best individual expert in the coalition. The two weighing methods frequently agree with each other on how the votes should be combined, as seen in Figure 9.

Figure 10 shows an example where the $\{-1, 1\}^M \rightarrow \{-1, 1\}$ mapping produced by the confidence matrix-based coefficients is different from those of the entropy reduction-based coefficients.

Figure 11 is an example where a perfect expert with 100% classification accuracy is included in the coalition. The perfect expert has the most valuable vote in the crowd, but since this vote’s weight is bounded, and the other experts get non-zero weights, the perfect expert can be diminished within the crowd. If enough poor experts align in their mistakes, their summed votes can outweigh that of the perfect expert. Consequently, weighted majority votes don’t always perform better than the best individual expert within the coalition.

An analysis of how these forecasters perform overall is presented in Figure 12 where coalitions of different sizes are combined with weighted majority voting and compared to the average best expert of the coalition. We only study groups of experts no more than 8, as all the experiments performed in this thesis involve an 8 expert coalition.

Figure 13 shows what happens when the coalition of experts includes a perfect expert. The figure shows that the entropy-based forecaster is better at prioritizing the perfect expert's input compared to the confidence matrix-based forecaster. The reason for this is the relatively lower vote weights assigned to imperfect experts by the entropy forecaster compared to the confidence matrix forecaster. It is thus easier for the perfect expert to achieve a dictatorship when the votes are combined with the entropy forecaster.

Due to its greater ability to retain the influence of a highly accurate expert (Figure 13) and its overall superior performance (Figure 12), the entropy-based forecaster is the preferred choice for all the experiments that follow.

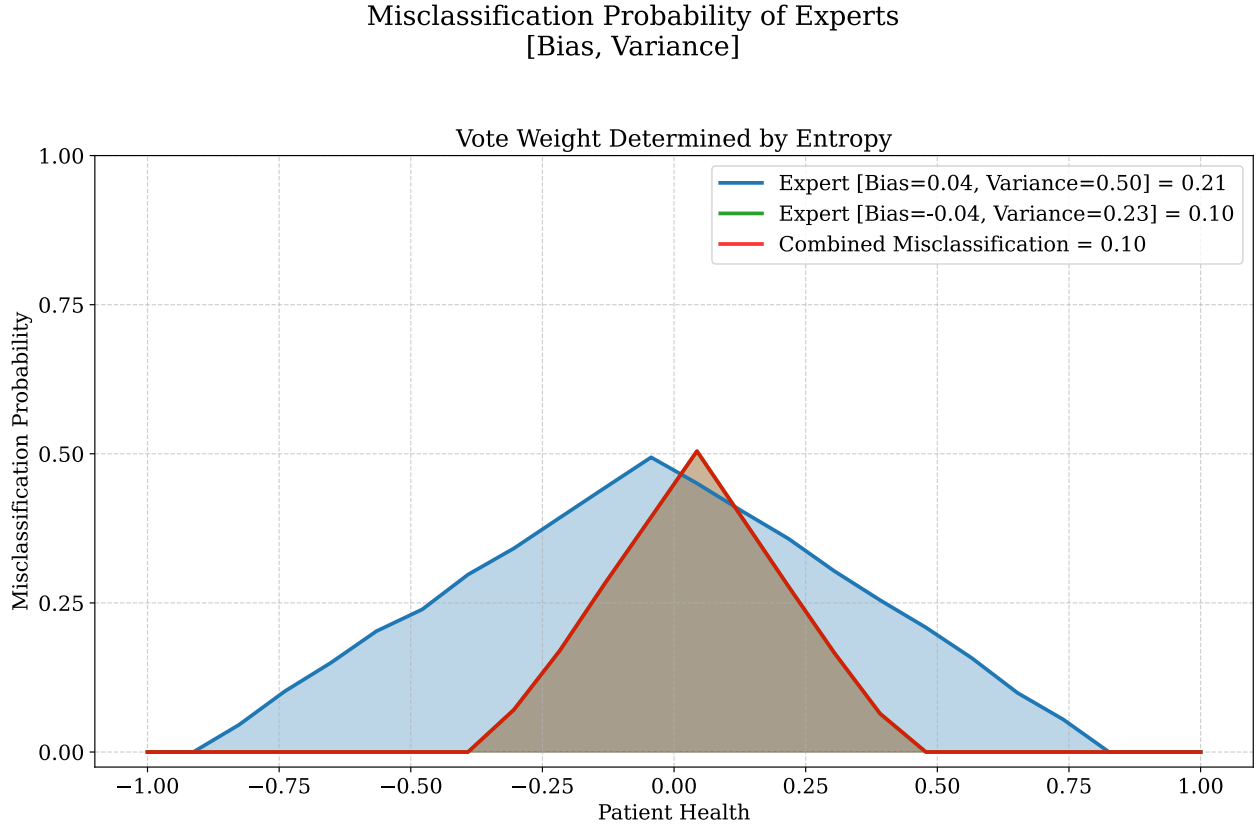


Figure 8: In the case of 2 experts, it often happens that one expert is better than the other for both its positive and negative votes. When this occurs, the vote of the other expert is discarded and only the more trusted expert's vote is kept. The smaller the red area, the better the outcome. In this example, the vote of the second expert (green) reigns supreme and the forecaster's final decision aligns to that of the second expert.

Misclassification Probability of Experts [Bias, Variance]

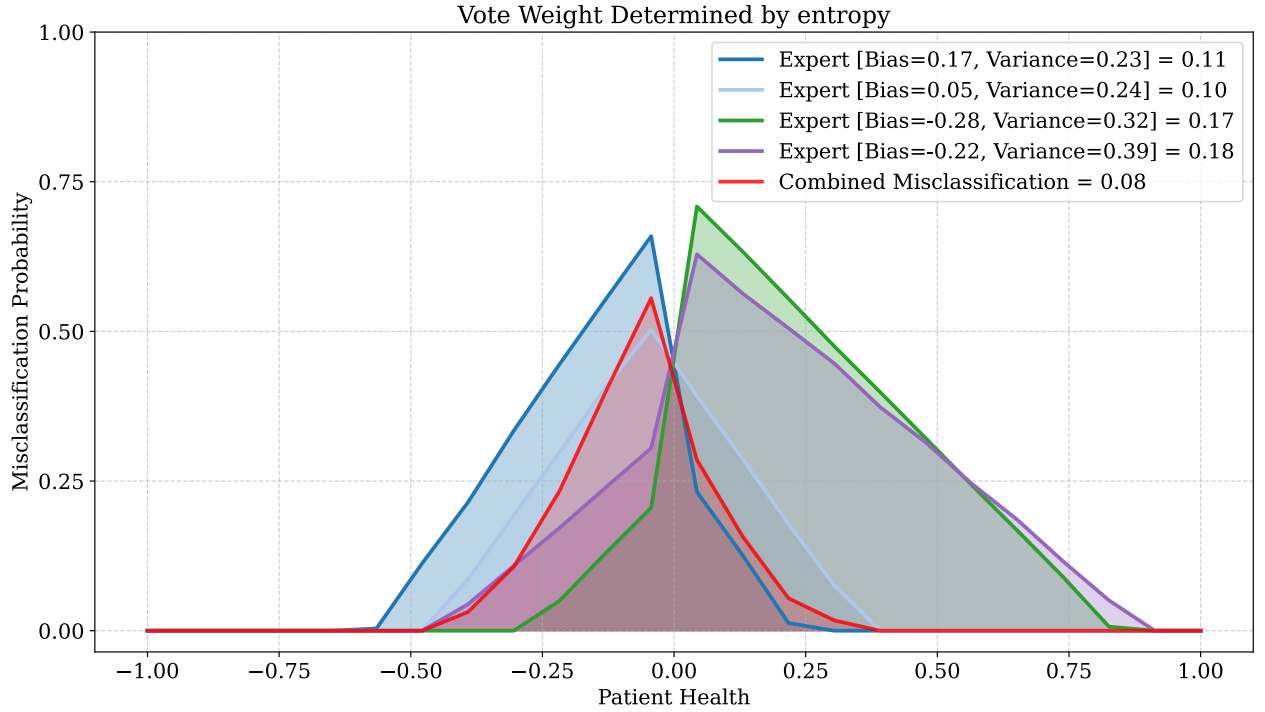
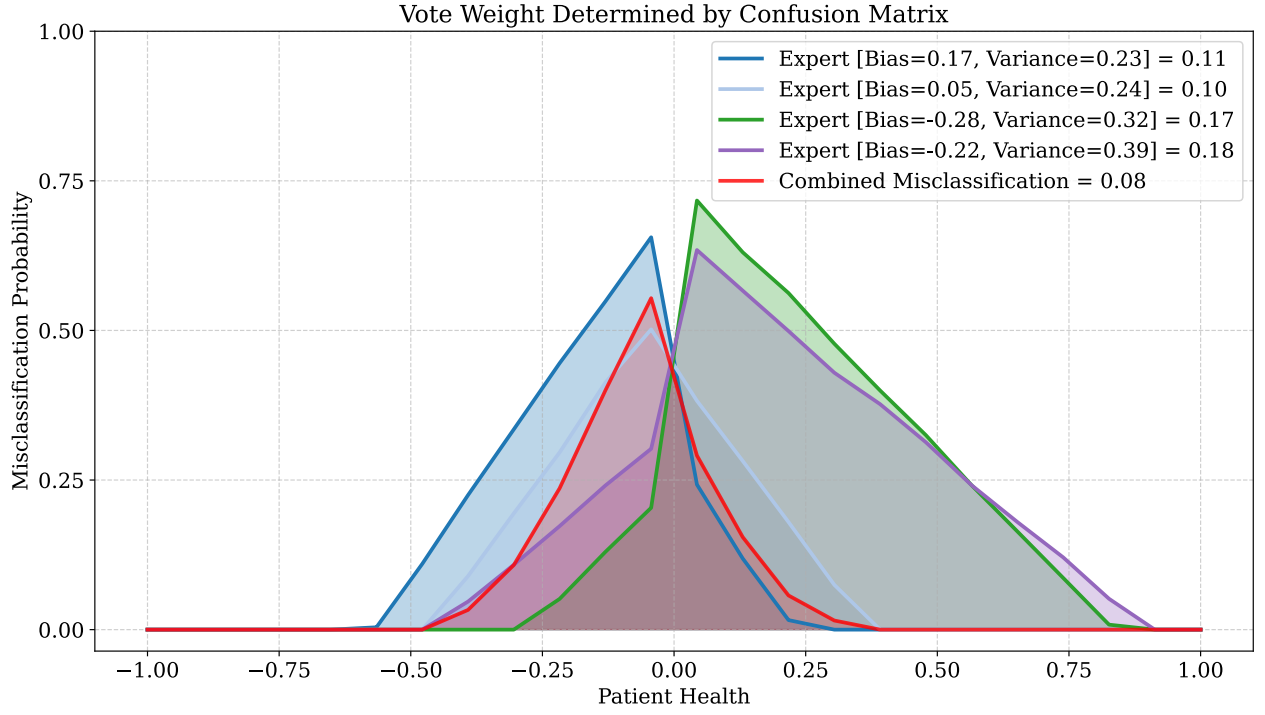


Figure 9: 4 Different experts performing weighted majority voting with the two methods.

The $\{-1, 1\}^M \rightarrow \{-1, 1\}$ mapping induced with the methods are the same, hence the majority votings share the same misclassification characteristics.

Misclassification Probability of Experts [Bias, Variance]

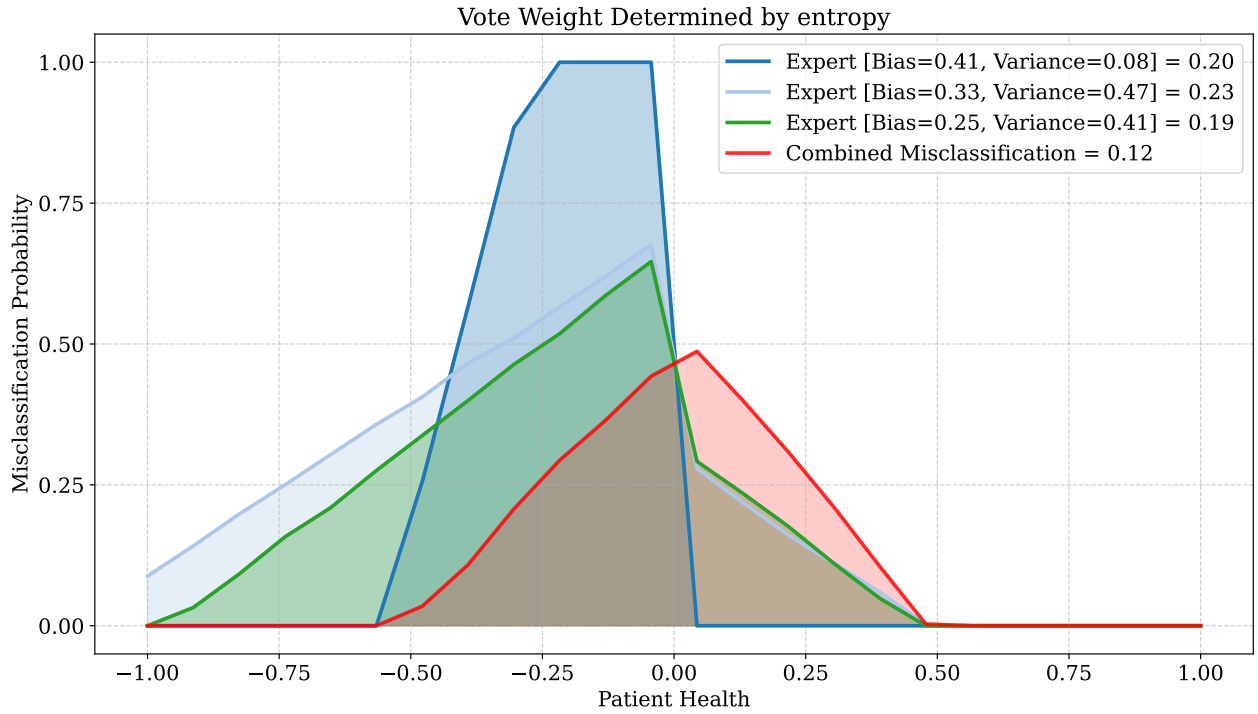
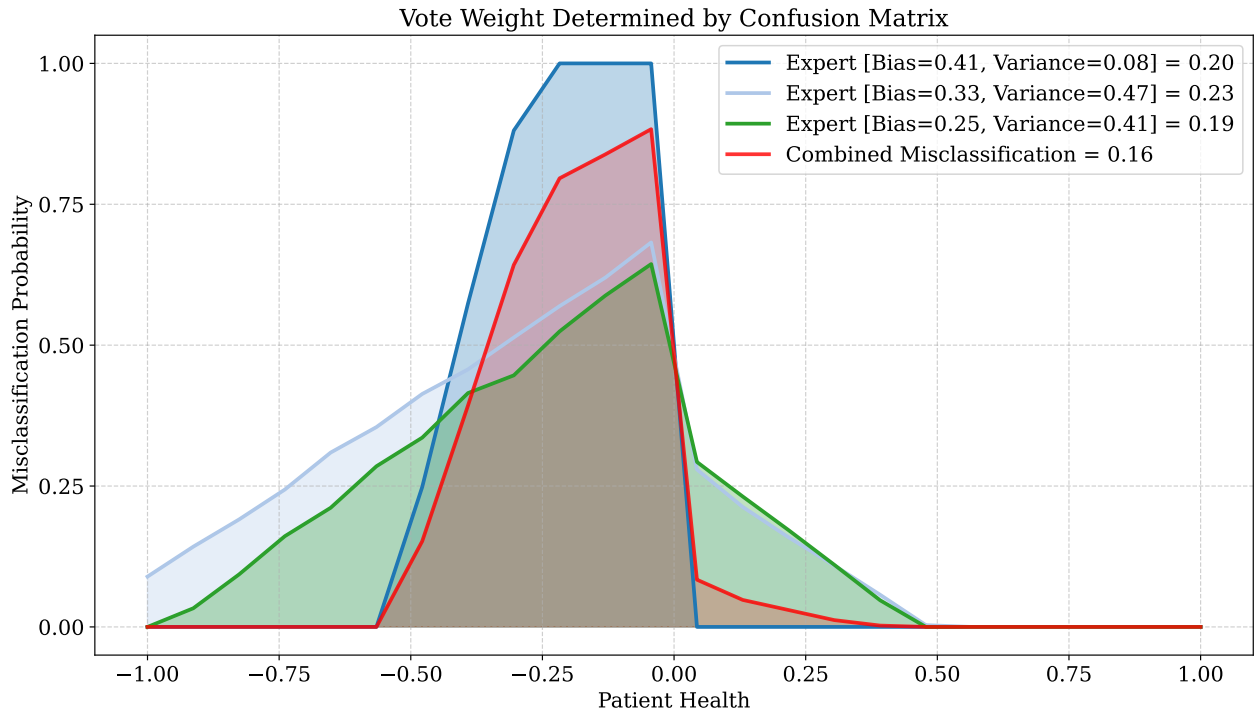


Figure 10: Example of disagreement between how the votes should be combined for the two different weighting techniques.

Misclassification Probability of Experts [Bias, Variance]

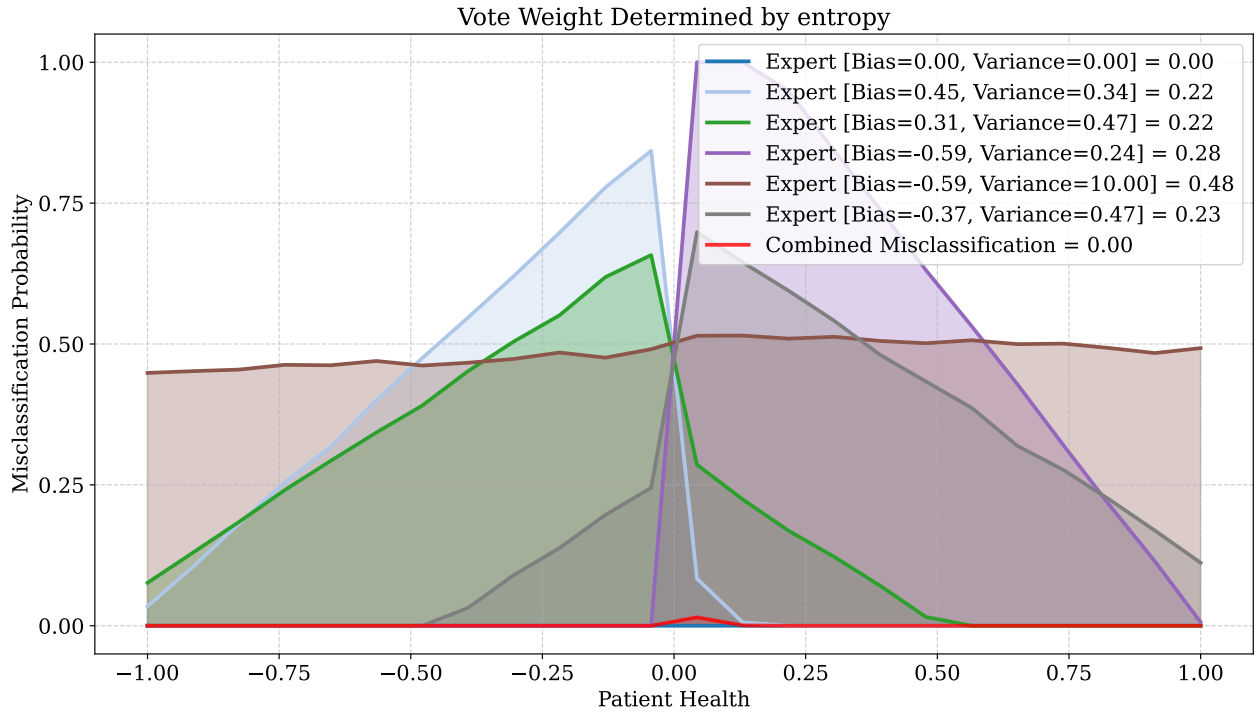
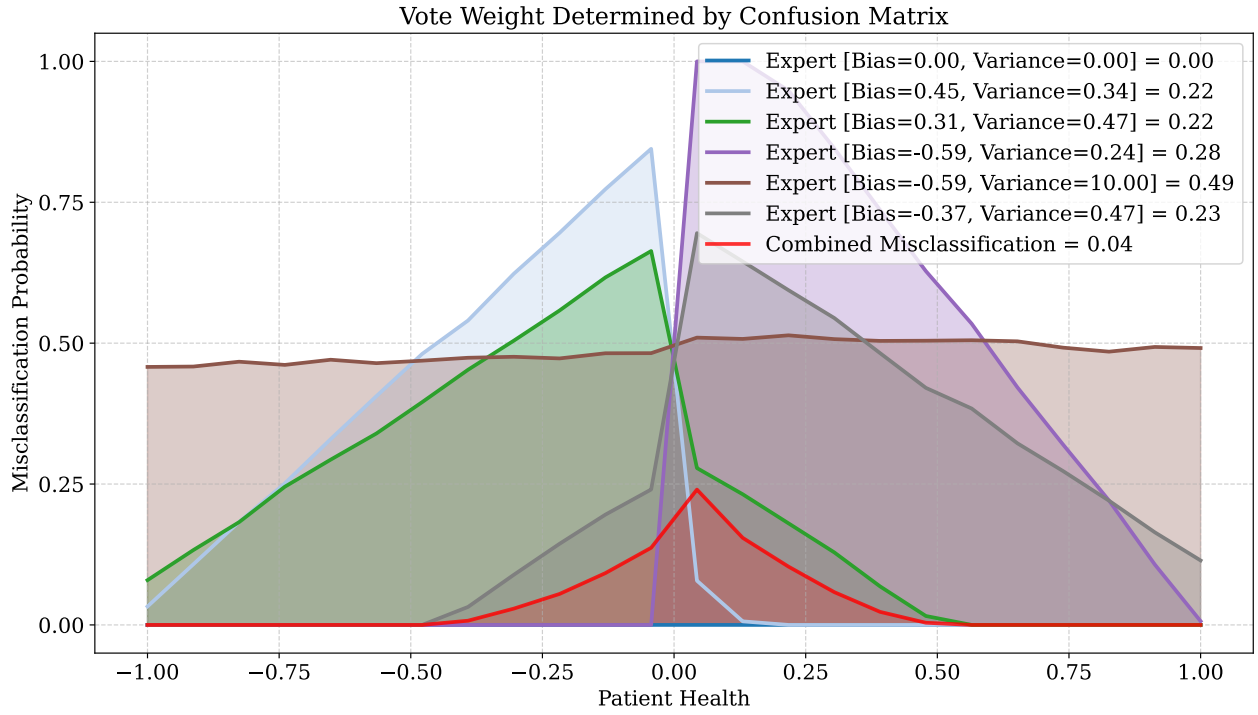


Figure 11: Weighted majority voting with a perfect expert placed within the coalition. The entropy based forecaster assigns a higher relative importance to the perfect expert than the confidence matrix forecaster.

For a quantitative evaluation of the two weight functions, we can examine how much majority voting improves the misclassification rate as the number of experts increases on average.

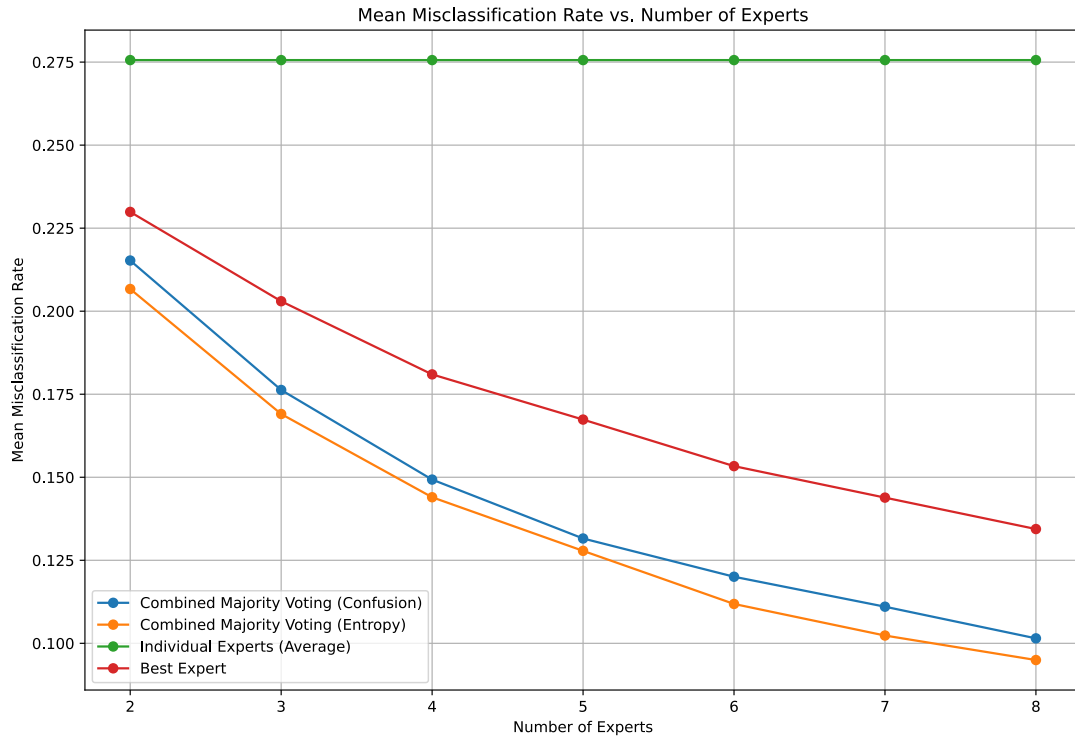


Figure 12: How the forecasters perform for varying number of experts compared to the average and the best experts of the bunch.

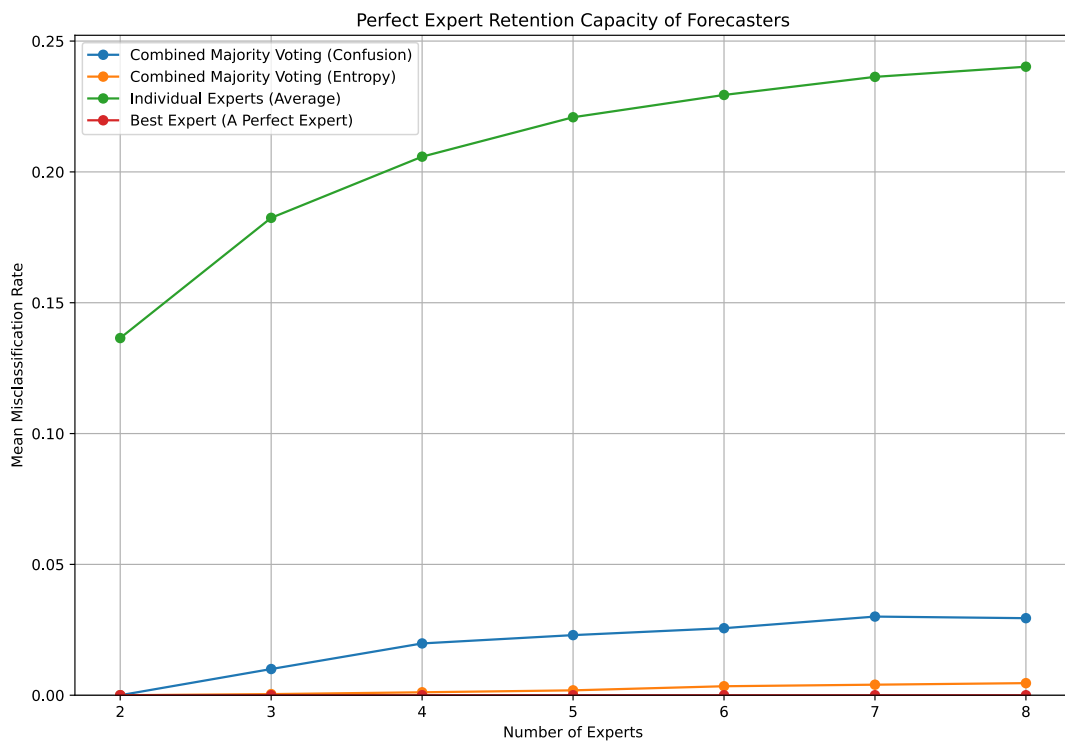


Figure 13: When one of the experts is a perfect expert, the majority vote degrades their performance by mixing it with the advice of the crowd.

5 Experiment Setup

The main experiment involves framing the expert-advice MAB as an RL problem. Depending on how the observations, and actions are defined, there are several ways to approach this. Three different approaches have been implemented, each varying primarily in terms of the actions involved. The main differences are briefly described here, with a more detailed description provided below.

All of the approaches use equivalent reward signals, namely the difference between the reward obtained for a correct classification and the costs of the experts consulted on that turn. This reward signal is called the **Std** (Standard) reward signal.

The RL agents for all of the approaches essentially share the same observation space. Specifically, they receive a matrix containing the relevant data for all experts, including their current forecaster weights as well as the confidence placed on these weights, the costs. Additionally, they receive inputs unrelated to the experts, such as the number of rounds (patients) are left for diagnosis.

Approach 1: At each step, the RL agent decides the collection of experts to consult for one given patient, and the level of confidence to place on each expert’s vote, all in a single action, hence one step.

The agents of Approach 1 do not receive any forecaster data in their observations since they do not use the forecaster. Instead, they receive the confidence matrix of the experts along with the associated costs.

Approach 2: At each step, the RL agent only decides the collection of experts to be consulted for a given patient. Each vote cast by the consulted experts is weighted according to the current estimated confidence level for each expert as calculated by the forecaster.

Approach 3: At each step, the RL agent decides whether or not to add a single expert to its current list of experts to consult for each patient. The full set of experts to be consulted for each patient is known only when the decision has been taken for every expert. The votes are weighted according to the forecaster.

Due to Approach 3 having to perform multiple steps to make its decisions for a single patient, the same Std reward needs to be dispersed over many steps. This reward signal, equivalent to the first one, is called the **Shaped** reward and is described in more detail under approach 3. The shaped reward is equivalent to the Std reward in terms of return.

The agents of Approach 3 receive a larger matrix for their observations because the RL agents must know which experts are already in their collection and the experts who haven’t yet been decided.

Extended Softmax: Is an implementation of the algorithm presented in [35] and described in Section 3. The algorithm chooses the experts to consult for each patient based on its learned preference to experts. The votes are combined with the entropy forecaster for the final classification. The number of experts to consult (expert budget) is an external choice that needs to be made for the algorithm. How rapidly the preference of the algorithm to experts is updated is another parameter that needs to be chosen. These two parameters are chosen and kept constant throughout the entire episode.

Table 1 regroups the information explained in this section in a table format for comparison.

	Approach 1	Approach 2	Approach 3	Extended Softmax
Observations	confusion matrix	std	std	-
Actions	experts + weights	experts	yes/no	parameters
Reward	std	std	shaped	std

Table 1: Depiction of how the observations, actions and the rewards are defined for each implementation of the expert advice mab as a RL environment.

5.1 Approach 1 (All Experts In One Step)

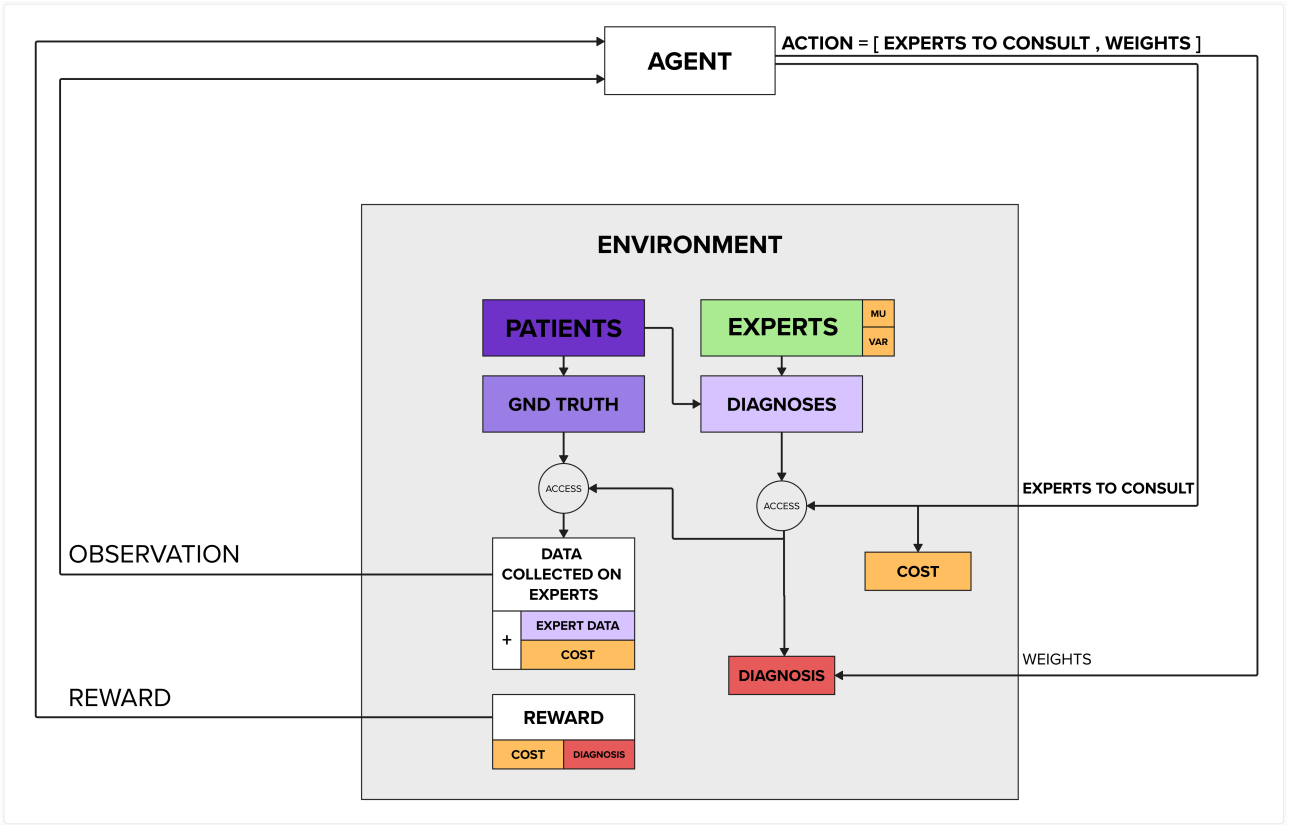


Figure 14: Approach 1 Expert-Advice MAB Setup: The agent gathers information by consulting experts (green). The Orange parameters represent parameters of the environment. Patients, and classifications given by the experts to those patients are all represented in purple. The agent decides which experts to consult at each round (for each patient), as well as what importance to place on the votes of each expert’s positive and negative votes. These weights are then combined with the experts’ votes to assign the final classification for that patient (red).

In general, the agent receives data regarding the current estimated competence of the experts. This information is only available through consulting the experts, after which the ground truth of the patient’s health is revealed. This allows for the creation and updating of a confusion matrix for each expert every time they are consulted. The diagnoses that the agent is able to purchase inflict a cost, and this cost, together with the correctness of the patient’s health classification, determines the reward signal.

The standard reward (**STD**) is just a linear combination of the two objective functions (correct classifications and number of consultancies).

$$R_t = \begin{cases} +1 - \sum_k c_k & \text{if } \text{diagnosis}_j = \text{ground truth}_j \\ -1 - \sum_k c_k & \text{if } \text{diagnosis}_j \neq \text{ground truth}_j \end{cases}$$

The c_k represent the consultancy cost of consulting an expert with k being the set of experts consulted on that round. Each cost c_k is uniformly varied over the interval $[0.04, 0.12]$ in all experiments, regardless of that experts’ classification capabilities. This cost range is critical to the problem dynamics; excessively high costs could lead agents to forego consulting experts altogether, opting instead to classify patients randomly, would yield an expected reward of zero. The upper bound of the cost range is set such that even the most expensive expert ($c = 0.12$) could still, on average, yield a positive reward. Conversely, the lower bound facilitates scenarios where agents might consult multiple experts to enhance their classification rate while achieving similar or marginally improved reward outcomes. This setup creates a more nuanced problem where the optimal number of experts to consult isn’t immediately apparent.

In this setup, the agent provides not only the selection of which experts to consult based on past data but also how to combine their votes by assigning weights. The agent thus also plays the role of the forecaster in this setup.

To effectively fulfill this role, it is crucial for the agent to observe the confusion matrix of each expert, which serves as a measure of the expert’s skills. Additionally, the agent must consider the number of times each expert

has been visited, to differentiate an expert who has never been wrong over 100 visits than one who has never been wrong over 2 consultancies. This data, along with the knowledge of the current round and the number of round remaining rounds, is what the agent observes as the state of the environment.

5.2 Approach 2 (All Experts In One Step + Forecaster)

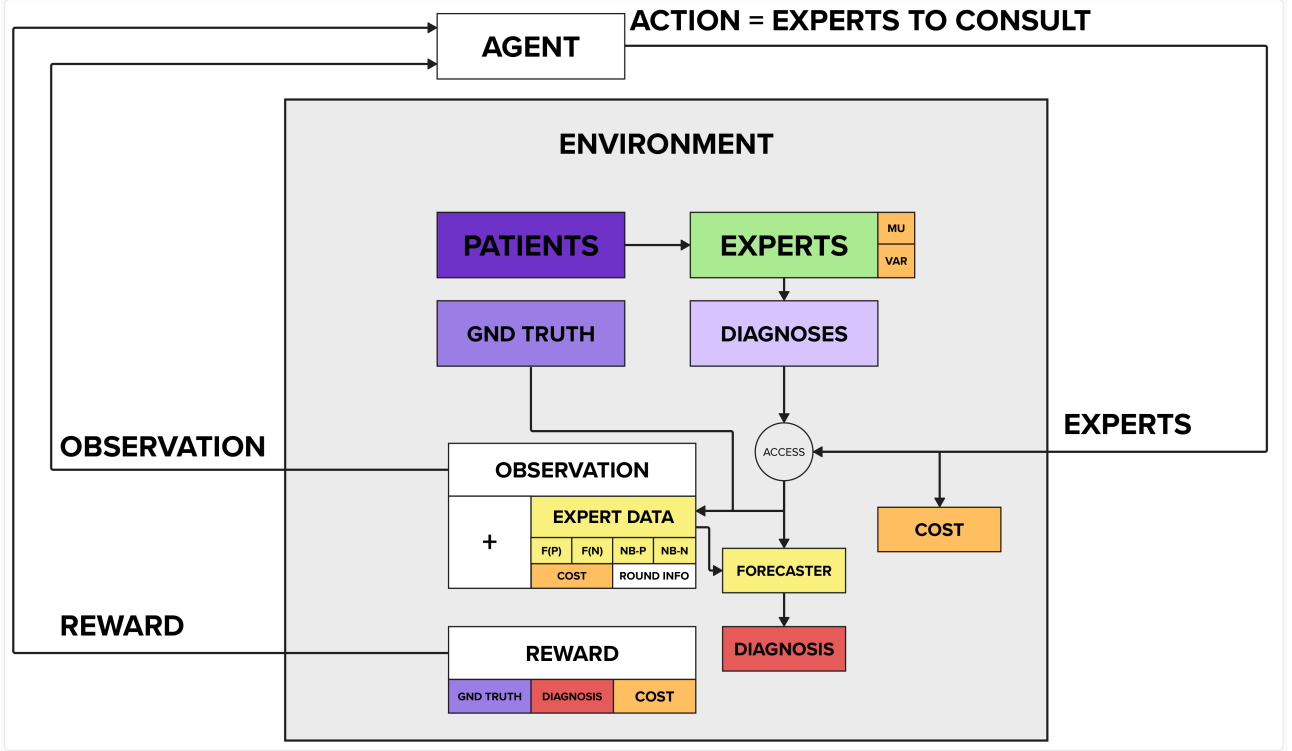


Figure 15: Approach 2 Expert-Advice MAB Setup: experts (green). The Orange parameters represent parameters of the environment. Patient data and expert diagnoses are in purple. The agent decides the set of all experts to consult at each step, the advice gathered from these experts is then combined with the entropy forecaster (yellow) to assign the final classification (red). The Forecaster uses the empirical confusion matrix of each expert to provide weight values.

The setup (Figure 15) for this RL environment closely resembles that of Approach 1, with the main difference being that the agent is no longer required to provide weights for expert votes, which significantly reduces the action space from continuous to discrete. The task of combining the votes is delegated to the forecaster. The forecaster uses the empirical confusion matrix of each expert as an unbiased estimate of that expert’s true confusion matrix at each step.

The observation vector associated with each expert is directly the positive and negative vote weights the forecaster calculates for that expert, as well as the logarithm of the number of times the expert cast each vote in the past as a measure of uncertainty with respect to the estimated weight values. The logarithm is preferred over the number itself due to varying rapidly when the number is small, changing less and less as the number gets larger just like the uncertainty we have about the experts.

5.3 Approach 3 (One Expert Per Step + Forecaster)

Approach 3 differs from Approaches 1 and 2 due to how the experts are selected for consultation. Both Approaches 1 and 2 decide the collection of experts to be consulted for a patient on a single step. Approach 3 however, makes this decision for a single expert at each step, therefore it is required to make this decision for each expert before moving onto the next patient.

The motivation behinds this difference comes from a stochastic bandit problem where the agent can choose multiple arms to play on a single step but only observes the sum of the rewards associated to the arms, thus making it difficult to attribute the rewards to arms. In a sense, the agents that learn under Approach 1 and Approach 2 suffer the same fate. They may consult 3 experts who together managed to classify the patient correctly, but one of the experts was wrong in their classification and the final vote was correct due to the majority being right. In this circumstance, it would be nice if the total reward could be broken up into parts,

with positive parts going to the experts who have made correct classifications and negative parts going to those who made wrong classifications, thus attributing the rewards to the arms.

In RL, designing the reward signal in ways that give it desirable properties while still keeping the cumulative reward identical is called reward shaping. On our problem, we attribute positive rewards to the agent when consulting experts with correct classifications and negative rewards to taking experts with wrong classifications, but an extra step is added after having gone over all the experts to equalize the reward to the Std reward. This way, the total reward obtained for each patient, over $K + 1$ steps, is equivalent to 1 step of the Std reward for a γ discounting factor of 1.

Having to choose the collection of experts over multiple steps means that the observations need to be modified as well, because when making the decision of whether or not to add an expert to the set of experts to consult, it needs to know the experts already in it, as well as the experts for whom this decision has not yet been made. Other than this detail, the observations for Approach 3 are the same as those of Approach 2, namely that the expert matrix contains the forecaster weights associated to each expert as well as confidence measures on these experts.

In this setting, the RL agent no longer gets to decide in a single step, whether or not to consult all the experts. It now gets to make this decision for each expert individually in a sequential manner. The action space $\mathcal{A}$ was $\{0, 1\}^K$ for Approach 2, but it gets to be $\{0, 1\}$ for this setup.

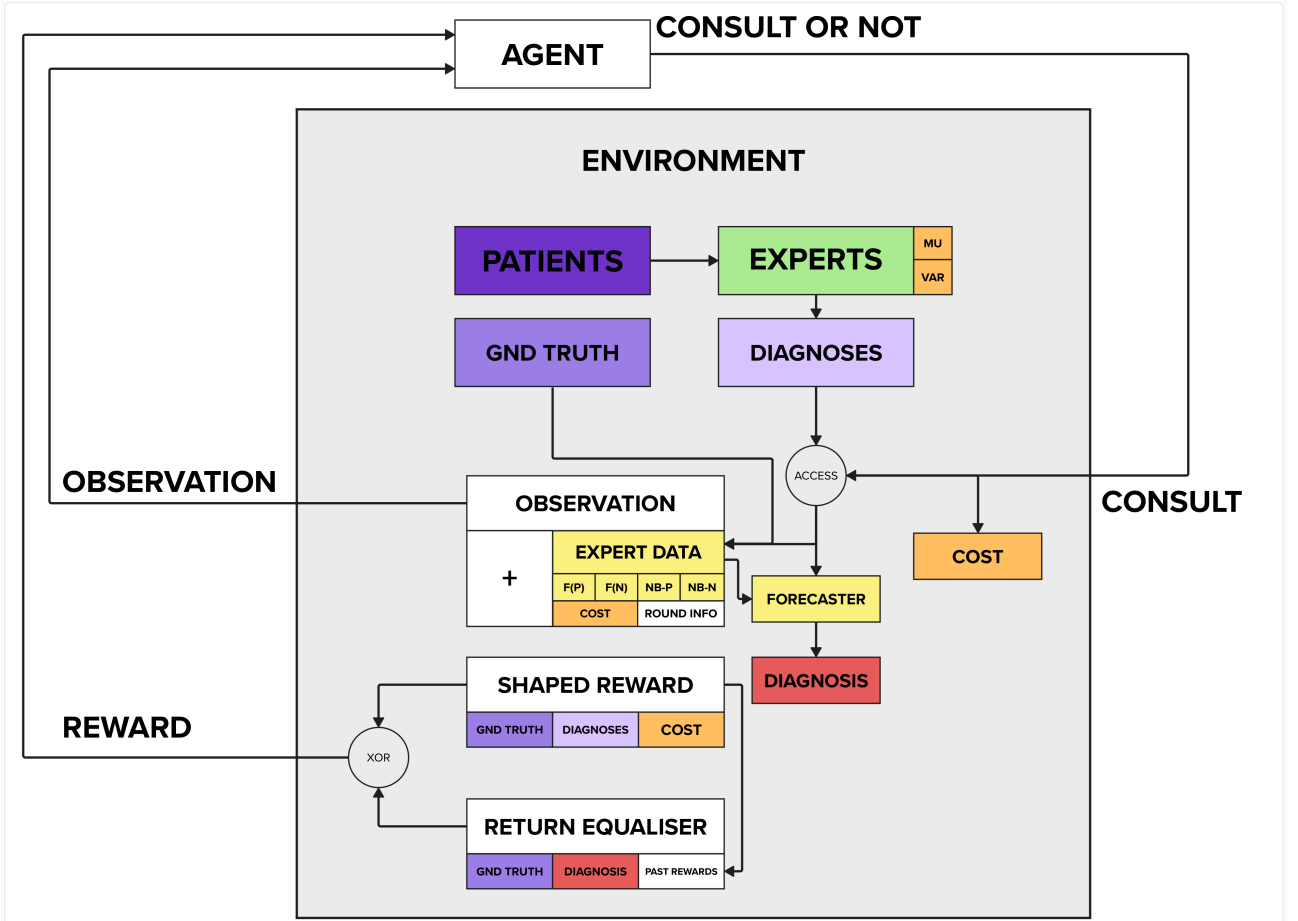


Figure 16: Approach Expert-Advice MAB Setup: experts (green). Problem Parameters (Orange). Patient data and expert diagnoses (purple). Forecaster (yellow) for classification (red). At each step, the agent decides to consult or not a certain expert. Depending on the success of the agent taken, reward is given. In the end, the sum of the rewards for one sweep of the experts is brought back to the **STD** reward.

5.4 Ordering

The “ordering” is not an approach to setting up the expert advice MAB as an RL problem in and of itself, rather it is a modification that can be made to the observation matrices for the other approaches to hopefully improve the performance. The idea is to exploit the symmetries of the problem and can be motivated in the following way:

The idea can be motivated by analyzing Table 3, which depicts two different possible observations that an RL agent might receive from the environment. The comparison makes it evident that the ID associated to each expert is nothing but an identifier for that expert, and that the decisions made by the RL agent should probably not depend on these IDs, assuming that the underlying probability distributions does not depend on the IDs. If, hypothetically, the best experts to consult for the next round are the **red** and the **blue** experts in the first scenario, then it should be also these same experts in the second scenario.

	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5
Positive-Vote Weight	0.37	0.51	0.60	0.66	0.42
Negative-Vote Weight	-0.37	-0.86	-0.22	-0.93	-0.10
Positive-Vote Occurrences	65	15	63	36	56
Negative-Vote Occurrences	40	44	9	5	34

	Expert 1	Expert 2	Expert 3	Expert 4	Expert 5
Positive-Vote Weight	0.51	0.37	0.66	0.60	0.42
Negative-Vote Weight	-0.86	-0.37	-0.93	-0.22	-0.10
Positive-Vote Occurrences	15	65	36	63	56
Negative-Vote Occurrences	44	40	5	9	34

Table 3: Depiction of two possible observations for the agents that are identical after a permutation.

Knowing that the neural networks used for the agents have to shape their decision boundaries in order to maximize the reward, this symmetry can be exploited to make it easier/faster for neural networks to learn.

The permutational symmetry motivated by Table 3 can be exploited in different ways:

- Either the policy could be made in such a way that permuted inputs yield permuted outputs, achieving equivariance. This approach is analogous to how translational and rotational symmetries required in segmentation tasks are incorporated into the agent’s neural network through the use of radially symmetrical kernels. Similarly, there are permutation equivariant neural network architectures that exist and have been tested to work for such equivariant tasks [33]. A similar approach is used in [32] with the use of attention mechanisms to internally extract permutation invariant features, making the PPO agents permutation invariant.
- A simpler approach in our case, however, is to feed the observations in an orderly fashion (ordered among the experts), ensuring that the two different configurations can never both be observed. It is important to be careful when looking at the outputs of such an agent because the outputs need to be returned to their original order by performing an inverse permutation.

One might also consider data augmentation as a method to help the neural network learn the symmetries of the problem. However, this approach is not suitable for such RL problems because the actions taken by the agent can change episode trajectories. If the neural network makes two different decisions on the set of experts to consult for different permutations, it would require processing entirely different episodes for each choice, which is impractical.

We might suggest two different advantages for exploiting this symmetry:

The first advantage is that this operation could improve the training sampling efficiency of the agents. Since scenarios 1 and 2 are effectively equivalent under the symmetry, encountering just one of these scenarios is sufficient if the experts are ordered according to a specific ordering function. This reduces the need to train on multiple equivalent scenarios, thus streamlining the learning process.

The second is that this can greatly increase how malleable our neural network agents get. As a matter of fact, training will tune the coefficients of the neural network to place the decision-making boundaries for consulting or not consulting an expert, where they need to go. With fewer decision boundaries to learn, the network can allocate more of its parameters to fitting more complex decision-making boundaries, leading to more precise and effective decision-making.

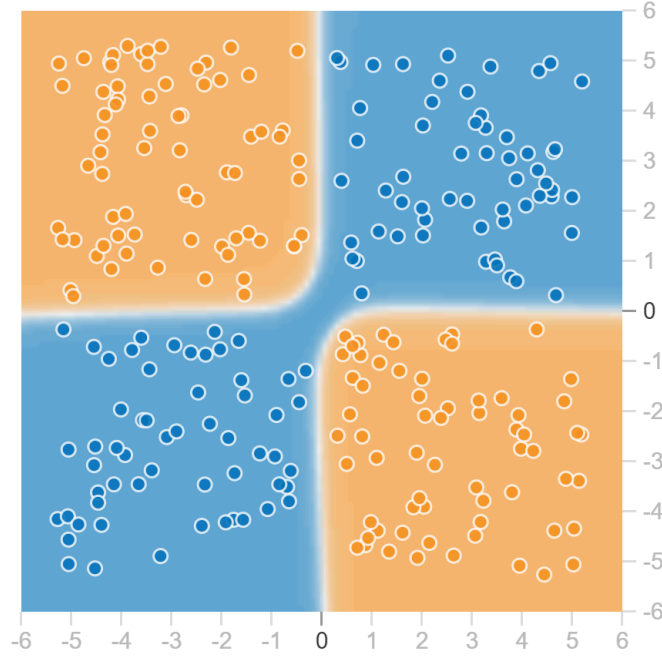


Figure 17: neural network decision making boundary for a classification task. This problem is famously known as the inseparability problem, requiring a neural network that has at least two layers to solve with the given features as input. The figure depicts an anti-symmetry but still conveys the idea that if the neural network has to tune its coefficients to fit the symmetric counterparts of the same decision making boundary, it loses malleability, meaning that it has less freedom to learn other details.

The application of the ordering technique to Approach 3 can be visualized in Figure 18.

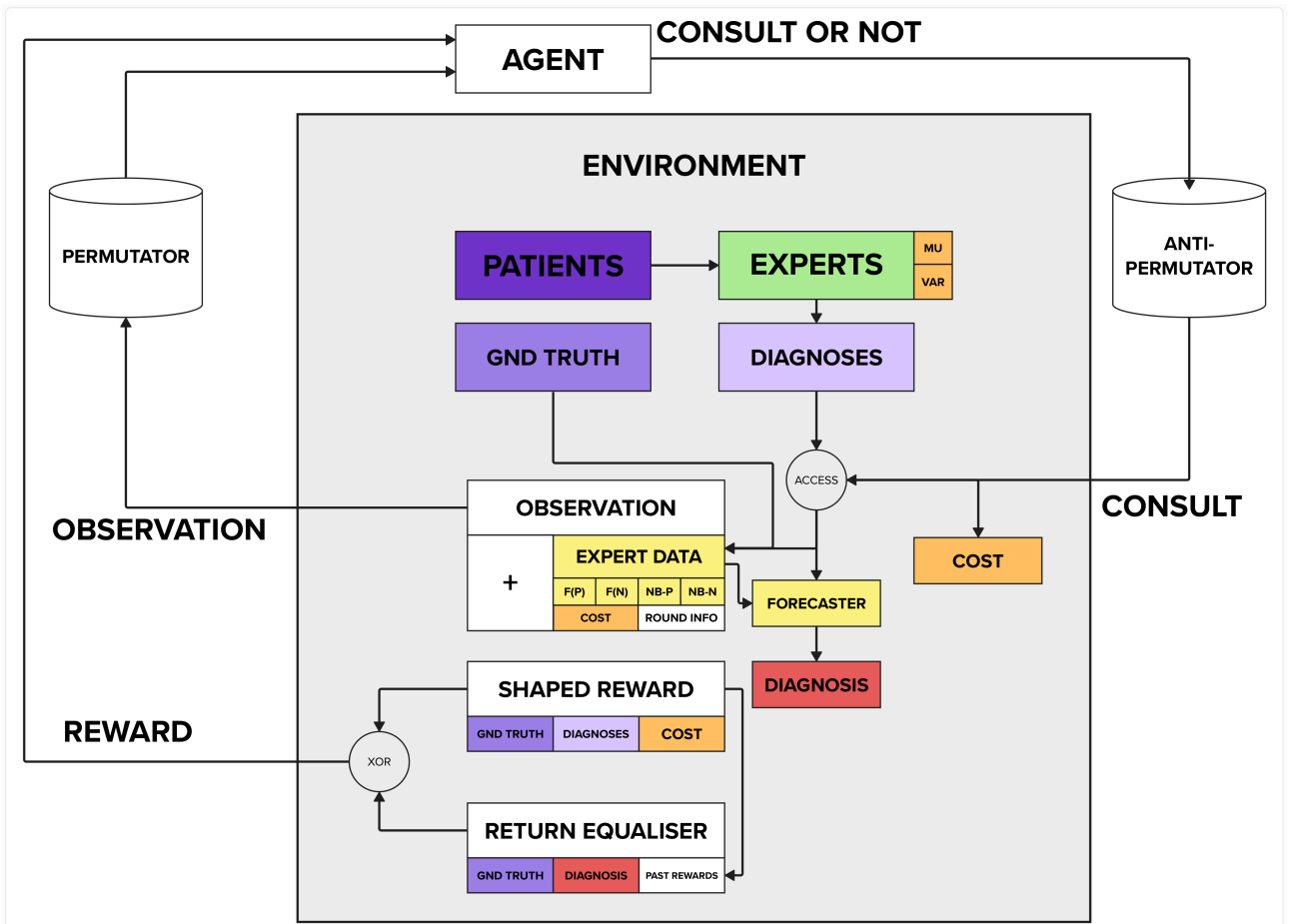


Figure 18: ordered Approach 3 Expert-Advice MAB Setup: Same as in Figure 16, but the inputs that are always presented in such a way that permutations of the experts do not alter the order in which the agent perceives them. The outputs are then assigned to the correct experts by the inverse permutation.

Figure 18 shows an additional advantage. Not only does this method reduce the observation space size by $K!$, enhancing training sampling efficiency by bringing samples closer together in a space that is $K!$ times smaller, but the ordering itself can carry information to the agent. Specifically, if we were in possession of an algorithm that was competent in this Expert Advice MAB, and if we had access to the internal preferences this algorithm has towards experts, these could be used to perform the ordering, thus making it very easy for the neural network agents to learn to perform the same operation. This is because ordering the experts in such a basis makes the ordering strategy canonically simple to attain. Because it can lead to a dominant local minimum where the agent doesn't learn anything new and always picks whatever the strategy used for the ordering would have taken. The experts will simply be ordered with respect to their costs in this work.

5.5 The Extended Softmax Method:

The algorithm from Section 3 is one that does the exploration and exploitation according to a budget of experts to visit per turn and a greediness parameter. However, this algorithm requires two inputs, the greediness parameter as well as the expert budget. Initially it was the idea to have an external RL agent wrapped around the Extended Softmax algorithm who chooses these two inputs based on what it observes from the environment for the softmax function. The intention was for the RL agent not only to avoid managing the combination of expert votes but also to be relieved of the responsibility of selecting which experts to consult. Instead, the RL agent would only determine the number of experts to consult in each round and how greedy the inner Extended Softmax should behave concerning its preference for experts. However, agents produced this way performed poorly with respect to their constant consultancy budget and greediness parameter counterparts. As a result, deciding not to involve RL with the Extended Softmax, we perform hyper-parameter optimization over these two parameters and compare the best performing parameters with the different approaches.

5.6 The Policies

The Policies used during the experiments were all Proximal-Policy-Optimization policies. Similar to Reinforce (Section 2.12), PPO uses one policy function π that assigns probabilities to each action based on the observations. The function π known as the policy network, utilizes a neural network to generate these probabilities. While Reinforce accelerates learning using a baseline function (a state value function), PPO employs a similar function called the advantage function (an action value function) to achieve the same goal. The advantage function estimates how much better one action is compared to others in terms of expected return based on the observations. This function is also referred to as the critic network because a neural network is used to compare action values.

The architecture used for these networks is depicted in Figure 19. Each network begins with a layer dedicated to normalizing the data—a common practice in deep learning that has been shown to enhance the performance of similar PPO agents in stochastic MAB problems Section 8.1.

Following the normalization step, the data matrix is processed by a feature extractor composed of convolutional layers that target the data for each individual expert. The output of this feature extractor is then merged with other observations provided by the environment before being passed through fully connected layers. The final step is either a softmax pooling from a set of activation nodes at the output of the policy network for discrete action spaces or a single scalar output value for the critic network.

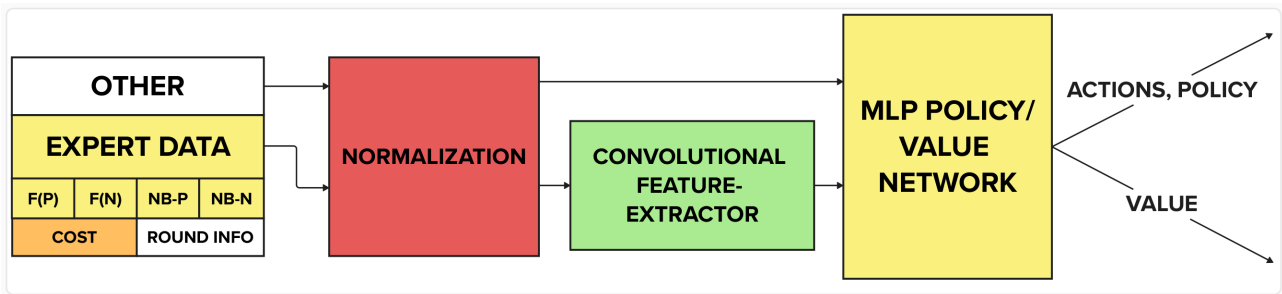


Figure 19: The neural network architecture used for the policy and the critic networks of the PPO agents trained on the environment

The rationale behind using convolutional layers to process the expert data matrix is to enhance the generalizability and efficiency of learning. Given that the number of data points in the expert data matrix can be substantial, depending on the number of experts (similar to the pixel count in an image), it's crucial to ensure that all data related to the same expert is processed together, utilizing expert-specific features. The feature

extractor narrows progressively as it processes the data from left to right, with the aim of summarizing the information about each expert using fewer data points.

Another potential advantage of these networks is their scalability. In contrast, using fully connected layers throughout the network poses a challenge: if a policy is learned for an expert advice MAB with 8 experts, it is not immediately clear how to adapt this policy to an expert advice MAB with 20 experts. One possible solution to this problem is to train feature extractors on the 8-expert MAB and then apply the same network, with the same coefficients, to a network responsible for decision-making with 20 experts. The expectation is that the transferred weights might either accelerate learning or help achieve a better global optimum if they were well-tuned initially. However, this hypothesis was not tested in this Master's thesis.

In order to tune the hyper parameters for the environments, hyper-parameter optimization was done on each environment for all the possible combinations of the parameters listed below:

- Feature extractor number of hidden layers and neurons: [1] , [2] , [3]
- MLP policy and critic networks (same shape imposed): [3,3] , [4,4] , [6,6] , [12,12]
- Entropy coefficient (used to soften the outputs of the policy network by penalizing deterministic policies for exploration): 10^{-1} , 10^{-2} , 10^{-3} , 0
- Learning rate: constant 0.0003 for all algorithms.
- Batch size: 2048
- Gamma discount factor: 0.99
- PPO clip range (to avoid making large abrupt changes to the policy, for stability): 0.2

Each network was trained for 800000 steps. This means that agents of Approach 3 received 9 times less data than the agents of other approaches, this is justified by the fact that they consume 9 times more computational power per episode. As a result, agents of Approach 1 and 2 receive 1000 episodes while those of Approach 3 receive 111 episodes.

Then the two most promising networks, based on the highest average reward, for each architecture were then selected for further training.

6 Results

6.1 Choice of Metrics

The expert advice MAB algorithms are compared using multiple figures of merit. The total reward accumulated over an episode (800 patients) is the first point of comparison, as to maximizing this quantity is the objective of the RL algorithms. This total reward is composed of two key components: the number of correct classifications and the expert consultancy costs, both of which are also important to examine.

Another basis for comparing these algorithms is the expert utilization distribution, which shows how frequently each expert is consulted. Analysis of Table 3 suggests that the selection of experts should be based solely on data, independent of the expert IDs. Consequently, all expert IDs should be visited approximately equally often. A significant discrepancy in the frequency of expert visits based on their IDs over many episodes indicates a failure to adequately explore all experts, leading to suboptimal performance [9]. This is because the best expert is just as likely to be one of the less frequently consulted IDs as any other. A histogram depicting how often each expert is visited by their ID is provided in Figure 21 or further analysis of this criterion.

Algorithms that fail to explore all the arms of a MAB risk missing out on the best arm with a constant, non-zero probability. If the best arm is missed out on in a constant number of cases, the expected regret cannot be better than $O(T)$ on a regular stochastic MAB, leading to algorithms that perform worse in expected regret [9]. The same idea holds for the expert advice MAB. Therefore, it is crucial to check if the exploration of all experts is present or lacking.

Finally, the algorithms will be compared against the cheapest expert within the coalition as a benchmark. The cheapest expert is chosen for comparison instead of the average expert because classification rates are independent of cost, meaning the cheapest expert is expected to earn the most rewards, making it irrelevant to look at the average expert. It is important to clarify that the cheapest expert should not be confused with the best expert. Though the best expert in terms of reward is more likely to be the cheapest expert than the others, cases arise where a more expensive expert scores more reward due to higher classification rates.

To illustrate the comparative performance of agents trained with each approach, Table 4 summarizes key metrics including total rewards, classification rates, cost efficiency, and expert utilization over multiple episodes. To help better understand the data in Table 4, the expert consultancy profile of the agents is shown in Figure 21. Additionally, a graph showing the costs incurred by the agents when consulting experts and the classification accuracy achieved as a result is provided in Figure 20.

Finally, to verify if the ordering principle introduced in Section 5.4 had a positive impact on the sample efficiency of learning, the training curves for the four best agents of Approach 2 and 3 are compared to their ordered counterparts Figure 22 and Figure 23.

Setup	Reward	Classification Rate	Cost Spent per Patient	Independence to ID	Presence of Experts Exploration	Experts Visited per Turn
Approach 1	230.5 ± 16.9	$73. \pm 1. \%$	0.17	No	3.89/8	2.13
Approach 2	291.7 ± 23.9	$76.4 \pm 1. \%$	0.16	No	2/8	2
Approach 2 ordered	358.8 ± 20.5	$81. \pm 1. \%$	0.17	Yes	3/8	3
Approach 3	331.2 ± 24.7	$75.5 \pm 2. \%$	0.10	No	1.87/8	1.74
Approach 3 ordered	331.4 ± 35.5	$73.3\% \pm 2. \%$	0.05	Yes	3.56/8	0.95
Extended Softmax	307.7 ± 19.3	$77.2 \pm 1. \%$	0.16	Yes	8/8	2
Cheapest Exp.	288.7 ± 28.7	$70.5 \pm 2. \%$	0.05	Yes	1/8	1

Table 4: The average results achieved by the best agent (in terms of reward) for each setup over 110 episodes of length 800 with costs $\in [0.04, 0.12]$, expert bias $\in [-1, 1]$ and variance $\in [0, 1]$. Reward and classification rate also include a 95% confidence interval.

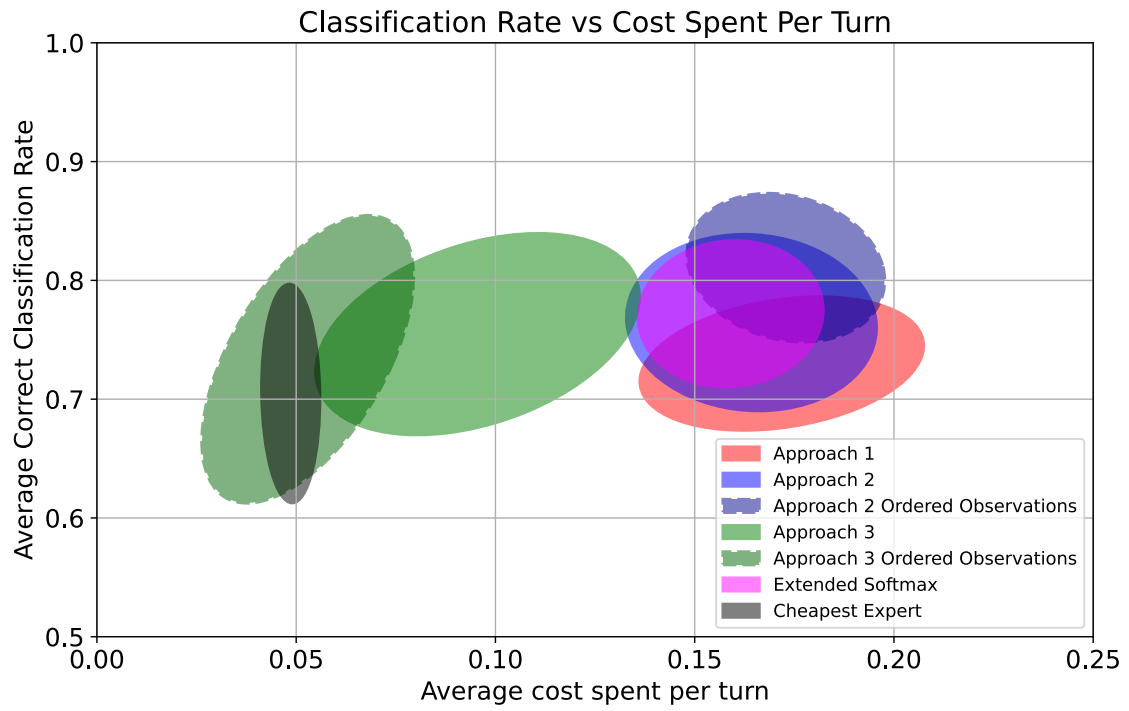


Figure 20: The cost vs classification rate for the best algorithm for each approach. The ellipses indicate the covariance between spending and classification accuracy.

Expert Visit Rate by ID

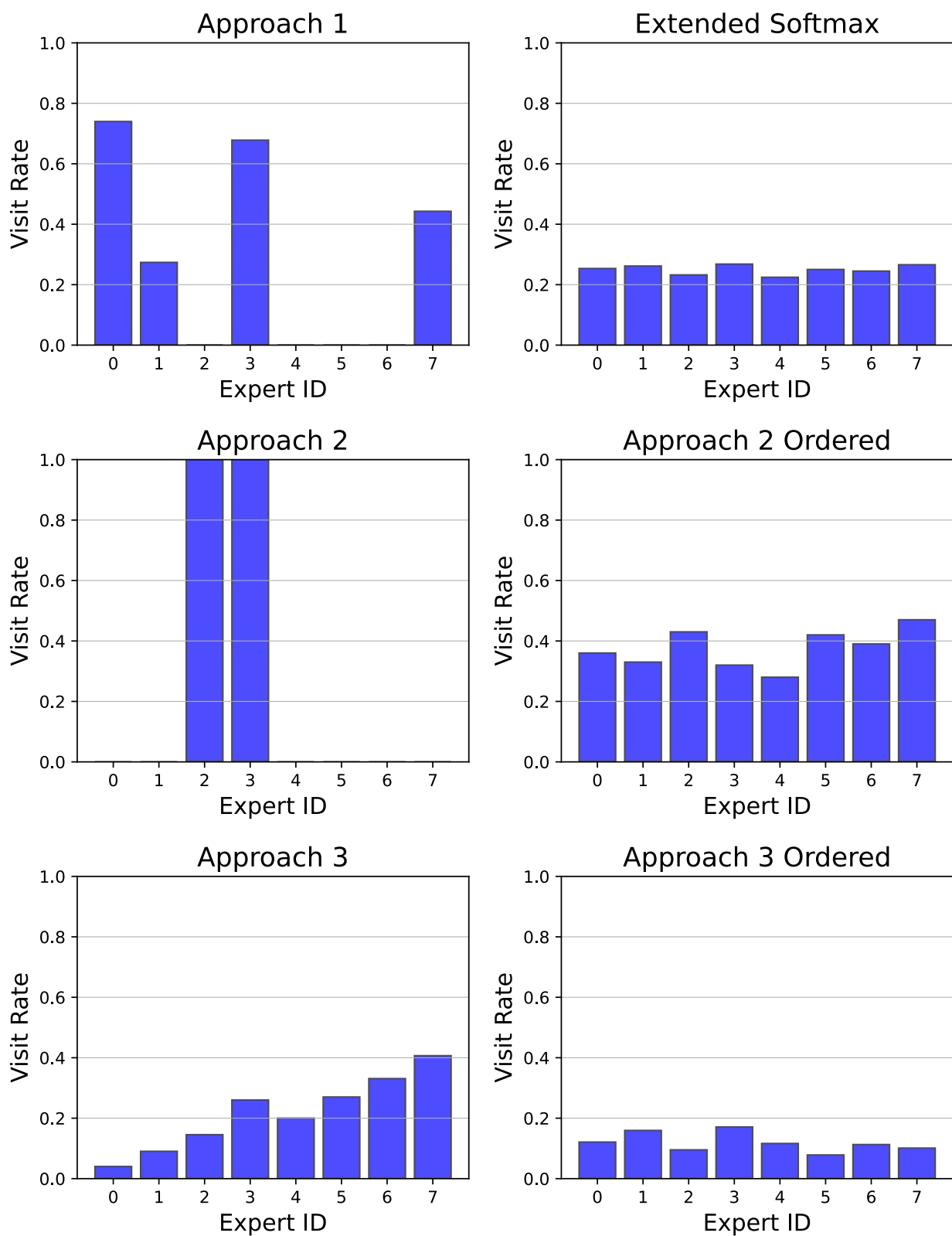


Figure 21: A histogram showing how often each ID of experts is consulted by the best agent trained with each approach.

Training Curves for the Top Four Agents of Approach 2 and 3

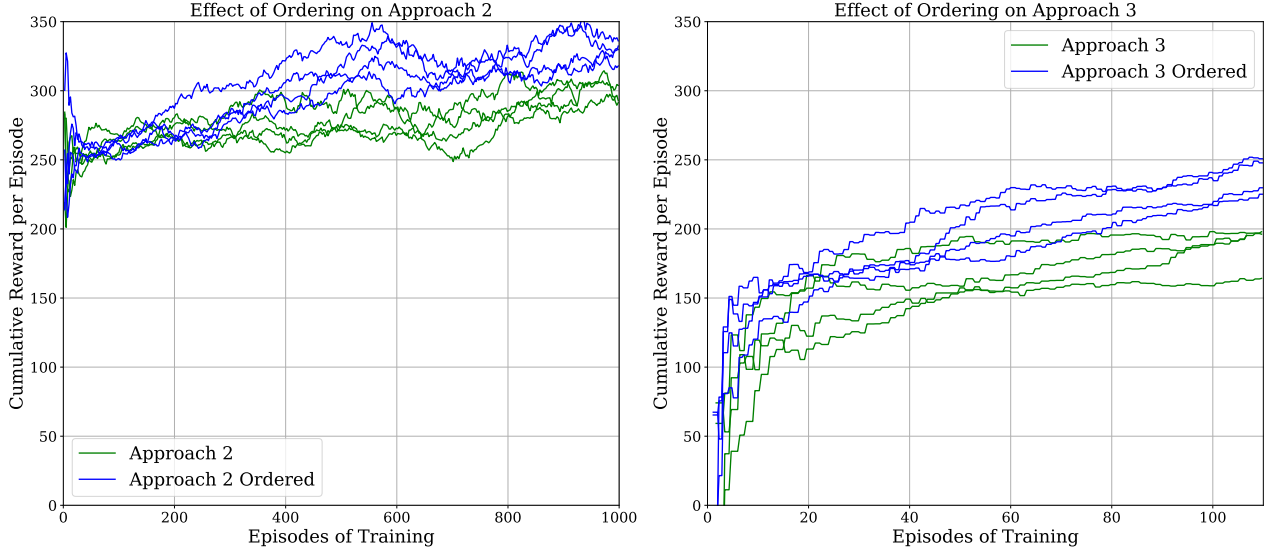


Figure 22: Training curves over the first 800k steps of training for the best four agents of Approach 2 and Approach 3 for understanding the effects of ordering the observation space.

Effect of Ordering on the Best Agents of Approach 2 and 3 at 800k Steps

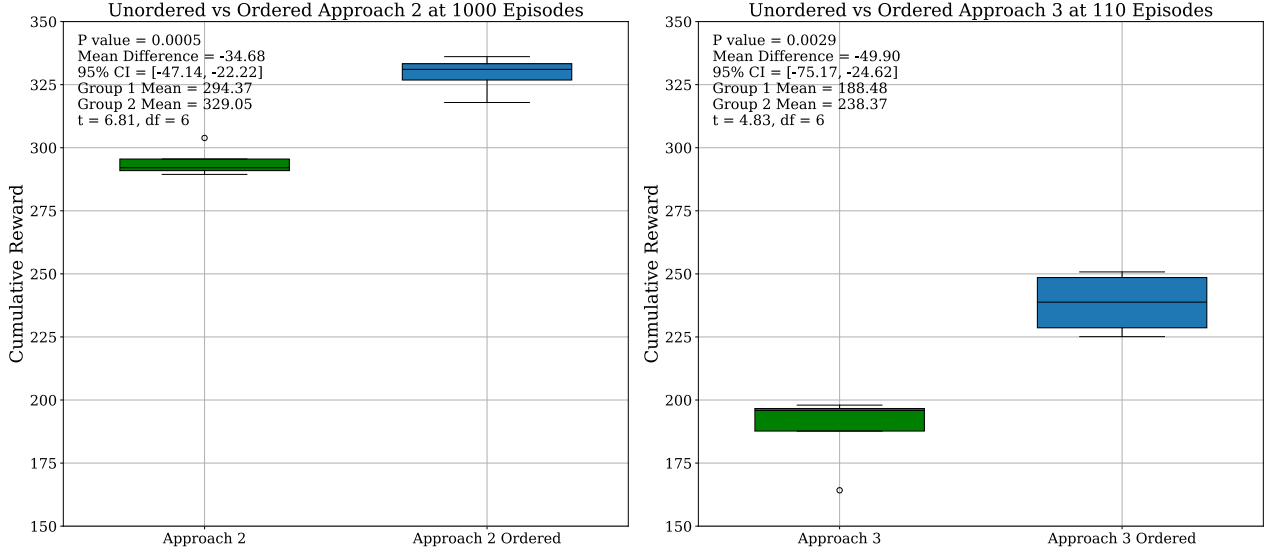


Figure 23: Comparison of how the four best agents for Approaches 2 and 3 perform after the initial 800k steps of training.

6.1.1 Approach 1

This approach is the least successful of the group in terms of reward and does not meet the other important criteria for an expert advice MAB. The basic thing the network has learned is to assign good trust weights to experts with the ID of 0,1,3 and 7 and it has learned to always consult some combination of these experts at every turn as can be seen Figure 21. Despite this, this approach manages to have a higher classification rate than the average expert, not because it effectively identifies better experts, but because it consults 2 – 4 experts in a single step and relies on a majority voting mechanism to determine the output, thereby improving the classification rate.

The set of experts this agent decides to consult are arbitrary, as the consulted experts simply happen to have those specific IDs, which have no inherent significance regarding their classification rate or cost. There are no expected differences between an expert with ID 0 who is frequently consulted, and an expert with ID 2, who is never visited.

The likely reason for this is that consulting more experts leads to poorer classification rates for this agent due to its inability to effectively combine the votes. This is evident as it achieves a lower classification rate than the agent from Approach 2 ($73\% < 76.4\%$), despite consulting more experts on average ($2.17 > 2$).

6.1.2 Approach 2 and Ordered Approach 2

The reward for Approach 2 is the second lowest of the group. Figure 21 shows that the agent trained using this approach consistently consults the experts with the same IDs, regardless of the observations. The classification rate achieved by this approach is 76.4%, which is worse than the expected average classification rate of two experts, as shown in Figure 12. Figure 12 indicates that a two-expert forecaster using the entropy forecaster should achieve a classification rate of approximately 79.5%.

There are two main reasons for the discrepancy between these values. First, when creating the plots in Figure 12, experts' confusion matrices is assumed. This allows the forecaster to weigh the votes of each expert according to their level of competence. In the Approach 2 environment, however, the forecaster has no prior knowledge of which expert is better and must learn this information progressively over the 800 steps, resulting in lower classification rates.

The second reason for the lower-than-expected classification rate with two experts also relates to the forecaster. A problematic situation arises when an expert fails to make a correct classification the first time it is consulted. These incorrect classifications cause the forecaster to assign negative weights to the positive votes of experts and vice-versa, leading to a scenario where adding the expert to the coalition actively degrades the classification rate. This effect is evident in the classification rate of the cheapest expert in Table 4, where the expert achieves 1% less in classification rate with the forecaster enabled than it would have achieved without a forecaster.

The introduction of forecasters in this approach is intended to promote exploration, especially when compared to Approach 1. Approach 1 struggles with exploration because it might assign distrustful vote weights to the experts it consults, leading to a situation where the agent not only incurs the cost of consultation but also degrades the classification quality. Avoiding exploration altogether prevents this issue. However, this is not the case for Approach 2. Even if a consulted expert makes a mistake during the first consultancy, the agent still has other experts to consult, allowing it to circumvent the forecaster's second issue. Despite this, Figure 21 shows that this agent only visits two experts (experts 2 and 3). These experts are arbitrary of course as the ID has no implications for any of the expert parameters. The failure to consult additional experts if the first classification is incorrect can likely be attributed to the RL agent not learning this behavior.

Ordering the expert observation matrix with respect to costs, yields the best results out of all the approaches. The agent learns to consult the 3 cheapest experts no matter their other characteristics. This situation is unrealistic due to all of the workload being placed on the same experts, but it makes sense within the context of our problem setup. Consulting two experts results in a dictatorship where the vote of one is deemed more important than the other, hence making the classification rate be in between that of the average expert and the best expert on a coalition of two, the difference being explained by the imperfect knowledge and estimation of the expert confusion mistake parameters.

However, when 3 experts are involved, the vote likely does not evolve to dictatorship of one expert over the others but is likely to evolve to a majority voting. Thus, achieving the prediction rate predicted by Figure 12 for 3 experts, which is a better classification rate than the best expert out of the 3 experts who form that coalition. Furthermore, because the costs aren't correlated with the expert classification rates, consulting the 3 cheapest experts means that it pays the cheapest possible price for the quality classifications. The recognition of the cheapest experts is done not by learning of the neural network to identify the price parameters in the data but by the experts always being presented in the same order. This is something that we had hypothesized when introducing the ordering change: ordering the experts with respect to the preferences of another algorithm (in this case, an algorithm who prefers cheap experts) makes the decisions of that algorithm canonically easy for the neural networks to learn by always choosing the same actions.

Moreover, the agent trained on the ordered version of Approach 2 visited all the expert IDs a similar number of times over many episodes. This is of course due to the fact that the costs are initialized randomly and independently of the ID, so that the three cheapest experts appear uniformly in all IDs. A similar thing will occur with the ordered version of Approach 3.

When it comes to training, ordering the agents of approach 2 showed significant improvement in performance (Figure 22). After 1000 episodes of training, the agents of the ordered version of the environment outperform their counterparts. This result is significant with a p-value of 0.0005 as depicted in Figure 23.

6.1.3 Approach 3 and Ordered

Figure 20 appears to indicate that the best agent trained on Approach 3 possesses the ability to spend more to gain a better classification rate. So, one might wonder whether or not the agent has learned to identify better experts and to take these experts regardless of their higher cost. However, this is simply not the case as the positive correlation between spending and better classifications is due to that extra spending being used to consult more experts. As indicated by expert consultancies per turn and the lack of exploration from Table 4, the agent has learned to only consult two experts exclusively just like approach 2, thus achieving very similar classification rates. Though this classification rate is importantly slightly less than that of approach 2 due to the algorithm sometimes deciding to consult a single expert per patient rather than two.

The act of sticking to one expert rather than consulting both is not a bad idea on this setup of the expert advice MAB. This is because after having consulted both for enough time to estimate the better expert (a question of cost accuracy trade off), it suffices to consult the better expert only. This keeps the same classification rate while also avoiding having to pay the cost for the other expert (an example of exploiting more and more instead of exploring as the episode nears its end). However, this is not what the algorithm for approach 3 has learned.

The peculiar shape of the plot for Approach 3 present within Figure 21 seems to indicate that the agent has a preference for higher IDs. There are two possible explanations for this:

One important observation is that the agent has learned to avoid situations where no experts are consulted, which would significantly reduce both the classification rate and the reward. If the list of experts to consult is empty, the agent is more likely to add the currently targeted expert to the list, especially as it nears the point of making a final decision before moving on to the next patient. The closer it gets to this decision point with no experts selected, the more likely it is to include the current expert in the collection. This behavior can result in the pyramid-like shape seen in [Figure 21]. Since the decisions are made progressively from left to right (following the order of experts' IDs), similar patterns are observed in other agents trained in this environment. While this might suggest that expert IDs influence decisions, a closer inspection provides a more accurate explanation.

A key takeaway from the data is the agent's ability to identify and select cheaper experts. This is evident from the average number of experts it consults per turn (1.74) and the average cost incurred per turn (0.10). Given that the average cost of an expert is 0.08, if the agent couldn't identify cheaper experts, it would likely spend more. This observation, along with the plot in [Figure 21], challenges the assumption that expert IDs don't impact expert parameters at initialization. However, it's reassuring to note that the environment functions correctly, and the real explanation lies in the agent's behavior.

The agent can distinguish between cheaper and more expensive experts and tends to prefer the cheaper ones. The pyramid-like plot seen in [Figure 21] arises because the agent is more inclined to select a cheaper expert if that expert has a higher ID. In other words, if a cheap expert has a high ID, the agent is more likely to choose them than if the same expert had a lower ID. Additionally, the agent prefers not to consult too many experts at once, and when it does consult multiple experts, it's usually because the experts with higher IDs happen to be the cheaper ones.

Moving on to the ordered version of Approach 3, Figure 20 indicates that this algorithm can also increase its classification rate by spending more per turn. Except that inspection of the data shows that this is only the case because the agent sometimes decides not to consult any expert and to perform the classification fully blind and randomly. This can also be seen in Table 4 where the number of experts consulted per turn is below 1 on average, meaning that it is sometimes equal to 0.

This is obviously not optimal for the environment from a reward perspective because the environment parameters were chosen specifically to avoid this type of behavior from being optimal.

One point of observation to make is that the agent for the ordered approach 3 does have the capacity to discern between good and bad experts. In fact, what seems to trigger the behavior where it decides not to consult any experts occurs if the cheapest expert (its favorite) happens to be a poor expert. This can cause the agent to either consult one more expert alongside the poor expert or it can also cause the agent to stop consulting experts altogether (the likelier option).

Similarly to the agents of approach 2, ordering the experts in the observation space significantly improved the performance of the agents in approach 3 (Figure 22). After the 800k steps of training, the agents in the ordered environment outperformed their unordered observations counterparts. This result is significant with a p-value of 0.0029 as depicted in Figure 23. Though this advantage was not kept by the best performing agent of approach 3.

6.1.4 Extended Softmax

Although the Extended Softmax algorithm did not achieve a higher total reward compared to the RL based algorithms on this experiment, this likely does not imply that the RL based methods are superior. The primary reason for the lower performance of the Extended Softmax algorithm is its insufficient adaptation to manage the different costs associated to the experts.

As a matter of fact, though the correct classification rate is not out of the ordinary for the two experts it consults, (because two experts can achieve such an exploration rate), it is the algorithm in the bunch that sees its correct classification increase significantly over the course of an Episode (due not only to the forecaster learning the appropriate weights but also being able to discard experts who made multiple mistakes).

Though the individual cost of each expert was used in the update rule when deciding to update the parameters with the algorithm present in Section 3, these numbers were small compared to those of the classification values being right or wrong. This algorithm could easily be improved just by giving it an initial preference skew towards cheaper experts and could achieve higher rewards and has the added benefit of being robust, meaning less variance in reward.

7 Conclusion

The primary objective of this thesis was to evaluate whether DRL algorithms could achieve a performance level comparable to or surpassing that of existing bandit algorithms in the expert advice MAB problem. The results presented in this thesis suggest that RL-based policies can indeed rival some basic algorithms. However, it is crucial to acknowledge that these RL-based approaches come with significant drawbacks, making them practically inferior to pre-existing algorithms in certain aspects.

One major drawback of RL-based methods is the lack of consistency in performance, particularly in the expert advice MAB context. Without carefully designed reward signals that penalize undesirable behavior, such as the decision not to consult any experts, RL agents may exhibit anomalies. For example, there were instances where RL-based algorithms failed to explore all available experts. While this issue might partly stem from the construction of the environment itself, it underscores the importance of thoroughly investigating RL agents to ensure their decisions are reasonable. Future research could explore different reward signals that penalize anomalies more harshly while not affecting agents that behave appropriately.

Scalability is another critical issue with deep RL methods. Pre-existing MAB algorithms and their extensions are often parametric, allowing them to be scaled to problems of varying sizes. Although these algorithms require parameter tuning when applied to larger problems, this process can typically be managed with computational tools and engineering intuition. In contrast, scaling RL-based methods to larger problems would likely necessitate the exploration of entirely new architectures or setups, beyond what was studied in this thesis.

Training RL agents also poses significant challenges compared to the hyperparameter optimization of existing algorithms. Deep RL agents require substantially more training time and computational resources. Moreover, they are highly sensitive to the random initialization of network weights, leading to significant variability in performance and behavior even among networks with identical architectures and hyperparameters. This fragility can result in inconsistent outcomes, which can be a major concern in practical applications.

The data requirements for training RL agents further complicate their use. Training often necessitates hundreds or even thousands of episodes to accumulate sufficient data. This is why it was essential to keep the neural networks small in our experiments; larger networks consistently underperformed compared to their smaller counterparts.

It remains to see how the weakening of some assumptions made on the problem statement affects the performance of RL and existing algorithms (experts not always available, non-stationary expert characteristics, delay or lack of ground truth data etc.), but these are likely to be challenging to overcome.

Despite these challenges, one of the potential improvements proposed in this thesis was the reordering of agent observations to avoid processing equivalent duplicates. This technique aimed to accelerate and improve learning, as discussed in [Section 5.4]. The results showed that reordering indeed sped up learning in both the stochastic MAB [Section 8.1] and the expert advice MAB [Figure 22, Figure 23] for Approaches 2 and 3. However, faster learning did not always translate into better long-term performance, as seen in Approach 3, where the best unordered agent eventually caught up to its ordered counterpart. It remains unclear how much of the observed improvement is due to the specific choice of reordering parameter (cost) versus the general technique, or random chance. These findings are reminiscent of results reported in [[32]], where PPO was applied to various tasks using both input permutation invariant and variant networks.

For continuing this line of work, I would suggest making a couple of modifications. First, it would be beneficial to use a more appropriate forecaster than the entropy forecaster employed in this thesis. Some design choices made during the development of this forecaster—such as fully trusting a new expert and completely distrusting an expert who has only made mistakes so far—were made to preserve symmetry in cases where “evil” experts might be present. However, this symmetry was not necessary for the experiments conducted in this thesis, as no “evil” experts were included. These design choices may have negatively impacted not just the exploitation but also the exploration behavior of the RL-based agents. RL methods are inherently designed to allocate fewer exploration steps to actions that are considered poor, which is intended to increase sample efficiency. However, this can lead to RL algorithms learning not to explore all experts if the forecaster does not make the first few visits to these experts sufficiently accommodating. Nonetheless, using a forecaster proved indispensable, as the agent in Approach 1 performed significantly worse than others without a forecaster.

Another recommendation would be to design a more specialized reward signal that penalizes RL agents for consulting the same set of experts repeatedly. As observed in this thesis, nearly all agents—particularly those

in the ordered Approach 3—tended to visit the same experts each round, regardless of the round. While this behavior might be justified from a reward perspective for the ordered Approach 2 agent (which has found a local maximum), it is not a desirable outcome in general and should be discouraged through an appropriately designed reward signal.

As the next step of this line work, it can be interesting to look at how networks similar to those of Approach 2 can be improved further. These networks suffer from having 2^K nodes at their output, which makes them less scalable than their Approach 3 counterparts whose output node size doesn't vary with K . To my knowledge, action space factorization techniques [36] have never been tested on expert advice MABs where they can be useful.

As a potential direction for future research, it would be interesting to explore how networks similar to those used in Approach 2 can be further refined. These networks suffer from scalability issues due to the exponential growth in output nodes as the number of experts K increases (2^K nodes). In contrast, networks used in Approach 3 are more scalable, as their output size does not vary with K . To my knowledge, action space factorization techniques, such as those discussed in [[36]], have not yet been tested on expert advice MABs and could prove beneficial in addressing these scalability challenges.

To conclude, RL-based agents have demonstrated their ability to compete with existing algorithms on small-scale problems under strong assumptions and with the application of specialized techniques. These agents can identify local optima with higher expected rewards, but this advantage comes with trade-offs, including increased computational costs, scalability challenges, and robustness issues. Further research is necessary to address these shortcomings and enhance the practical applicability of RL in more complex and varied environments.

Bibliography

- [1] Alessandro Carriero, Léon Groenhoff, Elizaveta Vologina, Paola Basile, and Marco Albera, “Deep Learning in Breast Cancer Imaging: State of the Art and Recent Advancements in Early 2024,” vol. 14, no. 8. 2024.
- [2] S. Kim *et al.*, “Deep-Transfer-Learning-Based Natural Language Processing of Serial Free-Text Computed Tomography Reports for Predicting Survival of Patients With Pancreatic Cancer,” vol. 8. 2024.
- [3] Shuroug A. Alowais *et al.*, “Revolutionizing healthcare: the role of artificial intelligence in clinical practice,” vol. 23, no. 1. Sep. 22, 2023.
- [4] Scott M. McKinney *et al.*, “International evaluation of an AI system for breast cancer screening,” vol. 577, no. 7788. 2020.
- [5] Aleksandr Esteva *et al.*, “Dermatologist-level classification of skin cancer with deep neural networks,” vol. 542, no. 7639. 2017.
- [6] Benoit Macq, *Coalitional Active Learning. Prospective Applications in Network Medicine*, (Sep. 30, 2021). [OnlineVideo]. Available: <https://www.youtube.com/watch?v=eNN-GL2yW7I&t=761s>
- [7] Karoline Freeman *et al.*, “Use of artificial intelligence for image analysis in breast cancer screening programmes: systematic review of test accuracy,” vol. 374. 2021.
- [8] Richard Weber, “On the Gittins Index for Multiarmed Bandits,” vol. 2, no. 4. 1992. [Online]. Available: <https://doi.org/10.1214/aoap/1177005588>
- [9] Aleksandrs Slivkins, *Introduction to Multi-Armed Bandits*. Now Publishers, 2019.
- [10] Sofia S. Villar, Jack Bowden, and James Wason, “Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges,” vol. 30.
- [11] Aditya Grover *et al.*, “Best arm identification in multi-armed bandits with delayed feedback,” vol. 84. PMLR.
- [12] Juan Qin, Wei Qi, and Baojian Zhou, “Research on Optimal Selection Strategy of Search Engine Keywords Based on Multi-armed Bandit.”
- [13] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal, “Survey on Applications of Multi-Armed and Contextual Bandits.”
- [14] X Huo and F Fu, “Risk-aware multi-armed bandit problem with application to portfolio selection,” vol. 4, no. 11.
- [15] Aurélien Garivier, Emilie Kaufmann, and Tor Lattimore, “On Explore-Then-Commit Strategies.”
- [16] Sophia Yang, *Multi-Armed Bandit Algorithms*, (2022).
- [17] Semiu Ayobami, “Towards an Improved Strategy for Solving Multi-Armed Bandit Problem.” Oct. 17, 2019. [Online]. Available: https://www.researchgate.net/publication/336650162_Towards_an_Improved_Strategy_for_Solving_Multi-_Armed_Bandit_Problem
- [18] Volodymyr Kuleshov and Doina Precup, “Algorithms for the Multi-Armed Bandit Problem.” Feb. 25, 2014.
- [19] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
- [20] Yasin Abbasi-yadkori, Dávid Pál, and Csaba Szepesvári, “Improved Algorithms for Linear Stochastic Bandits,” vol. 24. Curran Associates, Inc.
- [21] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári, “Exploration–exploitation tradeoff using variance estimates in multi-armed bandits,” vol. 410.
- [22] Rajeev Agrawal, “Sample mean based index policies by $O(\log n)$ regret for the multi-armed bandit problem,” vol. 27.
- [23] Faisal Siddiqi, “ML Platform Meetup: Infra for Contextual Bandits and Reinforcement Learning.”
- [24] Daniel J. Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen, “A tutorial on Thompson sampling,” vol. 11. Now Publishers, Inc.

- [25] Daniel Fink, *A Compendium of Conjugate Priors*.
- [26] Wikipedia contributors, “AlphaGo.” Wikipedia, The Free Encyclopedia, Jul. 27, 2023. [Online]. Available: <https://en.wikipedia.org/wiki/AlphaGo>
- [27] B. Ravi Kiran *et al.*, “Deep Reinforcement Learning for Autonomous Driving: A Survey.” Feb. 02, 2020.
- [28] Leemon Baird, “Residual Algorithms: Reinforcement Learning with Function Approximation.”
- [29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, “Proximal policy optimization algorithms.”
- [30] George Konidaris, Sarah Osentoski, and Philip Thomas, “Value Function Approximation in Reinforcement Learning Using the Fourier Basis.”
- [31] Vikram Waradpande, Daniel Kudenko, and Megha Khosla, “Graph-based state representation for deep reinforcement learning.”
- [32] John Mern, Dorsa Sadigh, and Mykel J. Kochenderfer, “Exchangeable Input Representations for Reinforcement Learning.”
- [33] Nicholas Guttenberg, Nathaniel Virgo, Olaf Witkowski, Hidetoshi Aoki, and Ryota Kanai, “Permutation-equivariant neural networks applied to dynamics prediction.” Dec. 13, 2016.
- [34] Ofer Dekel and Brendan McMahan, “CSE599s, Spring 2012, Online Learning.” 2012. [Online]. Available: <https://courses.cs.washington.edu/courses/cse599s/12sp/>
- [35] Yevgeny Seldin, Peter Bartlett, Koby Crammer, and Yasin Abbasi-Yadkori, “Prediction with Limited Advice and Multiarmed Bandits with Paid Observations,” vol. 32. 2014.
- [36] Shengpu Tang, Maggie Makar, Michael Sjoding, Finale Doshi-Velez, and Jenna Wiens, “Leveraging factored action spaces for efficient offline reinforcement learning in healthcare,” vol. 35.

8 Annexes

8.1 Experiment 2 (Stochastic Bandit With RL)

In order to test if Deep RL agents are appropriate to use in MAB problems and to see if the re-ordering of arms in the observation matrix (symmetry exploitation) does provide advantages, a second smaller experiment was done. These methods are tested on a stochastic bandit MAB described in section 2. This experiment is not important for the understanding of the thesis and is thus left after the conclusions for readers who aren't interested. The main takeaway is that normalization of inputs to neural networks and exploiting the symmetry of MAB problems can help improve the performance under some circumstances.

The MAB contains 10 arms. The underlying reward distribution of arm i follows a normal distribution, with parameters μ_i and σ_i^2 stationary and assigned to that arm at the start of an episode.

$$\text{Rew}_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

$$\mu_i \sim \text{Uniform}([-1, 1])$$

$$\sigma_i = 3$$

At each turn (step), the agent gets to choose one arm and observes the reward of that arm only, and the episode lasts for 200 steps.

It is important not to confuse the reward the MAB agent observes with the reward signal given to the RL agents. The reward associated to arm i is a stochastic quantity with mean Rew_i , however, the reward signal used to train the RL agents is the regret suffered by the agent by picking arm i . This regret corresponds to the difference between Rew_i of the arm i chosen by the agent and the Rew_* of the best arm on that instantiation of the episode.

Therefore, the perfect yet impossible scenario is the case of 0 regret where the agent did nothing but play the best arm from start to finish (200 steps). Any deviation from this perfect scenario results in regret, which is signaled to the RL agent by the reward signal with a negative reward. Hence, the total reward received by the RL agent corresponds in expected value to the total MAB reward of that episode. These two reward signals not being equal has no significance because the episodes are of constant length (Section 2.5.4).

8.1.1 Basis of Comparison

Before proceeding to analyze how well the RL agents perform on this environment, it helps to look at how basic algorithms perform as a benchmark. Epsilon-greedy and UCB (Section 2.3) are tested on this environment for different values of their respective hyper parameter $\varepsilon \in [0, 1]$ and $c \in [0, 10]$. The cumulative regrets each suffers can be seen in Figure 24.

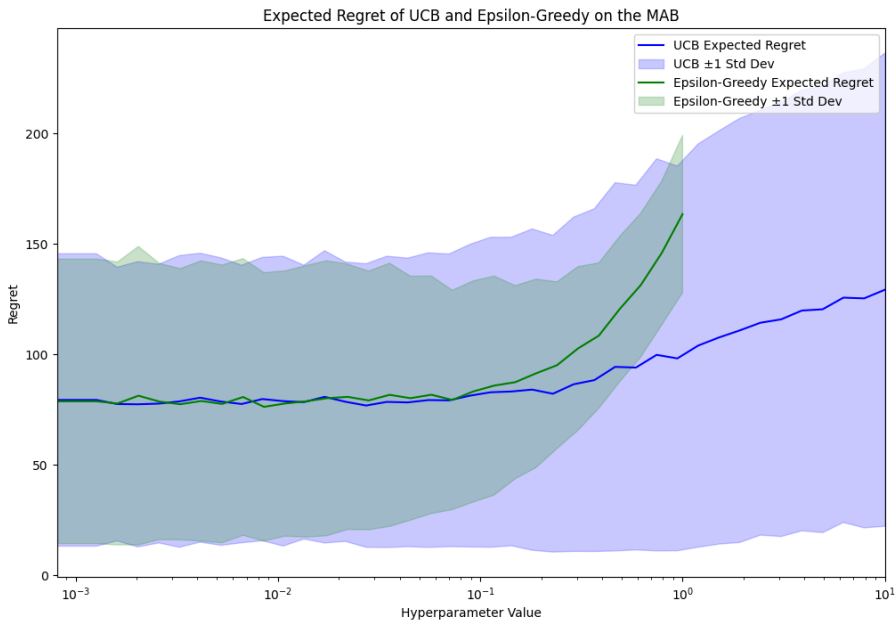


Figure 24: Regret suffered by Epsilon-greedy and UCB for the stochastic MAB

A peculiar thing that one can observe on Figure 24 is that the optimal exploration parameter for Epsilon-greedy and UCB is practically 0, but this should not be interpreted as saying that the algorithms perform no exploration, as both algorithms are set to be optimistic with respect to yet unexplored arms. So they both spend the first 10 steps exploring each arm once before proceeding to take decisions based on their criterion. This means that a significant proportion of the episode (5%) is used for exploration which is quite a lot of exploration. For this reason, the optimal case for both algorithms is to set their hyper-parameter to 0, and both algorithms are equivalent.

8.1.2 RL-Agents

Just like the epsilon greedy algorithm, the RL approaches decide which arm to play at each step. Epsilon greedy relies only on the estimated values of the expected reward of each arm $\hat{\mu}_i$ to choose its samples, and this information is also given to the RL algorithms as part of their observation. However, the RL agents are also provided also with a measure of how uncertain the estimated expected rewards are (just like UCB). This extra information can be thought of as the number of times an arm has been taken in the past as there is a monotonic relationship between the two. The more an arm has been taken, the less uncertainty there is on the estimated expected reward. If an arm has never been taken, the uncertainty about its estimated reward is infinite. The observations received by the RL agents look like the one in Figure 25.

Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Arm 6	Auxiliary
uncert =1.44	2.04	1.02	1.44	∞	∞	Current Round=8
$\mu=1$	-1	1.5	-1	∞	∞	Rounds Left=192

Figure 25: Example observation matrix for the MAB problem with 6 arms and 200 rounds.

On Figure 25, the yet unvisited arms are assigned an estimated mean $\hat{\mu}$ and an uncertainty of infinity, though the values used instead of infinity in practice were values slightly higher than the highest possible value attainable for that variable ($-1 < \mu_i < 1$ so unvisited arm $\hat{\mu}_i := 1.2$). This is done to avoid saturating the neural networks and to avoid having exploding or vanishing gradients. Normalization of the input data is also tested which essentially achieves the same goal of preventing some entries of the observation dominating the other by their relative size difference.

The stochastic MAB problem at hand also has the same type of symmetry (equivariance) between its different arms as does the expert advice MAB studied in Section 5.4. In that each arm is equivalent to the other arms in distribution over a series of episodes. The number associated to each arm is thus irrelevant and serves only as an identifier for that arm, just as it was the case with the experts in the expert advice MAB among the experts who were initialized for each episode as stochastic classifiers whose parameters were assigned according to the same distribution regardless of the expert. Therefore, in order to test if exploiting this symmetry by pre-ordering the observation space of the RL agent so as to reduce the observation space is useful, we compare RL algorithms whose observations are unordered, ordered with respect to estimate mean and ordered with respect to the number of visits.

To evaluate the effectiveness of exploiting this symmetry by pre-ordering the observation space to reduce its size, we compare reinforcement learning algorithms with three different observation configurations: **unordered observations**, **observations ordered by estimated mean reward**, and **observations ordered by the number of visits**.

The training progress of these methods can be seen in Figure 26 where the effectiveness of normalization on the observations of the RL agents can also be seen.

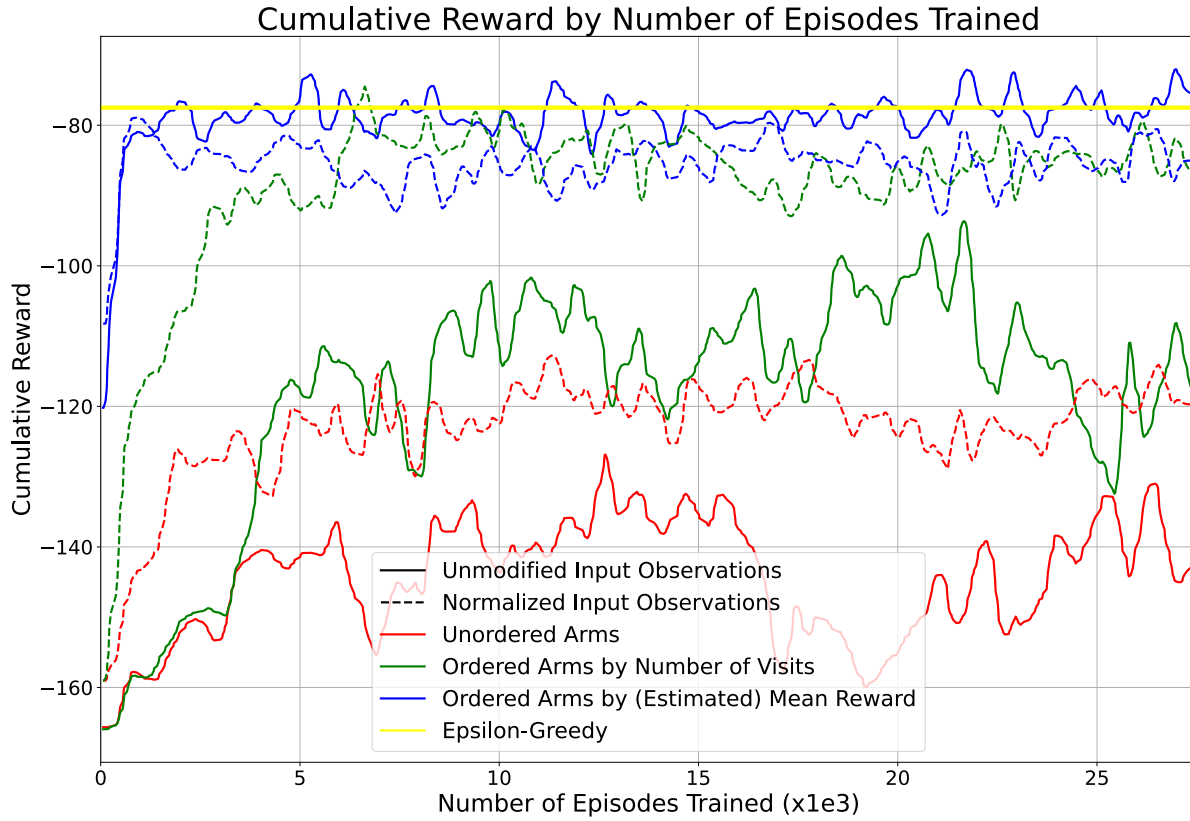


Figure 26: Training curves for the RL algorithms based on the presence of ordering and normalization.

All agents essentially start off by taking random decision and have the same initial mean regret. The blue curves for the estimated mean reward ordering agents appearing to start higher is due to a filtering by a triangular window applied to the curves to smooth out high noise of the curves (caused by the randomness of the stochastic bandit).

A few key observations can be made with the plot in Figure 26. The one is that ordering does seem to positively impact the performance of the agents by a significant amount, as the agents who receive **unordered observations** fail to make sense of the vast space of observations given to them. This inability to make sense of the observation space is pushed even further by the choice of the neural network architecture of two fully connected hidden layers of 64 nodes each for both the actor (policy) and the value networks (arguably not an excellent choice for this problem but is the default PPO architecture in the library used to generate the models).

Ordering the arms with respect to the estimated means of the arms pushes the performance of the PPO agents to that of Epsilon-greedy quite rapidly. This is unsurprising as not only does the fact of ordering reduce the size of the observation space for more efficient sampling, but also more importantly communicates the action preference of Epsilon-greedy. This causes the PPO agent to quickly learn to always pick whatever choice Epsilon-greedy (with $\varepsilon = 0$) made and not doing anything else for this environment. I do not know if such PPO algorithms can learn to sometimes deviate from being purely greedy for scenarios where some level of exploration (apart from the first few steps) is necessary for success, but it yields to optimal results for this environment.

Ordering the arms with respect to the number of times visited also has the advantage of reducing the observation space size, but does not communicate which arm Epsilon-greedy would take. As such, learning is much slower compared to the second case. Interestingly, this method of ordering requires the normalization of the input observations in order to reach the same level of performance as that of the other ordering technique.

Another peculiar observation to make is that the fact of normalizing the input observations appears to slightly negatively affect the performance of agents who receive ordered observations. Essentially causing the algorithm to not always be fully greedy and to lose some level of performance just as is the case with the green curves. An initial guess on why this might be the case is that normalization reduces the level of domination the estimated

means have relative to the other terms of the observation space. Though more inspection reveals this not to be the case as the normalization of the observations in this problem actually does the opposite, as the auxiliary observation terms of current round and number of rounds left as well as the uncertainty data is scaled down more than than the estimated current means. The cause is more likely to be a lack of entropy exploration which might halt the exploration of the PPO algorithms (on-policy method). Though it should be stated that in the ordered scenario, any entropy regularization is likely to lead to non-greedy behavior, causing more regret to be suffered by the agent.

The conclusions to be made from this small experiment are that symmetry exploitation can help make the agents learn better policies, and it can do so better if the ordering of the arms is somehow relevant to the underlying arms of the problem.

Deep RL based policies may be extended to more complex MAB problems with relevant observations and reward signals.

8.1.3 Parameters of the PPO algorithms

The PPO agents used to generate Figure 26, were trained with: two separate networks for policy and advantage networks each with two hidden layers of 64 nodes and tanh activation functions. The policy network has 10 output nodes corresponding to the action probabilities of the policy after a softmax.

- learning rate: $3e-4$
- batch size: 64
- number of epochs: 10
- gamma, discount factor: 1.0
- lambda, generalized advantage estimator lambda: 0.95
- clip range: 0.2
- entropy coefficient: 0.0
- maximum gradient norm: 0.5

8.2 Code

The library relied upon to implement, simulate and train agents on the environments of this thesis was the OpenAI gym framework (gymnasium). The PPO agents were instantiated with the custom architecture (Section 5.6) using the stable baselines 3 and were trained on the same framework.

The code for the environments of Approaches 1,2 and 3 and their ordered versions when applicable can all be found online: Arman EPL GitHub .

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl