Volume 41(2), pp. 1–17
http://orion.journals.ac.za

ORiON
ISSN 0259–191X (print)
ISSN 2224–0004 (online)

Weighted composition for entropy-regularised reinforcement learning

Caston Nyabadza*, Benjamin Rosman*, Steven James* and Geraud Nangue Tasse*

Received: 12 October 2025; Accepted: 8 October 2025

Abstract

One avenue for creating a generally intelligent agent is to equip it with the ability to combine its previously learned behaviours to generate new ones. Currently, most reinforcement learning approaches focus on learning task-specific skills with little consideration for learning skills that can later be composed to solve new problems immediately. In this work, we consider the setting where an agent is required to compose its existing skills to attain goals with different preferences. In particular, we show how this problem can be formulated as a contextual bandit and demonstrate how relevant algorithms can learn the optimal weighted composition of skills given a context. Experiments in a high-dimensional environment indicate that an agent operating in our framework is capable of learning these weights directly from data autonomously.

Key words: Reinforcement learning, Composition, Multi-task learning, Multi-armed bandits.

1 Introduction

Recent advances in reinforcement learning (RL) have led to great success in a variety of narrowly-defined tasks, ranging from two-player board games to robotic manipulation [24]. However, the primary focus of these works is on solving a single challenging task, but such solutions typically cannot be leveraged elsewhere. By contrast, there has been less focus on composing these learned abilities to construct solutions to new problems [4].

One line of work that seeks to address this issue is that of *composition*, which allows for previously learned skills to be combined to solve a new task without explicitly learning new behaviours from scratch [5]. This approach has several advantages. For instance, it results in a combinatorial explosion of all the possible tasks an agent can perform [2]. Empirically, it is also often more efficient for an agent to learn new behaviours by composing existing skills instead of training from scratch [5].

*School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa, email: Benjamin.Rosman1@wits.ac.za
http://dx.doi.org/10.5784/41-2-002

For example, consider an autonomous vehicle that has learned behaviours to (1) pick up passengers, (2) avoid traffic, and (3) refuel. The autonomous vehicle should be able to perform e-hailing services without learning from scratch by simply composing all three existing behaviours. Previous work has demonstrated how to achieve such composition by evenly weighting the existing skills [2]; however, this assumes that all behaviours are equally desirable, which is not always the case. Returning to our example, we might prefer avoiding traffic during rush hour or refuelling when the fuel tank is almost empty. These preferences depend on the *context* the agent finds itself in and can be encoded as weights assigned to the existing behaviours.

The main challenge we address is how to adequately perform *weighted* composition, where the optimal weights depend on the context the agent finds itself in. We propose a reinforcement learning (RL) framework for learning to perform weighted composition given a context, such that the agent maximises the returns it receives when executing the resulting behaviour. We complement the results of previous methods [2] by exploring efficient ways of learning these optimal weights for the composition. Specifically, we formulate the setting as a multi-armed bandit problem where an agent selects weights and observes the rewards after executing the composed behaviour with these weights in a given context.

While our proposed framework is agnostic to the learning algorithm, we investigate three approaches. The first two approaches do not require that the context be provided to the agent and instead attempt to learn the weights directly from feedback from the environment by either (1) discretising the values in the weight space or (2) treating the weight space as a continuous variable. The third approach learns weights conditioned on the context provided to the agent. We test several learning algorithms, such as Softmax [21], Hierarchical Optimistic Optimisation (HOO) [22], and Linear Upper Confidence Bound (LinUCB) [18] in a high-dimensional environment.

Our results indicate the efficiency of Softmax, HOO and LinUCB algorithms compared to the other armed bandit algorithms. The Softmax algorithm was compared to the Upper Confidence Bound (UCB), Exponential-weight (Exp3) and Thomson Sampling bandit algorithms in the dicretised bandit setting, the HOO was compared to the Zooming continuous bandit algorithm in the continuous bandit setting and the LinUCB was compared to the Linear Thompson Sampling (LinTS) bandit algorithm in the context-based bandit setting.

The rest of the paper is organised as follows. Section 2 provides an introduction to RL and compositional RL and briefly discusses key concepts. Section 3 discusses the armed bandit frameworks that will enable the RL agent to explore the weights for composition until the optimal weights are identified. Section 4 evaluates the proposed methods through experimentation. Section 5 concludes this paper with some final remarks.

2 Background

2.1 Reinforcement learning

RL involves an agent interacting with an environment to learn to choose actions that contribute towards its desired goal. Environments are often formulated as Markov Decision

Processes (MDPs), characterised by the tuple $M = \langle S, A, \rho, r, \gamma \rangle$. Here, S is the state space that represents all possible states of the environment. The action space, A, is all the possible actions the agent can execute in a given state. The transition probability, ρ, is the probability that the agent will move from one state to another under a given action. $r(s, a)$ is the immediate reward received after executing action a in state s. The discounting factor $\gamma \in [0, 1]$, discounts rewards exponentially the further they are into the future [6].

The agent's policy π determines which action $a \in A$ is taken when in any observed state $s \in S$. The optimal policy is one that maximises the accumulated reward by the agent over time. During training, at every step, the agent observes a state, selects an action according to the policy, executes it and receives a reward. To gauge the quality of the agent being in the state $s \in S$ or choosing action $a \in A$ when in the state $s \in S$, it is important to assign a value to that state or state-action pair. State or state-action values are represented by state-value functions and state-action value functions (Q-functions), respectively. The agent typically wishes to maximise the expected return G_t, representing the sum of discounted rewards received by an agent starting at timestep t. The state-value function $V_\pi(s_t)$ is simply the expected total return G_t when beginning in state s_t and using policy π to select actions from there onwards:

$$V_\pi(s_t) \;=\; \mathbb{E}[G_t | S = s_t] \;=\; \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | S = s_t \right]. \tag{1}$$

Similarly, the state-action value or Q-function is the cumulative reward G_t for choosing action a_t when in state s_t and thereafter following policy π:

$$Q_\pi(s_t, a_t) \;=\; \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | S = s_t, A = a_t \right]. \tag{2}$$

These value functions are subsequently used to organise and structure the search for better policies [6]. By definition, a policy π_2 is better or equal to π_1 if and only if the expected returns are greater or equal to that of π_1 for all the states. There is always at least one optimal policy π_*, that is better or equal to all the other policies [6]. It is, therefore, possible that multiple optimal policies exist but they will all have the same optimal value function because policy π_* will have a value function $V_* \geq V_{i \forall n}$ for all $s \in S$.

$$V_*(s) \;=\; \max_\pi V_\pi(s) \tag{3}$$

for all $s \in S$.

$$Q_*(s, a) \;=\; \max_\pi Q_\pi(s, a) \tag{4}$$

for all $a \in A$ and $s \in S$.

2.2 Entropy-regularised reinforcement learning

Although the purpose of RL is to determine an optimal policy, which maximises the discounted cumulative returns in an MDP, that policy is deterministic [7]. However, deterministic policies are bad at coping with unseen or uncertain situations [8]. To help resolve this limitation, *entropy regularisation* is a technique that can be used to improve policy optimisation by encouraging the selection of more stochastic policies [9].

Entropy-regularised RL makes it preferable to learn all feasible optimal actions which ensure a robust policy function [10], as it encourages exploration and avoids getting stuck in local optima [11]. This is useful when there is a need to transfer skills to solve unseen tasks [3]. Instead of learning the best way to perform a particular task, the policy tries to learn all possible ways of performing that task [12]. Unlike the standard RL objective, which seeks to maximise only the sum of cumulative rewards, the objective of entropy-regularised RL is to use entropy as a regulariser to encourage the agent to maintain a diversity of solutions. Key advantages are that it encourages the policy to explore widely, for it to capture multiple modes of optimal behaviour and improve learning speeds [5].

Such properties are highly favourable in a complex environment. Entropy is usually taken as the Kullback-Leibler (KL) divergence between the current policy π and some fixed reference policy $\bar{\pi}$, which is typically the uniform random policy. The KL divergence serves as a penalty term in the reward function to encourage the current policy to remain "close" to the reference policy. The above entropy-regularisation is defined by an n-step entropy-regularised value function from state s which follows policy π as shown in Equation 5 [2]:

$$V_{\pi,n}(s) \;=\; \mathbb{E}^{\pi}_{s,n}\left[\sum_{t=0}^{n-1} r(s_t,a_t) - \tau KL[\pi_{s_t}||\overline{\pi}_{s_t}]\right], \tag{5}$$

where τ is a positive scalar that governs how much of an effect the regularising term has [2]. The entropy-regularised value function is then defined as the infinite horizon value function, which represents the total expected returns after executing π from s as shown in Equation 6 [2]:

$$V_{\pi}(s) \;=\; \lim_{n\,\to\,\infty} V_{\pi,n}(s) \tag{6}$$

The corresponding entropy-regularised Q-function is shown in Equation 7:

$$Q_{\pi}(s,a) \;=\; r(s,a) \;+\; \int_{S} V_{\pi}(s')\rho(a,s)(ds'). \tag{7}$$

In order to learn optimal entropy-regularised value functions, standard RL algorithms such as policy iteration or Q-learning can be modified to perform updates based on the returns in Equation 5 [13].

2.3 Weighted composition

Composition allows for multiple primitive policies used to solve specific tasks to be combined to create a composed policy that simultaneously solves these tasks [5]. The main

aim of compositionality is to use already learnt primitive tasks to solve for any new skill without the agent having to learn from scratch [2, 1].

If we let $\mathbf{r}$ be a vector of all tasks' reward functions, and $\mathbf{Q}^*$ be the corresponding optimal action-value functions, then each new composed task can be described by the reward function:

$$r(s,a) \;=\; \tau \log \left(\left|\left| e^{\left(\frac{\mathbf{r}(s,a)}{\tau}\right)} \right|\right|_{\mathbf{w}} \right), \tag{8}$$

where $\mathbf{w}$ is the weighted 1-norm, which contains the weights for each task that is part of the composition and τ is a positive scalar temperature parameter [2]. These weights are arbitrary and can be used to assign priority to different tasks. Van Niekerk et al. [2] show that the optimal action-value function is given by:

$$Q^*(s,a) \;=\; \tau \log \left(\left|\left| e^{\left(\frac{\mathbf{Q}^*(s,a)}{\tau}\right)} \right|\right|_{\mathbf{w}} \right). \tag{9}$$

3 Problem Formulation

More formally, this work is concerned with developing an RL framework that can be used to efficiently determine the correct weights to compose base tasks through a weighted combination over existing value functions and for any given context. In other words, for a given task, can an agent determine the values of $\mathbf{w}$ so as to maximise the cumulative rewards for that task?

Given that $\mathbf{w}$ is a set of non-negative weights, with $\|\mathbf{w}\|_1 = 1$, our main insight is that we can frame this challenge as a multi-armed bandit (MAB) problem. In particular, we formulate a meta decision problem, where the actions consist of potential weights $\mathbf{w}$ to be selected, and returns are based on acting using the value function that results from Equation 9.

As in all MAB problems, it is vital to determine the effects of finding a good balance between exploration and exploitation [16]. Each weight vector $\mathbf{w}$ is associated with an unknown reward distribution and a mean value for that distribution. We either explore new weights or exploit weights that are already known to have high returns at each step taken (see Algorithm 1). At every step t, the agent selects weights $\mathbf{w}$ using a particular MAB method.

We then compose the base skills with these selected weights $\mathbf{w}_t$ according to Equation 9. This produces an action-value function that can be used to generate an episodic trajectory. The returns from this trajectory serve as a learning signal to the MAB algorithm, which then updates its policy over weights.

Algorithm 1 Multi-armed bandit for weighted composition

Input : Multi-armed bandit algorithm $\mathcal{A}$
for $t = 1, 2, ..., T$ **do**
  Select $\mathbf{w}_t$ according to $\mathcal{A}$.
  Compute Q_t using Equation 9 with $\mathbf{w}_t$.
  Execute policy derived from Q_t and record episodic return R_t.
  Update $\mathcal{A}$ with feedback $\{\mathbf{w}_t, R_t\}$
end for

Given this formulation, several questions arise:

- The weight space, and hence the number of arms, is continuous, necessitating specialised MAB algorithms. Does discretising the weight space still result in good performance while improving sample efficiency?
- Alternatively, is it advantageous to allow the MAB algorithm to select from the (continuous) set of all possible weights?
- If the agent is provided with some context that indirectly informs the desirable weights, can this be leveraged to improve performance?

4 Multi-armed bandit algorithms

The multi-armed bandit for weighted composition algorithm (see Algorithm 1) determines the composition of base policies using a computed weight vector, $\mathbf{w}_t$, at each time step t. In this section, we present and analyse the selection strategies employed by various finite, continuous, and contextual multi-armed bandit algorithms.

4.1 Finite armed bandit

For the finite armed bandit algorithm, the learner has a set of discrete weights from which it will choose. The finite armed bandit algorithms to be evaluated are the UCB, Softmax, EXP3 and Thomson Sampling. The UCB algorithm maintains the number of times that each weight $w \in W$ has been selected, denoted by $N_{w,t}$, in addition to the empirical means $\hat{\mu}_{w,t}$. At time t, the algorithm greedily picks the weight w_t [25] as follows:

$$w_t \;=\; \underset{w \in W}{\operatorname{argmax}} \left(\hat{\mu}_{w,t} + \alpha \sqrt{\frac{\ln t}{N_{w,t}}} \right). \tag{10}$$

The Softmax algorithm will select the next composition weight with probability proportional to the average empirical rewards associated with each composition weight $w \in W$ following Equation 11, where τ is a regularisation parameter [21].

$$P(w_t) \;=\; \frac{e^{\frac{\hat{\mu}_{w,t}}{\tau}}}{\sum_{j=1}^{W} e^{\frac{\hat{\mu}_{j,t}}{\tau}}}. \tag{11}$$

Similar to Softmax, the Exponential Weight algorithm for exploration and exploitation (Exp3) is also a probability-matching method. The Exp3 method selects the next composition weight w_t with probability proportional to the "*weights*" ω of all the K composition weights $w \in W$ [26] (see Equation 12). The regularisation parameter is γ. After every step, the weights ω_k of every composition weight $w \in W$ are updated based on the cumulative rewards [27].

$$P(w_t) \;=\; (1-\gamma)\frac{\omega_{w,t}}{\sum_{j=1}^{W}\omega_{j,t}} + \frac{\gamma}{K},\; \forall\, w \in W. \tag{12}$$

The Thompson Sampling (TS) method continuously updates the prior distribution for each composition weight $w \in W$ based on the rewards obtained from the composition weights selected in previous rounds. The general TS algorithm seeks to produce a sample parameter $\hat{\mu}$ for each weight w_k using posterior distributions $P(\hat{\mu}|D)$ where D is the data generated i.e. rewards obtained from the composition weights selected in previous rounds [20]. The weight w_k with the best parameter is selected in every round.

In our setup, we use a Gaussian-Gaussian model: both the likelihood $P(r_t|\mathbf{w}_t, \hat{\theta})$ and the prior $P(\hat{\mu})$ are Gaussian. We assume rewards are i.i.d. Gaussian with unit variance, i.e. $r_t \mid w_t = k \sim \mathcal{N}(\mu_k, 1)$, with μ_k unknown [20]. At time t, using the posterior probability distribution, the agent selects the weight w_t with the best randomly sampled parameter $\hat{\mu}$ [20]. The observed reward $r_t(w_t)$ is used to update both the prior and the new posterior probability distribution for the next round [20].

4.2 Continuous armed bandit

For problems where the number of arms or weights is infinite or larger than the possible number of experiments, it is impossible to explore all the weights [28]. Continuous Armed Bandits Algorithms (CAM) address problems that have infinitely many arms or weights to choose from. In cases where accuracy is paramount, such as the autonomous car example, the safe composition weight could be any real number between 0 and 1. Failure to choose an accurate weight could lead to catastrophic results. We will consider the CAM problem to be stochastic, hence we assume that the reward for each weight w is an independent sample from some fixed distribution with mean μ_w [18]. The continuous armed bandit algorithms to be evaluated are the Zooming Algorithm and Hierarchical Optimistic Optimisation Algorithm.

4.2.1 Zooming algorithm

With the Zooming algorithm, weights are randomly sampled within the continuous space, in our case $w \in [0, 1]$, creating an artificial k-armed space. Any new weight w covered by the confidence interval of an active arm is discarded. The weights $w \in W$ are evaluated using UCB technique to identify promising weights that will be zoomed into at the expense of less promising weights [18].

4.2.2 Hierarchical optimistic optimisation algorithm

The Hierarchical Optimistic Optimisation (HOO) Algorithm is the least restrictive of all the infinitely many armed bandit algorithms; however, it requires extensive computation time [22]. The HOO is a tree-based CAM algorithm that makes use of a binary tree data structure $\mathcal{T}$ with nodes that are linked to the continuous weight space. As the binary tree is expanded, the nodes connect to measurable sub-areas of the weight space $W \in [0, 1]$ [19]. The HOO uses B-values, which are optimistic estimates and uses the upper confidence bound for each of those sub-areas. The sub-area with the highest B-value is selected at each iteration. B-values for each node are recursively computed to determine the path to the most promising region [19]. At every time step t, the tree traverses from the root node by choosing the nodes with higher B-values between each pair, creating a path P.

The weight at step t is sampled in the region identified by the path P. The **LEAF** function is used to retrieve any arbitrary tree leaf, i.e. the last tree sub-region that can be traversed down the tree following a certain path. More information and the corresponding empirical mean are stored and updated as each node is traversed in succession up to T rounds [19].

4.3 Contextual multi-armed bandit

To make the best decisions, the agent seeks to gather enough information on how the context or the feature vector and rewards of arms played relate to each other [20]. As opposed to the previous armed bandit problems, action or weight features in contextual bandits may be useful to infer the conditional average payoff of a weighted composition, which allows the agent to even improve the average payoff over time. This implies that the rewards $r_t(w_t)$ depend on the weight w_t and context vector x_t.

4.3.1 LinUCB algorithm

The LinUCB is a contextual armed bandit method that allows for rigorous regret bounds and works well in any experimental setup [18]. In this algorithm, a linear payoff function is assumed, i.e. the confidence interval of the parameters can be estimated from the data generated by computing the upper confidence bound (UCB) for the estimated arm payoff [17].

The LinUCB algorithm observes a set of d user features signified as a vector $\mathbf{x}_t$ at every time step t. $\hat{\theta}_w$ is the coefficient estimate for each weight w and it is calculated using a matrix $\mathbf{A}_w$ of size d, initialised as an identity matrix, and $\mathbf{b}_w$ a vector of size d, initialised as a zero-vector of size d (see Equation 13).

$$\hat{\theta}_w \leftarrow \mathbf{A}_w^{-1}\mathbf{b}_w. \tag{13}$$

At every step, the selected weight's coefficient estimate is updated based on the observed features and rewards[21]. The weight that will be selected at every step is one that maximises the regularized upper confidence bound estimate $p_{t,w}$ (see Equation 14) [21]. α is the regulariser.

$$p_{t,w} \leftarrow \hat{\theta}_w \mathbf{x}_t + \alpha\sqrt{\mathbf{x}_t^\intercal \mathbf{A}_w^{-1}\mathbf{x}_t}. \tag{14}$$

4.3.2 LinThompson sampling algorithm

The LinThompson Sampling Algorithm is an extension of the Thomson Sampling Algorithm, however, for the posterior distribution $P(\hat{\mu}|D)$, the data generated includes the reward r_t and a given context x_t at every step t [20].

5 Experiment

This research was conducted using Python 3.12 as the primary programming language, with the development environment being PyCharm Community Edition third-party software (version 2023.2). We utilized the PyTorch (version 2.0) libraries for machine learning tasks and NumPy (version 1.26) for data manipulation and analysis. Data was preprocessed and visualized using Matplotlib (version 3.7). The experiments were run on a Linux workstation with 64 GB of RAM and an NVIDIA GeForce RTX 2070 GPU.

We empirically test these algorithms in a high-dimensional video game, in which an agent must navigate to collect objects of different colours and shapes [2]. The layout is given by Figure 1. The agent is equipped with actions to move in any of the four cardinal directions (provided it does not collide with a wall), as well as a pickup action, and is given as input an 84×84 pixel representation of the game.

The agent is first trained on tasks for searching for the `Purple Circle` and `Beige Square` base tasks separately. From the pixels, the value functions are learnt using Deep Q-Networks (DQN). The resulting DQN networks are collected into a library from which we later compose new Q-functions.

Our experiments show how different bandit algorithms sample different weights in the weight space W to find the optimal weight to collect purple circles $-$OR$-$ beige squares, depending on the reward assigned to each object.

We compare the different bandit algorithms to determine the most efficient ones using the average rewards curve. We identify weight estimates for the context-free bandits. The optimal one becomes the correct weight for the composition. By adding context, the experiments show the efficiency in determining optimal weights for each context, and we compare the contextual bandits.

A reward function linked to the steps an agent takes to reach the target object and the type of object collected is maintained for both the context-free and contextual bandits for each composition. We evaluate the weights using context-free and contextual armed bandit algorithms, and in each iteration, the agent chooses between weights. For each of the k-armed and contextual k-armed bandits, we select the best parameters as per the literature (see Table 1) and run them across five seeds. Each episode has a maximum step limit of 1000, which begins by randomly positioning the agents and the objects in the environment. We determine the best weight w that gives the best rewards averaged out over the different seeds for each parameter. To determine the best output for each algorithm, we assess the different parameters and identify the best weight w to compose.

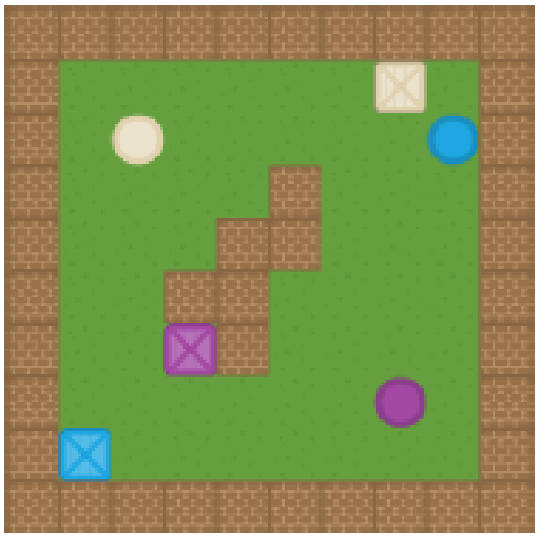


Figure 1: Layout of the video game. The position of the agent and objects are randomly sampled for each episode, and the observations are RGB pixel images.

Algorithm	**Parameters**	**Values**
UCB	α	$[0, \sqrt{1}, \sqrt{2}, \sqrt{3}, \sqrt{4}]$
Softmax	τ	$[0.1, 0.2, 0.3, 0.4, 0.5]$
Exp3	γ	$[0.1, 0.2, 0.3, 0.4, 0.5]$
TS	initial μ	$[0.1, 0.2, 0.3, 0.4, 0.5]$
LinUCB	α	$[0, \sqrt{1}, \sqrt{2}, \sqrt{3}, \sqrt{4}]$
LinTS	R	$[0, 0.01, 0.5, 1.0, \sqrt{2}]$
Zoom	α	$[0, \sqrt{1}, \sqrt{2}, \sqrt{3}, \sqrt{4}]$
HOO	v_1	$v_1 \in [0.2, 0.4, 0.5, 0.7, 0.9]$
HOO	ρ	$\rho \in [0.8, 0.6, 0.5, 0.4, 0.3]$

Table 1: Parameter table.

5.1 Finite armed bandits analysis

We pay attention to the weight w estimates as well as the arm selection strategy through the arm distributions of the proposed method and the methods used as a comparison. We test the proposed Softmax method and compare it to the UCB, Thompson Sampling and Exp3 (see Figure 4). The results of all the finite-armed bandit algorithms show that the Softmax algorithm is the best finite-armed bandit algorithm with a higher average reward compared to other finite algorithms (see the shaded average and variance graph in Figure 2). The Softmax algorithm was run over the different parameters and the parameter that gives the best output was determined experimentally (see Figure 5). If we cross-reference to the Softmax arms selection strategy, the best weight for the selected composition is 0.5 for each object to be collected. These weights are also similar to what is determined by the other methods (see Figure 4). However, we note that the Softmax method spends the least time sampling sub-optimal arms compared to the rest of the algorithms (see Figure 4c), which shows that it is more efficient.

5.2 Continuous armed bandits analysis

In the quest to improve the accuracy of the weights above, HOO continuous armed bandit method performed best when the parameters are set to $v_1 = 0.2$ and $\rho = 0.8$ (see Figure 6a), which were determined experimentally. Figure 6b, therefore indicates that the best weight estimate is $w = 0.493\,84 \pm 0.086\,04$. The HOO, when compared to the Zooming continuous armed bandit algorithm, performs much better than the Zooming method (see the shaded average and variance reward graph in Figure 2). The HOO and Zooming continuous bandits perform much better than the finite algorithms because they are more specific and give better average rewards (see the shaded average and variance reward graph in Figure 2), implying that the given weights are more accurate than those obtained by the Softmax method.

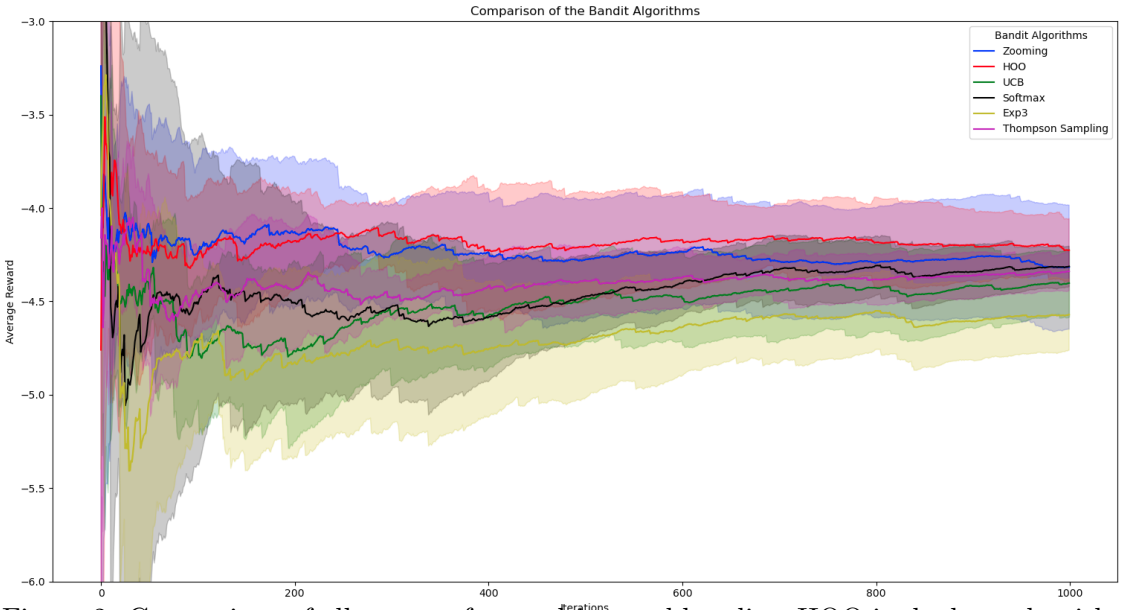


Figure 2: Comparison of all context free multi-armed bandits. HOO is the best algorithm with the highest average rewards.

5.3 Contextual bandits analysis

In the contextual bandit setup, the goal is to determine the correct weight that will be used to determine the preference of the agent towards collecting the beige squares or purple circles after having observed a given feature vector. Synthetic user features of 100 dimensions are randomly generated, each feature on a scale between 0 and 5. A slightly different reward function to the context-free bandits is used, but still randomly generated and not known to the agent. This still makes it a stochastic contextual bandit problem.

The proposed LinUCB contextual bandit method, also had the optimal parameters being determined experimentally. Figure 7 shows the different effects the LinUCB method has on the composition of different entropy-regularised RL policies. It converges slightly faster than any of the context-free contextual bandit setups, compare Figure 2 and Figure 7. Although for a standard 1000 iterations, all the algorithms' average rewards curves assume

an almost linear curve, it is well defined for the LinUCB method (see Figure 7). There is almost no significant difference brought by the different parameters used as the lines almost lie on each other.

Comparing the LinUCB and Lin Thompson Sampling shows that the LinUCB algorithm performs much better than the Lin Thompson Sampling algorithm (see the shaded average and variance reward graph in Figure 3). The LinUCB selects the best weights w in any given context most of the time compared to the LinThompson Sampling algorithm, hence the best average curve.

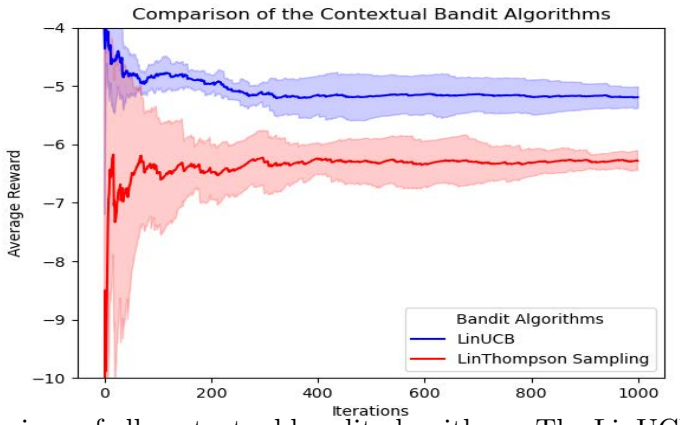


Figure 3: Comparison of all contextual bandit algorithms. The LinUCB outperformed the LinThompson Sampling algorithm.

6 Conclusion

In this paper, we discuss a bandit-based framework that can be used to efficiently evaluate the correct weights to compose an entropy-regularised value function given a context. Our experiments show that the Softmax algorithm is the most efficient context-free finite armed bandit algorithm and provides a better estimate of the correct weight for composition better than UCB, Thompson Sampling and Exp3. The HOO is the most efficient context-free continuous bandit algorithm compared to the Zooming algorithm. Where precision and accuracy are required, continuous armed bandits outperform the finite armed bandits as they are able to maximise the cumulative rewards thereby making the HOO the best method to evaluate the optimal weight to compose for context-free bandits. Introducing the context implies that the required optimal weight for any given context has to be determined and the LinUCB contextual armed bandit algorithm outperforms the LinTompson Sampling making it the best method for determining the optimal weights that are dependent on the context. By making use of the context, the contextual bandit algorithms converge faster than the context-free algorithms and are more stable due to limited variations.

References

[1] Tasse G. N, James S & Rosman B, 2020, *A boolean task algebra for reinforcement learning*, arXiv preprint arXiv:2001.01394, 790–799.

[2] van Niekerk B, James S and Earle A & Rosman B, 2018, *Will it blend? composing value functions in reinforcement learning*, arXiv preprint arXiv:1807.04439.

[3] Zhao R, Sun X & Tresp V, 2019, *Maximum entropy-regularized multi-goal reinforcement learning*, arXiv preprint arXiv:1905.08786.

[4] Kaelbling LP & Lozano-Pérez T, 2017, *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 886–893.

[5] Haarnoja T, Pong V, Zhou A, Dalal M, Abbeel P & Levine S, 2018, *Composable deep reinforcement learning for robotic manipulation*, IEEE, 6244–6251.

[6] Sutton RS & Barto AG, 2018, *Reinforcement learning: An introduction*, MIT press.

[7] Yang W, Li X & Zhang Z, 2019, *A Regularized Approach to Sparse Optimal Policy in Reinforcement Learning*, Advances in Neural Information Processing Systems, 5938–5948.

[8] Husain H, Ciosek K & Tomioka R, 2021, *Regularized Policies are Reward Robust*, International Conference on Artificial Intelligence and Statistics, PMLR, 64–72.

[9] Ahmed Z, Roux NL, Norouzi M & Schuurmans D, 2018, *Understanding the impact of entropy on policy optimization*, arXiv preprint arXiv:1811.11214.

[10] Lee K, Choi S & Oh S, 2018, *Sparse Markov decision processes with causal sparse Tsallis entropy regularization for reinforcement learning*, IEEE Robotics and Automation Letters, Vol. 3(3), 1466–1473.

[11] Xin B, Yu H, Qin Y, Tang Q & Zhu Z, 2020, *Exploration Entropy for Reinforcement Learning*, Mathematical Problems in Engineering.

[12] Haarnoja T, Tang H, Abbeel P & Levine S, 2017, *Reinforcement learning with deep energy-based policies*, Proceedings of the 34th International Conference on Machine Learning, Vol. 70, 1352–1361.

[13] Schulman J, Chen X & Abbeel P, 2017, *Equivalence between policy gradients and soft q-learning*, arXiv preprint arXiv:1704.06440.

[14] Qureshi AH, Johnson JJ, Qin Y, Henderson T, Boots B, & Yip MC, 2020, *Composing task-agnostic policies with deep reinforcement learning*, ICLR.

[15] Todorov E, 2009, *Compositionality of optimal control laws*, Advances in neural information processing systems, Vol. 22, 1856–1864.

[16] Bouneffouf D & Rish I, 2019, *A survey on practical applications of multi-armed and contextual bandits*, arXiv preprint arXiv:1904.10040.

[17] Li L, Chu W, Langford J & Wang X, 2011, *Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms*, Proceedings of the fourth ACM international conference on Web search and data mining, 297–306.

[18] Slivkins A, 2019, *Introduction to multi-armed bandits*, arXiv preprint arXiv:1904.07272.

[19] Bubeck S, Munos R, Stoltz G & Szepesvári C, 2019, *X-Armed Bandits.*, Journal of Machine Learning Research, Vol. 12(5).

[20] Agrawal S & Goyal N, 2013, *Thompson sampling for contextual bandits with linear payoffs*, International Conference on Machine Learning, PMLR, 127–135.

[21] Burtini G, Loeppky J & Lawrence R, 2015, *A survey of online experiment design with the stochastic multi-armed bandit*, arXiv preprint arXiv:1510.00757.

[22] Mattos D, Mårtensson E, Bosch J & Olsson H, 2018, *Optimization experiments in the continuous space*, In International Symposium on Search Based Software Engineering, Springer, 293–308

[23] Kalvit A & Zeevi A, 2020, *From finite to countable-armed bandits*, Advances in Neural Information Processing Systems, volume=33, 8259–8269

[24] Yu T, Quillen D, He Z, Julian R, Hausman K, Finn C & Levine S, 2020, *Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning*, Conference on robot learning, 1094–1100

[25] Kuleshov V& Precup D, 2014, *Algorithms for multi-armed bandit problems*, arXiv preprint arXiv:1402.6028, 1402.6028

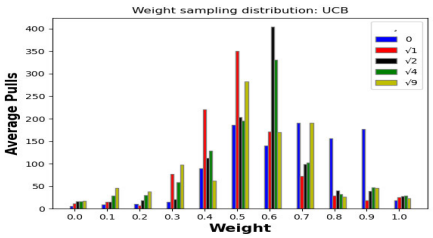
[26] Seldin Y, Szepesvári c, Auer P& Abbasi-Yadkori Y, 2013, *Evaluation and analysis of the performance of the EXP3 algorithm in stochastic environments*, European Workshop on Reinforcement Learning, 103–116

[27] Vermorel J& Mohri M, 2005, *Multi-armed bandit algorithms and empirical evaluation*, European conference on machine learning, 437–448

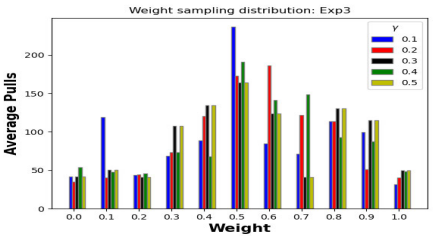
[28] Wang Y, Audibert J& Munos R, 2005, *Algorithms for infinitely many-armed bandits*, Advances in Neural Information Processing Systems

[29] Mnih V, Kavukcuoglu K, Silver D, Rusu D, Andrei A, Veness J, Bellemare M, Marc G, Graves A, Riedmiller M, Fidjeland A & Ostrovski G, 2015, *Human-level control through deep reinforcement learning*, nature, 529–533

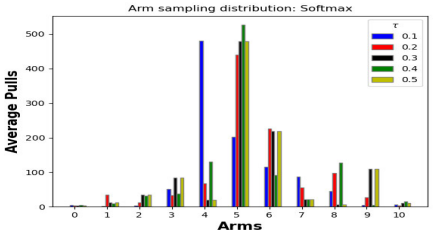
Appendix



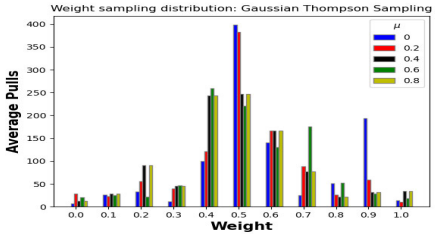
(a) UCB.



(b) EXP3.



(c) Softmax arm.



(d) Thompson Sampling.

Figure 4: Arms selection distributions. The Softmax method spends the least time sampling sub-optimal arms compared to the rest of the algorithms, further emphasising its efficiency.

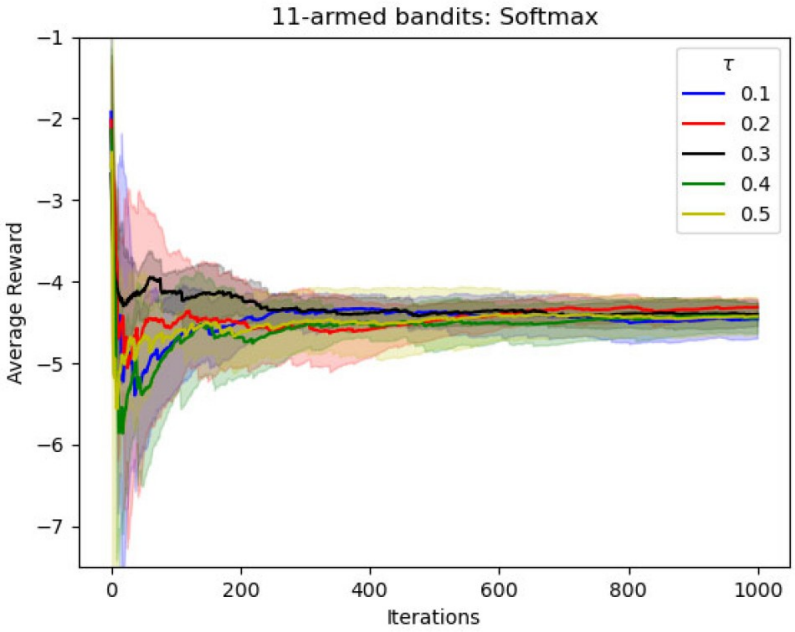
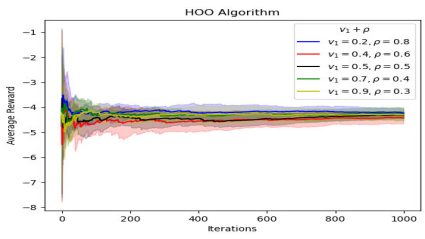
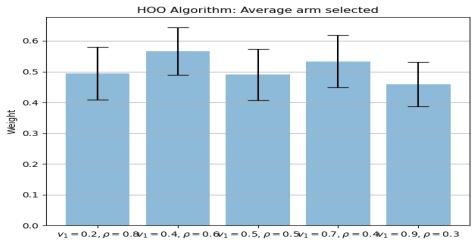


Figure 5: The Softmax average reward curves. The regularisation parameter τ setting of 0.2 gives the best Softmax returns.

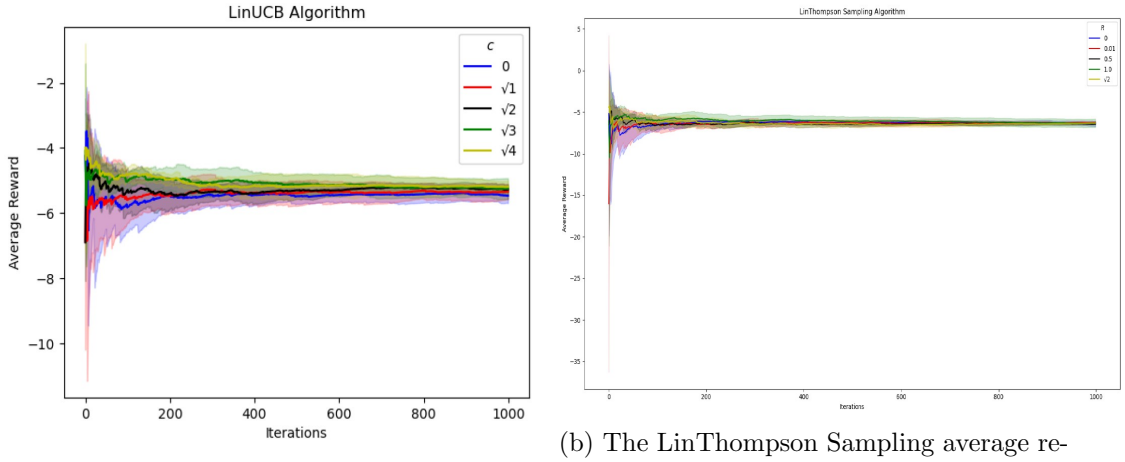


(a) Average HOO reward curves. The parameters are real numbers $v_1 > 0$ and $\rho \in (0, 1)$ and the best return was obtained when v_1 was set at 0.2 and ρ was set at 0.8



(b) Optimal HOO Arm Estimate: The parameter settings ($v_1 = 0.2$ and $\rho = 0.8$) gave the best returns, therefore the best weight estimate is $w = 0.493840.08604$

Figure 6: HOO Algorithm: (a) Reward curve and (b) the estimated weights w for the different parameters.



(a) Average LinUCB reward curves.

(b) The LinThompson Sampling average reward curves.

Figure 7: There is almost no significant difference brought by the different parameters used as the lines almost lie on each other. (a) The LinUCB algorithm performed best when the regularisation parameter α was set to $\sqrt{4}$. (b) The LinThompson algorithm performed best when the regularisation parameter R was set to 0.01.