Imperial College London

Imperial College London

Department of Computing

# Habit Reinforcement Learning

*Author:*
Petros Christodoulou

*Supervisor:*
Dr A. Aldo Faisal

Submitted in partial fulfillment of the requirements for the MSc Computing (Machine Learning) degree of Imperial College London

October 5, 2019

## Acknowledgements

I would like to thank to my supervisor Dr. A. Aldo Faisal for giving me the opportunity to work on this project and providing me with guidance along the way. It would not have been as interesting and rewarding a project without him. I would also like to thank my fellow students for their support and discussions. Finally I would like to thank Imperial as a whole for such a great year.

Thank you!

## ABSTRACT

Habits strongly influence behaviour and likely facilitate faster learning. Yet the concept of habits is almost completely absent from the field of Reinforcement Learning. Habit Reinforcement Learning corrects this by demonstrating the power of habits to dramatically and consistently improve the sample efficiency of Reinforcement Learning agents. It works by using a grammar inference algorithm to infer the "habits" of an agent midway through training. Then it augments the agent's action space with macro-actions representing the habits. We apply this framework to Double Deep Q-Learning (Habit-DDQN) and a discrete action version of Soft Actor-Critic (Habit-SAC) and find that it improves performance in 8 out of 8 Atari games (median improvement 31%, maximum improvement 668%) and 19 out of 20 Atari games (median improvement 96%, maximum improvement 3756%) respectively even without substantive hyperparameter tuning. We also show that Habit-SAC is the model-free state-of-the-art for sample efficiency in 17 out of the 20 Atari games, again even without substantive hyperparameter tuning.

## Contents

## 1 Introduction

Reinforcement Learning (RL) has famously made great progress in recent years, successfully being applied to settings such as board games [Silver et al., 2017], video games [Mnih et al., 2015] and robot tasks [Andrychowicz et al., 2018]. However, widespread adoption of RL in real-world domains has remained slow primarily because of its poor sample efficiency which Wu et al. (2017) see as a "dominant concern in RL".

RL generally has poor sample efficiency because it suffers from the curse of dimensionality - as the dimension of an RL problem in terms of state space, action space and time increase, RL requires exponentially more experience in order to learn effective policies.

Deep learning in general also suffers from the curse of dimensionality. As Arulkumaran et al. (2017) argue however, deep learning in modern times has made great progress addressing the curse by embedding more assumptions (called "inductive biases") into the learning procedure. By injecting extra inductive biases into a learning algorithm we can greatly restrict the hypothesis space (i.e. the space of potential solutions we are searching over). This reduces the flexibility of the learning algorithm but if the extra inductive biases are informative then it does so in a way that rules out proportionally more bad solutions than good solutions which can facilitate dramatically faster learning.

Botvinick et al. (2019) go so far as to argue that extra incorporation of inductive biases has been largely responsible for the modern resurgence of neural networks in artificial intelligence research. Cohen and Shashua (2017) suggest the most successful example of an effective inductive bias is found in convolutional neural networks (CNNs) [LeCun and Bengio, 1995]. Through their use of the translation-invariant convolution operation, CNNs enforce weight sharing across pixels which reduces the flexibility of the network in an informative way that dramatically speeds up learning in image-based tasks. Similarly, recurrent neural networks (RNNs) [Williams, Hinton, and Rumelhart, 1986] enforce the biases that the future can not impact the past and weight sharing across time which combine to significantly speed up learning for sequential tasks.

The experience of deep learning therefore suggests RL could greatly mitigate its curse of dimensionality and improve its sample efficiency if it were able to introduce additional informative inductive biases. The question is what should these inductive biases be?

Turning to biology for inspiration we note that it is well established that human's reward-related actions arise out of the interaction of not one but two systems: one which picks actions to achieve goals and another which *picks actions to execute habits* (i.e. repeat past behaviour) [Dezfouli and Balleine (2010), Dezfouli and Balleine (2012), Dickinson (1994), Watson, Wingen, and Wit (2018)]. It is therefore often the case that we are acting in a certain way not because we explicitly think it will help us achieve a goal but because we are repeating something we have done before. Brushing our teeth and getting our key out to open our front door are examples of two generally habitual behaviours. Habits are so important to behaviour that some even estimate that up to half of human behaviour is habitual [Wood, Quinn, and Kashy, 2002]. We also know that the brain physically facilitates easier habit formation by modifying its synaptic networks to make the repetition of behaviour less effortful over time - Lewis (2016) liken this to the carving of ruts by rainwater in a garden.

It may seem strange at first that the brain encourages habit formation. After all, having a significant part of the brain that simply encourages the repetition of past behaviour must make us less flexible. This is however exactly the point and is why habits can be helpful for learning.

In the language of machine learning, habits are merely an inductive bias towards the past and because our past behaviour is informative it means that by having this inductive bias we restrict our hypothesis space in a way that likely speeds up learning. In other words, having habits restricts our flexibility in a way that likely rules out proportionally more bad behaviour than good behaviour, making it easier for us to learn what the good behaviours are. This is the same reason why a convolutional layer, for example, is less flexible than a fully-connected layer (because it enforces weight sharing) but faster at learning visual tasks.

Clearly, therefore, habits are an extremely important aspect of human behaviour that likely facilitate faster learning. Contrasting this fact with modern RL algorithms however we realise that the concept of habits is almost completely absent from the field of RL. For example, a Q-learning agent [Watkins, 1989] picks actions only in order to achieve the goal of maximising rewards, it incorporates no consideration or bias towards explicitly repeating action sequences that we have chosen before.

This dissertation begins to correct this by developing a new type of RL method called Habit Reinforcement Learning that introduces the concept of habits into the field of RL and shows that they can help speed up learning.

Habit Reinforcement Learning is a form of hierarchical RL that successfully extends and generalises the framework of Lange and Faisal (2018) to continuous state settings. It uses techniques from computational linguistics to infer a "grammar" of action on the basis of the agent's past actions. This "action grammar" summarises the agent's most common pattern of actions i.e. its habits. We then adapt the agent's action space in order to facilitate repetition of these habits. For the first time we demonstrate that, with numerous critical extensions detailed in Section 5.2, the framework can be used to consistently combat the curse of dimensionality and drastically improve sample efficiency in a wide variety of challenging settings.

The **main novel contributions** of this project are:

- The **Habit Reinforcement Learning framework** which can be applied to any off-policy RL algorithm in environments with either discrete or continuous states
- The **Habit Double Deep Q-Learning (Habit-DDQN) algorithm** which improves sample efficiency in 8 out of 8 Atari games with a median improvement of 31% and maximum improvement of 668% even without substantive hyperparameter tuning
- The **Habit Soft Actor-Critic (Habit-SAC) algorithm** which improves sample efficiency in 19 out of 20 Atari games with a median improvement of 96% and maximum improvement of 3756%. We also show that Habit-SAC is the model-free state-of-the-art in terms of sample efficiency in 17 out of 20 Atari games, all even without substantive hyperparameter tuning
- The **Hindsight Action Replay** technique which can dramatically improve information efficiency when using macro-actions in any setting (increasing information efficiency by over 100% in some cases)
- The **Abandon Ship** technique which reduces the variance involved in using macro-actions and enables us to effectively use very long macro-actions (of over length 100 in some cases)
- A derivation and implementation of a **discrete action version of Soft Actor-Critic** that we show is competitive with state-of-the-art off-policy RL algorithms on the Atari suite in terms of sample efficiency

The dissertation proceeds as follows: in *Section 2* we explain some preliminary RL and action grammar concepts that we will refer to later; *Section 3* conducts a review of the hierarchical RL literature as this is the subfield of RL that Habit Reinforcement Learning falls into; *Section 4* derives a discrete action version of the Soft Actor-Critic algorithm that we later use as a base algorithm for Habit Reinforcement Learning; *Section 5* moves on to explaining the Habit Reinforcement Learning algorithm and discussing its results; finally *Section 6* concludes and makes suggestions for future work.

# 2 Preliminary Concepts

## 2.1 Reinforcement Learning

We first explain some preliminary reinforcement learning concepts that we will refer to later.

Following Sutton and Barto (2017), **Reinforcement Learning** is a way of learning what actions to take in order to maximise a numerical reward signal. In particular, it is for situations where the learner is not told what the optimal actions are but must instead discover the best actions to take through a guided process of trial-and-error.

We formalise the problem of reinforcement learning using **Markov Decision Processes (MDPs)**. A MDP is a tuple $(S, A, P, r, \rho_0, \gamma)$ where $S$ is a set of states, $A$ a set of actions, $P : S \times A \times S \to [0, 1]$ a state transition probability function, $r : S \times A \to [r_{min}, r_{max}]$ a reward function, $\rho_0 : S \to [0, 1]$ is the starting state distribution, and $\gamma \in [0, 1]$ a discount rate. The reinforcement learning problem is then to find a policy $\pi_\theta : S \times A \to [0, 1]$ to maximise the expected discounted sum of future rewards $\sum_{t=0}^{T} E_{(s_t,a_t)\sim\tau_\pi}[\gamma^t r(s_t, a_t)]$ where $\tau_\pi$ is the distribution over trajectories induced by policy $\pi_\theta$, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim P(s_{t+1}|a_t)$.

One of the most popular types of RL algorithms used to solve MDPs is **Q-learning** [Watkins, 1989]. Q-learning involves learning a Q-function (also called an action-value function) $Q_\pi : S \times A \to \mathbb{R}$ that estimates the discounted expected reward from being in state $s_t$, choosing action $a_t$ and then following policy $\pi$. A Q-Learning agent usually acts in the world by following an epsilon-greedy policy over the Q-values, meaning that for epsilon proportion of the time it chooses a random action and the rest of the time it chooses the action with the highest Q-value. It then periodically updates its Q-function according to the formula:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t)] \tag{1}$$

where $\alpha > 0$ is a learning rate. Note that we call $R_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t)$ the temporal difference (TD) error.

Mnih et al. (2013) introduced an extension to Q-learning called **Deep Q-learning (DQN)** that later went on to achieve superhuman performance in 23 Atari games [Bellemare et al. (2013), Mnih et al. (2015)]. Q-learning becomes "Deep" Q-learning when we use a function approximator (typically a neural network) to estimate the Q-values rather than maintaining a table of Q-values. It also involves maintaining a replay buffer of experiences observed during training that we randomly sample from periodically to train the function approximator. To stabilise learning, we also then maintain a second neural network called a "target network" that is used to calculate the target Q-values for the main neural network (called the "local network") to update towards. The target network is then either set equal to the local network every $T > 1$ timesteps or every timestep is updated towards the values of the local network by some small percentage (e.g. $1\%$).

Hasselt, Guez, and Silver (2015) provide an extension to DQN called **Double DQN (DDQN)**. They note that ordinary DQN tends to overestimate Q-values because of the max operation in (1) and so to counter this they propose using the local network to calculate the argmax action and then the target network to calculate the target Q-values for this action which we then use to update the Q-function.

Building on the use of a replay buffer in DQN, Andrychowicz et al. (2017) introduce **Hindsight Experience Replay** to improve the data efficiency of any RL algorithm using a replay buffer in multi-goal settings. Hindsight Experience Replay involves replaying each episode with a different goal than what the agent was trying to achieve - in particular, they suggest replaying an episode with the goal that was actually achieved. This lets RL algorithms become more efficient as they can then begin to learn as much from their unsuccessful trials as they do from successful trials. Empirically the authors then show that using Hindsight Experience Replay can dramatically improve sample efficiency in settings with changeable goals.

Building on Q-Learning, Hessel et al. (2017) combined improvements from multiple papers to create an agent called **Rainbow** that achieves a median score of 223% of human performance on 57 games from the Atari suite. Kaiser et al. (2019) suggest Rainbow represents the current state-of-the-art model-free RL algorithm for discrete action settings in terms of sample efficiency.

We now offer some brief details on how Rainbow works without going into too much detail as it is beyond the scope of this dissertation. Rainbow is a Deep Q-Learning agent that incorporates the following extensions: i) Double Q-Learning as described above, ii) Prioritised Replay [Schaul et al., 2016] which involves prioritising experiences with higher TD errors when learning; iii) Dueling Networks [Wang et al., 2016] which is a Q-network architecture where the final layer outputs one value to represent the value of the state and then one value for each action to represent the additional value of playing each action, then to recover the q-values we combine the two types of output; iv) Multi-step learning [Sutton, 1988] which involves bootstrapping after n-steps rather than 1-step when calculating q-values; v) Distributional RL [Bellemare, Dabney, and Munos, 2017] which involves working with an approximation of the distribution of returns rather than only the expected return; and vi) Noisy Nets [Fortunato et al., 2017] which involves injecting noise into the outputs of different layers of the Q-network to encourage state-conditioned exploration.

Besides Q-learning, another popular type of RL algorithm is **Actor-Critic**. Following Sutton and Barto (2017), an "Actor-Critic" RL method is one where we maintain both a policy (called the "actor") $\pi_\theta : S \times A \rightarrow [0, 1]$ and a state-value estimator (called the "critic") $v : S \rightarrow \mathbb{R}$. The agent then learns by iterating between training the critic to predict the value of states under the policy, and training the actor to reach states that the critic predicts have high values.

Haarnoja et al. (2018) introduce the **Soft Actor-Critic** algorithm for continuous action settings that achieves state-of-the-art results on a wide variety of continuous control benchmark tasks. The key innovation in Soft Actor-Critic over ordinary Actor-Critic is that it uses the maximum entropy formulation of reinforcement learning [Ziebart (2010), Neu, Jonsson, and Gomez (2017)]. This means that instead of maximising the expected discounted sum of rewards $\sum_{t=0}^{T} E_{(s_t,a_t)\sim\tau_\pi}[\gamma^t r(s_t, a_t)]$, its RL objective is to maximise the expected discounted sum of rewards *plus the entropy of the policy*:

$$\sum_{t=0}^{T} E_{(s_t,a_t)\sim\tau_\pi}[\gamma^t (r(s_t, a_t) + \alpha\mathcal{H}(\pi(.|s_t))] \tag{2}$$

where $\mathcal{H}(\pi(.|s_t))$ denotes the entropy of an action $a_t$ with distribution $\pi(a_t|s_t)$ and is calculated as $\mathcal{H}(\pi(.|s_t)) = -\log \pi(.|s_t)$. Two of the main benefits of maximising entropy are that it encourages exploration and can act as a form of regulariser by encouraging more robust policies. Soft Actor-Critic then proceeds to learn by alternating between policy evaluation and policy improvement steps. In the policy evaluation step a Q-function is updated to match the current policy and in the improvement step the policy is updated to maximise the Q-function.

Now we explain preliminary action grammar concepts before taking advantage of all of these preliminary concepts to conduct a literature review.

### 2.2 Action Grammars

It is well established that, to flexibly convey meaning, humans use grammatical principles to combine words into larger structures like phrases or sentences [Ding et al., 2012]. In other words there is some structure or hierarchy to language that guides how we use it. An example of an aspect of the structure is that to construct a valid sentence it is always necessary to combine a noun phrase and a verb phrase [Yule, 2015].

**Action grammars** is the parallel idea that there is an underlying set of rules for how we combine actions that we use to hierarchically produce new actions in a similar way to how we produce new sentences. There is growing biological evidence for this link between language and action as explained in Pastra and Aloimonos (2012).

The concept of action grammars is particularly interesting in the context of habits because it seems that if we were able to understand the underlying structure or grammar of action then we could use this to easily identify any useful habits.

Additionally, to uncover this grammar of action we may be able to leverage pre-existing research from the field of computational linguistics. In computational linguistics, *grammar inference* is the problem of inferring the formal grammar of a language from a set of sentences. This problem has received much attention over the years and so numerous algorithms have been developed around it. If we see the problem of inferring an action grammar as analogous to the problem of inferring a grammar of language (as Pastra and Aloimonos (2012) do) then we can take advantage of the techniques already developed by the linguistics community for our task.

One such grammatical inference technique we will make use of is the algorithm **Sequitur** [Nevill-Manning and Witten, 1997]. The algorithm infers a context-free grammar from a string in a greedy way. It reads all symbols in the string and then creates new characters to represent and replace any repeating sub-sequences while adhering to two constraints:

1. The final encoded string must only have unique bigrams (i.e. there must be no pair of symbols that appear more than once in the final string)
2. Only sub-sequences that appear twice or more can get replaced by a new symbol

The left-hand side of Figure 1 provides an example of how Sequitur would encode the string "aaaababbaababaa" to "cebbec". The first repeated bigram it finds is "aa" which appears 4 times and so a new symbol "c" is created to represent

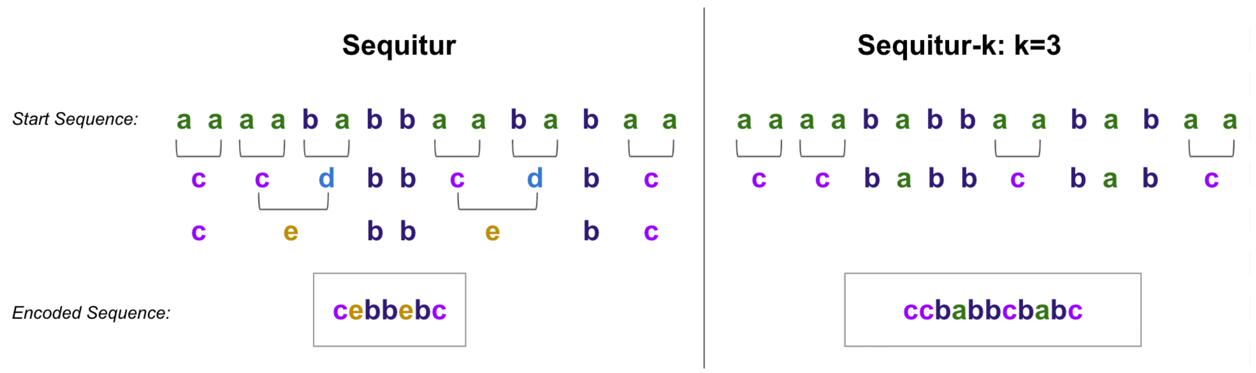

Figure 1: Example of how Sequitur and Sequitur-k with k=3 would code a string differently

"aa". Then it finds that "ba" appears twice and so creates "d" to represent "ba". Then in the next iteration it finds that "cd" repeats twice and so it creates "e" to represent "cd". This leaves the final encoded string of "cebbebc".

To make Sequitur less susceptible to noise, Stout et al. (2018) introduce a more general version called **k-Sequitur**. k-Sequitur works exactly as Sequitur except a sub-sequence is required to occur at least k-times before generating a rule rather than at least 2 times. Higher values for k will be associated with less spurious rules and so will reduce the influence of noise on the final encoded string. The right hand side of Figure 1 shows how Sequitur with k=3 would code a string differently and more conservatively than ordinary Sequitur. [1]

Siyari, Dilkina, and Dovrolis (2016) offer another grammar inference algorithm called **G-Lexis** that takes a slightly different approach. Firstly, they model the process of decomposing a string into substrings using a Directed Acyclic Graph where each node represents a string or substring and each edge represents a concatenation operation. The graph starts off as having a node for each of the characters, one node for the whole string, and then enough edges to combine the characters to form the whole string. Then to encode the string, G-Lexis adds new nodes to represent substrings and adds and deletes edges appropriately so that the nodes still combine to form the whole string.

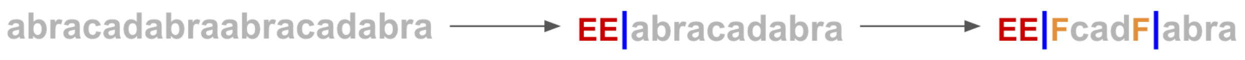

Figure 2: Example of the process by which IGGI encodes the string "abracadabraabracadabra". First we replace "abracadabra" with "E" to create the encoded string "EE". Then we add a special "|" symbol to the right which indicates that what follows is the definition of the new character "E" we created. Then we add "abracadabra" to the right so show that this is what "E" represents. Next we repeat the process with the new character "F" representing "abra"

Instead of deciding which substring to create a new node with on the basis of how many times it appears (like in Sequitur), G-Lexis replaces the substring that leads to the biggest reduction in a *cost function*. The number of times a substring appears can be an important factor in the cost function but the cost function can also take into account other things such as the change in the number of edges adding this node would induce. Rather than there being only one cost function for all situations, the authors see the choice of cost function as application-specific.

In a similar vein, Schoenhense and Faisal (2017) offer an alternative algorithm called **Information-Greedy Grammar Inference (IGGI)** that they show is more robust to noise than Sequitur and G-Lexis. Similarly to G-Lexis, IGGI works by replacing repeated substrings with new symbols in order to iteratively minimise a cost function. IGGI's cost function is the same in all circumstances however and is the "information content" of the encoded string. This cost is given by the length of the encoded string multiplied by the entropy:

$$I = -n \sum_i p_i \log_2 p_i \tag{3}$$

[1]We provide a Python implementation of Sequitur-k at https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/utilities/grammar_algorithms/k_Sequitur.py

where $n$ is the length of the string and $p_i$ is the proportion of symbols in the string that are symbol $i$.

Unlike G-Lexis, IGGI does not use a graph structure but instead when replacing a repeated substring it appends extra elements to the end of the string to indicate the meaning of the newly created characters. Figure 2 gives an example of this process.

Note that introducing a new character to represent a substring means we also have to append elements to the end of the string. This means replacing a repeating substring will not always reduce the cost function and this is why IGGI is generally more conservative in its encoding than an algorithm like Sequitur. [2]

This ends our review of preliminary concepts, we now use them to help review the recent hierarchical RL literature.

[2]We provide a Python implementation of IGGI at https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/utilities/grammar_algorithms/IGGI.py

## 3 Hierarchical Reinforcement Learning Literature Review

Hierarchical RL is a type of RL that attempts to force an agent's policy to be hierarchical rather than "flat". A policy being hierarchical means the agent's decisions have influence beyond the current timestep (compared to a flat policy in which all decisions only directly influence the current timestep). Making the policy hierarchical therefore effectively reduces the time dimensionality of the problem and so can help mitigate the curse of dimensionality if done correctly. Identifying the right hierarchical policy structure is however "not a trivial task" [Osa, Tangkaratt, and Sugiyama, 2019] and so far progress in hierarchical RL has been slow and incomplete - Vezhnevets et al. (2016) argued that: "no truly scalable and successful [hierarchical] architectures exist" and even today state-of-the-art RL agents are rarely hierarchical.

There have been many attempts in the last few years to develop efficient hierarchical RL algorithms. We do not attempt to review all hierarchical algorithms as there are too many, instead we focus on the most successful attempts as well as those that are most informative of the approach we shall take.

In this review we will broadly characterise hierarchical RL algorithms by the framework they use: options, feudal or macro-actions. Habit Reinforcement Learning falls under the macro-actions framework. We note that the lines can sometimes be blurry between the frameworks but believe the characterisation we employ below helps promote clarity. Additionally we note that there are two other significant hierarchical RL frameworks (MAXQ and Hierarchical Abstract Machines) however they do not receive much attention in modern literature and so are not covered here.

### 3.1 Options

Following Sutton, Precup, and Singh (1999), formally an option $w$ is defined as a tuple $w =< I_w, \pi_w, \beta_w >$ where i) Initiation Set $I_w \subseteq S$ specifies which states an option can be initiated from, ii) Intra-Option Policy $\pi_w : S \times A \rightarrow [0, 1]$ specifies the policy to follow whilst using this option, and iii) Termination Condition $\beta : S \rightarrow [0, 1]$ gives the probability of an option terminating.

A typical options-based agent then has two sets of policies: a *policy-over-options* and a set of *intra-option policies*. The policy-over-options takes the state as input and outputs a choice of option. The picked option then takes in the state and chooses the agent's next primitive action. The chosen option continues interacting with the environment until its termination condition is satisfied in which case control returns to the policy-over-options to pick the next option.

Bacon, Harb, and Precup (2016) provided the first scalable and end-to-end approach for learning options called **Option-Critic**. It followed the setup described above where they parameterised the policy-over-options and intra-option policies using neural networks. After deriving an intra-option policy gradient theorem and a termination gradient theorem they use an actor-critic algorithm to train both the policy-over-options and the intra-option policies.

In terms of results they show that Option-Critic is able to make progress on the four-rooms domain [Sutton, Precup, and Singh, 1999] and the pinball domain [Konidaris et al., 2011]. They also show that it performs better than a DQN agent on 3 out of 4 Atari games tested.

Florensa, Duan, and Abbeel (2017) provide another hierarchical RL algorithm arguably in the options mould called **Stochastic Neural Networks for Hierarchical Reinforcement Learning (SNN4HRL)**. SNN4HRL has two types of policy: a higher-level policy and a lower-level policy. Every T > 1 timesteps the higher-level agent takes the state as input and outputs a latent code $z$. The latent code $z$ then gets fed into a lower-level policy which uses it as input along with the state to produce primitive actions for each of the next T timesteps. After T timesteps control returns back to the higher-level agent who chooses a new latent code $z$. We can see each choice of latent code $z$ as inducing a different option.

SNN4HRL's training procedure is also distinct compared to Option-Critic. It has two stages. In the first stage, latent codes $z$ are chosen uniformly at random and fed into the lower-level policy. The lower-level policy is then trained using TRPO to maximise extrinsic reward plus a weighted information theoretic regulariser. The information theoretic regulariser encourages the policy to reach states that are different to those previously reached by the policy when different latent codes were provided. It encourages this by providing an intrinsic reward [Sing, Barto, and Chentanez, 2005] equal to the log of the proportion of times the current state was visited under this latent code compared to under other latent codes. In the second phase of training the lower-level policy is frozen and the higher-level policy is trained using the Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] algorithm to pick a latent code every T steps to maximise rewards from the environment.

They show that SNN4HRL is able to make progress on two hierarchical tasks described in Duan et al. (2016) called Locomotion + Maze and Locomotion + Food Collection. In particular they show that the lower-level agent does provide a varied set of behaviours depending on the latent code it is provided.

Pashevich et al. (2018) provide a similar algorithm to SNN4HRL called **Modulated Policy Hierarchies (MPH)**. The main difference being that rather than the higher level providing the lower-level with a latent code, they provide a bit vector which can have more than one entry equal to one. This has the effect of allowing a combination of low level skills to act at a time rather than only one skill at a time.

Abramova et al. (2019) provide a different type of hierarchical RL algorithm called **Reinforcement Learning Optimal Control (RLOC)** that combines reinforcement learning with optimal linear control. RLOC trains a pre-defined number of linear quadratic regulators [Kalman, 1960] after randomly placing their linearisation centres around the state space. A higher-level policy then acts in the world by taking the state as input and picking which linear quadratic regulator to hand over control to and to let interact in the environment. Finally, a monte-carlo RL method is used to train the higher-level policy. In terms of results they show RLOC outperforms deterministic state-of-the-art control system iLQR on a cart-pole task.

## 3.2 Feudal

Next we review the feudal hierarchical RL framework which takes a different approach to options. Instead of training multiple lower-level policies to perform different tasks, it trains lower-level policies to achieve the *goals* given to it by the higher-level policy.

Hierarchical RL agents reviewed below all follow the basic feudal setup: every T > 1 timesteps, a higher-level agent takes the state as input and produces a goal state as its output. The lower-level agent then takes the state and the goal state as input and produces the agent's primitive actions every timestep. The higher-level agent receives extrinsic rewards from the environment whilst the lower-level agent receives some mixture of extrinsic reward and intrinsic reward where the intrinsic reward is determined by how close the agent got to the goal state it was set.

Kulkarni et al. (2016) propose one of the first feudal-style hierarchical RL algorithms, called **Hierarchical DQN (h-DQN)**. h-DQN trains its higher-level policy using deep Q-learning on the observed extrinsic rewards after treating the goal states it outputted as its "actions". The lower-level agent is also trained using deep Q-learning but instead of extrinsic rewards it receives intrinsic rewards equal to one at the timestep any goal was achieved and zero otherwise. The lower-level agent is therefore trained only to follow the orders it is given and not to directly solve the game.

The authors show h-DQN learns faster than ordinary DQN in a toy game with discrete states and actions, and in Montezuma's Revenge when domain knowledge is used to specify goal candidates. However they provide no evidence that h-DQN is an improvement over ordinary DQN in non-toy games without the use of domain knowledge.

Vezhnevets et al. (2017) provide another feudal-style hierarchical RL algorithm called **FeUdal Network (FuN)**. Unlike h-DQN, FuN is trained using A3C rather than deep Q-learning. FuN also requires the higher-level agent to provide the lower-level agent with a goal state *embedding* rather than a raw goal state and also the lower-level agent in FuN is not allowed to treat the goal as any other input and must incorporate it in a multiplicative manner.

FuN achieves much more impressive results than h-DQN as they are able to show that FuN performs better than a strong A3C baseline for 5 out of 10 Atari games.

Next, Nachum et al. (2018) provide another feudal-style hierarchical RL agent called **Hierarchical Reinforcement Learning With Off-Policy Correction (HIRO)**. HIRO's main innovation over other hierarchical RL algorithms like FuN is that they train their higher-level agent in an off-policy manner. The authors note that, previously, higher-level agents had generally required on-policy training because the changing behaviour of the lower-level policy creates a non-stationarity problem for the higher-level policy. On-policy training is however less sample efficient than off-policy training so HIRO introduced the idea of an *off-policy correction* that allows training the higher-level agent in an off-policy way. The off-policy correction involved drawing experience from the replay buffer to learn from and then relabelling the actions conducted by the higher-level policy (i.e. the goals it set) to be those actions that make the observed action from the lower-level policy the most likely to have occured given the most up to date lower-level policy. In practice they tried relabelling each goal with 10 different goals and picked the one that produced the highest probability of the observed actions.

They show that HIRO performs better than 3 hierarchical RL algorithms (FuN, SNN4HRL and VIME [Houthooft et al., 2017]) and a non-hierarchical baseline of DDPG on 4 ant locomotion and object manipulation environments. However, the environments they used were either non-standard or were uniquely adapted versions of standard environments and so it is difficult to tell how generally applicable their results were.

Levy et al. (2019) introduce another feudal-style hierarchical RL agent called **Hierarchical Actor Critic (HAC)**. As in HIRO, the HAC higher-level agent stores its experiences in a replay buffer and is trained in an off-policy manner. To allow for more stable off-policy training, instead of using an off-policy correction as in HIRO however, HAC uses two forms of hindsight [Andrychowicz et al., 2017]. After drawing a random sample of experience from the replay buffer, they firstly use hindsight action transitions by changing the goal state outputted by the higher-level to be the state the lower-level agent actually ends up in at the end of its sequence of actions. They then also use ordinary hindsight experience replay by changing the overall goal of an episode to be equal to the final state achieved at the end of the episode.

Other innovations besides their use of hindsight include that they periodically turn off all exploration and punish a lower-level policy if it is not able to achieve its subgoal. They also show that HAC can jointly learn 3-level hierarchical policies instead of only 2-levels as we saw in h-DQN, FuN, and HIRO.

In terms of results they demonstrate that HAC outperforms non-hierarchical versions of the algorithm in a series of grid-world and simulated robotic environments. For the simulated robotic environments they also show that it performs better than HIRO.

Finally, Beyret, Shafti, and Faisal (2019) provide another feudal-style agent called **Dot-to-Dot** that uses hindsight in a similar way to HAC to train the higher-level agent in an off-policy manner. The main difference with Dot-to-Dot however was that they force the goals the higher-level agent sets to be nearby to where the agent currently is. This can be seen as a form of curriculum learning [Bengio et al., 2009] and means that is never too hard for the lower-level agent to achieve the goal it is set. They show that Dot-to-Dot with an underlying DDPG algorithm is able to perform better on the Fetch Push and Fetch Pick and Place tasks than DDPG on its own or DDPG with hindsight experience replay.

## 3.3 Macro-Actions

Macro-actions are the third and arguably simplest hierarchical RL framework. Following Vezhnevets et al. (2016), a macro-action is a sequence of actions where the action sequence (or distribution over them) is decided at the time the macro-action is initiated. For example, if the possible primitive actions are $a$ or $b$ then a 2-step macro-action might be $ab$ and a 5-step macro-action might be $ababa$ etc. The agent picking macro-action $ab$ would then mean it commits to playing $b$ at timestep 2 even before it has seen the next state and reward that occured as a result of the first move. Using macro-actions of length 2 or more induces a hierarchical policy as it means some decisions will have influence beyond the current timestep.

The **Strategic Attentive Writer (STRAW)** [Vezhnevets et al., 2016] agent is one of the few attempts to create a hierarchical RL agent that uses macro actions. STRAW is a deep recurrent neural network with two modules. The first module takes in an observation of the environment and produces an "action-plan" $A \in R^{|A| \times T}$ where $A$ is the discrete action set and T is a pre-defined number of timesteps greater than 1. Each column in the action-plan gives the probability of the agent choosing each action at that timestep. As $T > 1$, creating an action-plan involves making decisions that span multiple timesteps. The second module produces a commitment-plan $c \in R^{1 \times T}$ which is used to determine the probabilities of terminating the macro-action and re-calculating the action-plan at a particular timestep.

As the underlying RL algorithm they use the Asynchronous Actor-Critic (A3C) algorithm [Mnih et al., 2016]. In terms of results they show that STRAW performs better than an ordinary A3C algorithm on 6 out of 8 Atari games. They also show that injecting noise into the communication channel between the feature extractor and the planning modules to encourage exploration leads to better results in most cases.

Sharma, Lakshminarayanan, and Ravindran (2017) provide another interesting hierarchical RL agent that uses macro-actions called **Fine Grained Action Repetition (FiGAR)**. Simply, FiGAR agents maintain two policies: one policy which chooses a primitive action as normal and another policy that chooses how many times the chosen primitive action will be repeated. The authors note that the two policies must be independent so as to avoid an exponential increase in possibilities. They show how FiGAR can be incorporated with a number of underlying RL agents such as A3C, TRPO, and Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al., 2016]. FiGAR's results were more impressive as they show that the FiGAR extension improves results over an A3C baseline in the 25 out of 33 Atari games tested.

Lastly, Lange and Faisal (2018) provide one of the only attempts so far to develop hierarchical RL algorithms using action grammars. The algorithm they develop that is most similar to ours is called **Macro-Q-Learning with Online Updated CFG Production Rules**. It is only relevant for simple discrete state settings and works by alternating between two steps:

1. *Action learning* which involves the agent taking actions and using rewards from the environment to learn the tabular Q-values of different actions
2. *Grammar learning* which involves using past action sequences and algorithms such as Sequitur-k and G-Lexis to infer an action grammar. The agent's action space then gets augmented with the macro-actions implied by the action grammar

To append new macro-actions from the action grammar to the action set the authors simply add new columns to the Q-table. This means that they do not have to re-learn the Q-values of pre-existing actions every time they update the action set. This therefore saves them a lot of training time but is something that can only work in discrete state setting. After adding a new macro-action to the action set the agent plays as normal and treats the macro-action as if it were any other action.

Table 1: Summary of the differences between Habit Reinforcement Learning and other algorithms reviewed in this dissertation. Dark purple indicates a difference with Habit Reinforcement Learning and green indicates a rough similarity.

| | **Habit RL** | **Macro Deep Q Learning with CFG Production Rules** *(Lange and Faisal 2019)* | **Macro Q Learning with CFG Production Rules** *(Lange and Faisal 2018)* | **STRAW** *(Vezhnevets et al. 2016)* | **FiGAR** *(S. Sharma et al. 2017)* | **h-DQN** *(Kulkarni et al. 2016)* | **FuN** *(Vezhnevets et al. 2017)* | **HIRO** *(Nachum et al. 2018)* | **HAC** *(Levy et al. 2019)* | **Dot to Dot** *(Beyret et al. 2019)* | **Option Critic** *(Bacon et al. 2016* | **SNN 4HRL** *(Florensa et al. 2017)* | **MPH** *(Pashevich et al. 2018)* | **RLOC** *(Abramova et al. 2019)* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ***Type of HRL Algorithm*** | Macro Actions | Macro Actions | Macro Actions | Macro Actions | Macro Actions | Feudal | Feudal | Feudal | Feudal | Feudal | Options | Options | Options | Options |
| ***Action Space*** | Discrete | Discrete | Discrete | Discrete | Continuous + Discrete | Discrete | Discrete | Continuous | Continuous | Continuous | Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete |
| ***State Space*** | Continuous + Discrete | Continuous + Discrete | Discrete | Continuous + Discrete | Continuous + Discrete | Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete | Continuous + Discrete |
| ***On or Off-policy*** | Off-policy | Off-policy | Off-policy | Off-policy | On or Off policy | Off-policy | On-policy | Off policy | Off policy | Off policy | On-policy | On-policy | On-policy | On-policy |
| ***Uses Function Approximation*** | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No |
| ***Underlying RL Algorithm*** | DDQN and SAC Discrete | Deep Q-Learning | Tabular Q-Learning | A3C | A3C, TRPO or DDPG | DQN | A3C | DDPG | DDPG | DDPG | Policy Gradients and DQN | TRPO | PPO | Monte Carlo Control |
| ***Infers Action Grammar / Habits*** | Yes | Yes | Yes | No | No | No | No | No | No | No | No | No | No | No |
| ***Demonstrates Improvement in...*** | 27/28 Atari game settings | 0/1 simple games | 2/4 simple games | 5/8 Atari game settings | 25/33 Atari game settings | 1/1 toy game | 5/10 Atari game settings | 4/4 custom games | 6/6 Mujoco games | 2/2 Mujoco games | 3/4 Atari game settings | 2/2 Mujoco games | 2/2 Mujoco games | 1/1 Cartpole task |

They test their algorithm on the very simple discrete state settings of Towers of Hanoi, The Four Rooms Problem [Sutton, Precup, and Singh, 1999], Taxi and Frozen Lake [Brockman et al., 2016]. For Towers of Hanoi and Frozen Lake, their algorithm appears to improve on the alternative of ordinary Q-Learning, however for the other games the results are unclear.

In concurrent work to this dissertation, the same authors released a second paper [Lange and Faisal, 2019] that attempts to make their algorithm compatible with Deep Q-Learning (instead of only tabular Q-Learning) and therefore applicable to continuous state settings. To append a new macro-action to the action set, instead of adding a new column to a Q-table, they now add a new node to the final layer of the Q network. Another change they make to improve information efficiency is that when storing the experience of playing a macro-action they also store the experiences of the individual primitive actions taken during execution of the macro-action. Finally, they also limit the number of macro-actions that can be added to the action set to 2 at a time.

They test the algorithm on a very simple discrete state Gridworld environment but are unable to show that the algorithm improves over ordinary Deep Q-Learning. They also do not test the algorithm in any continuous state settings.

Overall therefore, both attempts at using the action grammars framework of Lange and Faisal (2018) have been relatively unsuccessful in terms of results and it remains to be seen whether the framework can be used to consistently improve the performance of RL algorithms. This dissertation takes inspiration from the framework but builds on it in many ways to provide algorithms that we show *do* significantly and consistently improve the performance of RL agents while also working in continuous state settings.

We provide a broad summary of the reviewed algorithms and how they compare to Habit Reinforcement Learning in Table 1. The bottom row demonstrates the superior results Habit Reinforcement Learning is able to achieve that we go through later in Section 5.3. This ends our literature review and now we derive a discrete action version of the Soft Actor-Critic algorithm before introducing the Habit Reinforcement Learning framework.

# 4 Discrete Soft Actor-Critic

As mentioned earlier, the **Soft Actor-Critic** algorithm is a state-of-the-art model-free algorithm for environments with *continuous* action spaces. The original algorithm is not applicable to environments with *discrete* actions though. We wish to use it later as a benchmark to apply Habit Reinforcement Learning to however, so we now derive a version of Soft Actor-Critic that is applicable to discrete action settings and show that it is competitive with state-of-the-art Rainbow.

First, we explain the derivation of Soft Actor-Critic for continuous action spaces in Haarnoja et al. (2018) and Haarnoja et al. (2019) below. Then we derive and explain the changes required to create a discrete action version of the algorithm.

## 4.1 SAC with Continuous Actions

As mentioned earlier, Soft Actor-Critic attempts to find a policy that maximises the maximum entropy objective:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T} E_{(s_t,a_t)\sim\tau_\pi}[\gamma^t(r(s_t,a_t) + \alpha\mathcal{H}(\pi(.|s_t))] \tag{4}$$

where $\pi$ is a policy, $\pi^*$ is the optimal policy, $T$ is the number of timesteps, $r : S \times A \rightarrow \mathbb{R}$ is the reward function, $\gamma \in [0,1]$ is the discount rate, $s_t \in S$ is the state at timestep t, $a_t \in A$ is the action at timestep t, $\tau_\pi$ is the distribution of trajectories induced by policy $\pi$, $\alpha$ determines the relative importance of the entropy term versus the reward and is called the temperature parameter, and $\mathcal{H}(\pi(.|s_t)$ is the entropy of the policy $\pi$ at state $s_t$ and is calculated as $\mathcal{H}(\pi(.|s_t)) = -\log\pi(.|s_t)$.

To maximise the objective the authors use soft policy iteration which is a method of alternating between policy evaluation and policy improvement within the maximum entropy framework.

The policy evaluation step involves computing the value of policy $\pi$. To do this they first define the soft state value function as:

$$V(s_t) := E_{a_t\sim\pi}[Q(s_t,a_t) - \alpha\log(\pi(a_t|s_t))] \tag{5}$$

They then prove that in a tabular setting (i.e. when the state space is discrete) we can obtain the soft q-function by starting from a randomly initialised function $Q : S \times A \rightarrow \mathbb{R}$ and repeatedly applying the modified Bellman backup operator $T^\pi$ given by:

$$T^\pi Q(s_t,a_t) := r(s_t,a_t) + \gamma E_{s_{t+1}\sim p(s_t,a_t)}[V(s_{t+1})] \tag{6}$$

where $p : S \times A \rightarrow S$ gives the distribution over the next state given the current state and action.

The policy improvement step then involves updating the policy in a direction that maximises the rewards it will achieve. To do this they use the soft Q-function calculated in the policy evaluation step to guide changes to the policy. Specifically, they update the policy towards the exponential of the new soft Q-function. Because they also want the policy to be tractable however, they restrict the possible policies to a parameterised family of distributions (e.g. Gaussian). To account for this, after updating the policy towards the exponential of the soft Q-function they then project it back into the space of acceptable policies using the information projection defined in terms of Kullback-Leibler divergence. So overall the policy improvement step is given by:

$$\pi_{\text{new}} = \underset{\pi\in\Pi}{\operatorname{argmin}} D_{\text{KL}}\left(\pi(.|s_t) \,\middle\|\, \frac{\exp(\frac{1}{\alpha}Q^{\pi_{old}}(s_t,.))}{Z^{\pi_{\text{old}}}(s_t)}\right) \tag{7}$$

They note that partition function $Z^{\pi_{\text{old}}}(s_t)$ is intractable but does not contribute to the gradient with respect to the new policy and so it can be ignored. They then go on to prove that in the tabular setting, alternating between policy evaluation and policy improvement as above will converge to the optimal policy.

For the much more general continuous state setting (rather than tabular setting) though they make some changes to allow them to approximate the above soft policy iteration. Firstly, they parameterise the soft q-function $Q_\theta(s_t,a_t)$ using a neural network with parameters $\theta$. Then they also parameterise the policy $\pi_\phi(a_t|s_t)$ using a neural network with parameters $\phi$ that outputs a mean and covariance that is then used to define a Gaussian policy.

The soft Q-function is then trained to minimise the soft Bellman residual :

$$J_Q(\theta) = E_{(s_t,a_t)\sim D}[\frac{1}{2}(Q_\theta(s_t,a_t) - (r(s_t,a_t) + \gamma E_{s_{t+1}\sim p(s_t,a_t)}[V_{\bar{\theta}}(s_{t+1})]))^2] \tag{8}$$

where D is a replay buffer of past experiences and $V_{\bar{\theta}}(s_{t+1})$ is estimated using a target network for $Q$ and a monte-carlo estimate of (5) after sampling experiences from the replay buffer .

The policy parameters are learned by minimizing the expected KL-divergence (7) after multiplying by the temperature parameter $\alpha$ and ignoring the partition function $Z^{\pi_{\text{old}}}(s_t)$ as it does not impact the gradient [3]:

$$J_\pi(\phi) = E_{s_t\sim D}[E_{a_t\sim\pi_\phi}[\alpha\log(\pi_\phi(a_t|s_t)) - Q_\theta(s_t,a_t)]] \tag{9}$$

This involves taking an expectation over the policy's output distribution which means errors cannot be backpropagated in the normal way. To deal with this they use the reparameterisation trick [Kingma and Welling, 2013] - instead of using the output of the policy network to form a stochastic action distribution directly, they combine its output with an input noise vector sampled from a spherical Gaussian. For example, in the one-dimensional case our network outputs a mean $m$ and standard deviation $s$. We could randomly sample our action directly $a \sim N(m,s)$ but then we could not backpropagate the errors through this operation. So instead we do $a = m + s\epsilon$ where $\epsilon \sim N(0,1)$ which allows us to backpropagate as normal.

To signify that they are reparameterising the policy in this way they write:

$$a_t = f_\phi(\epsilon_t; s_t) \tag{10}$$

where $\epsilon_t \sim N(0,I)$. The new policy objective then becomes:

$$J_\pi(\phi) = E_{s_t\sim D,\epsilon_t\sim N}[\alpha\log(\pi_\phi(f_\phi(\epsilon_t;s_t)|s_t)) - Q_\theta(s_t, f_\phi(\epsilon_t;s_t))] \tag{11}$$

where $\pi_\phi$ is now defined implicitly in terms of $f_\phi$.

Haarnoja et al. (2019) provide a long derivation for the temperature objective value, however because the details are not strictly relevant for our derivation of the discrete action version of SAC we do not repeat it here. The objective they get to for the temperature parameter is however given by:

$$J(\alpha) = E_{a_t\sim\pi_t}[-\alpha(\log\pi_t(a_t|s_t) + \bar{H})] \tag{12}$$

where $\bar{H}$ is a constant vector equal to the hyperparameter representing the target entropy. They are unable to minimise this expression directly because of the expectation operator and so instead they minimise a monte-carlo estimate of it after sampling experiences from the replay buffer.

Lastly, in practice the authors maintain two separately trained soft Q-networks and then use the minimum of their two outputs to be the soft Q-network output in the above objectives. They do this because Fujimoto, Hoof, and Meger (2018) showed that it helps combat state-value overestimation in a similar way to Double Deep Q-Learning.

## 4.2 SAC with Discrete Actions

We now derive a discrete action version of the above SAC algorithm.

The first thing to note is that all the critical steps involved in deriving the objectives above hold whether the actions are continuous or discrete. All that changes is that $\pi_\phi(a_t|s_t)$ now outputs a probability instead of a density. Therefore the three objective functions $J_Q(\theta)$ (8), $J_\pi(\phi)$ (9) and $J(\alpha)$ (12) still hold.

We must however make 5 important changes to the process of optimising these objective functions.

i) It is now more efficient to have the soft Q-function output the Q-value of each possible action rather than simply the action provided as an input. i.e. our Q function moves from $Q : S \times A \rightarrow \mathbb{R}$ to $Q : S \rightarrow \mathbb{R}^{|A|}$. This was not possible before when there were infinitely many possible actions we could take.

[3]Note that (9) is not exactly same as in Haarnoja et al., 2019 because the authors made a typographical error and wrote $Q(s_t, s_t)$ when they should have written $Q(s_t, a_t)$

ii) There is now no need for our policy to output the mean and covariance of our action distribution, instead it can directly output our action distribution. The policy therefore changes from $\pi : S \rightarrow \mathbb{R}^{2|A|}$ to $\pi : S \rightarrow [0,1]^{|A|}$ where now we are applying a softmax function in the final layer of the policy to ensure it outputs a valid probability distribution.

iii) Before, in order to minimise the soft Q-function cost $J_Q(\theta)$ (8) we had to plug in our sampled actions from the replay buffer to form a monte-carlo estimate of the soft state-value function (5). This was because estimating the soft state-value function in (5) involved taking an expectation over the action distribution. However, now, because our action set is discrete we can fully recover the action distribution and so there is no need to form a monte-carlo estimate and instead we can calculate the expectation directly. This change should reduce the variance involved in our estimate of the objective $J_Q(\theta)$ (8).

This means that we change our soft state-value calculation equation from (5) to:

$$V(s_t) := \pi(s_t)^T[Q(s_t) - \alpha \log(\pi(s_t))] \tag{13}$$

iv) Similarly, we can make the same change to our calculation of the temperature loss to also reduce the variance of that estimate. So the temperature objective changes from (12) to :

$$J(\alpha) = \pi_t(s_t)^T[-\alpha(\log(\pi_t(s_t)) + \bar{H})] \tag{14}$$

v) Before, to minimise $J_\pi(\phi)$ (9) we had to use the reparameterisation trick to allow gradients to pass through the expectations operator. However, now our policy outputs the exact action distribution we are able to calculcate the expectation directly. Therefore there is no need for the reparameterisation trick and the new objective for the policy changes from (11) to:

$$J_\pi(\phi) = E_{s_t \sim D}[\pi_t(s_t)^T[\alpha \log(\pi_\phi(s_t)) - Q_\theta(s_t)]] \tag{15}$$

Combining all these changes, our algorithm for SAC with discrete actions is given by Algorithm 1. We also provide a Python implementation for this algorithm in the project's GitHub repository. [4]

**Algorithm 1** Soft Actor-Critic with Discrete Actions

```
Initialise $Q_{\theta_1} : S \rightarrow \mathbb{R}^{|A|}$,$Q_{\theta_2} : S \rightarrow \mathbb{R}^{|A|}$,$\pi_\phi : S \rightarrow [0,1]^{|A|}$        ▷ Initialise local networks
Initialise $\bar{Q}_{\theta_1} : S \rightarrow \mathbb{R}^{|A|}$, $\bar{Q}_{\theta_2} : S \rightarrow \mathbb{R}^{|A|}$        ▷ Initialise target networks
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$        ▷ Equalise target and local network weights
$\mathcal{D} \leftarrow \emptyset$        ▷ Initialize an empty replay buffer
for each iteration do
    for each environment step do
        $a_t \sim \pi_\phi(a_t|s_t)$        ▷ Sample action from the policy
        $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$        ▷ Sample transition from the environment
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$        ▷ Store the transition in the replay buffer
    for each gradient step do
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J((\theta_i))$ for $i \in \{1,2\}$        ▷ Update the Q-function parameters
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$        ▷ Update policy weights
        $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$        ▷ Update temperature
        $\bar{Q}_i \leftarrow \tau Q_i + (1-\tau)\bar{Q}_i$ for $i \in \{1,2\}$        ▷ Update target network weights
Output $\theta_1, \theta_2, \phi$        ▷ Optimized parameters
```

## 4.3 SAC with Discrete Actions vs. Rainbow

To test the effectiveness of SAC with discrete actions we run it for 100,000 steps on 20 Atari games for 5 random seeds each. The games chosen vary significantly and we believe they broadly represent the Atari suite. The games were also picked before any results were calculated so as to avoid any bias. We chose to run the algorithm for 100,000 steps because this dissertation is most interested in sample efficiency and Kaiser et al. (2019) used the same number of steps to investigate sample efficiency after demonstrating that Rainbow can make significant progress on Atari games during that time.

[4] https://github.com/p-christ/Habit-Reinforcement-Learning

We compare the results of SAC with Rainbow which, as mentioned earlier, is a state-of-the-art model-free algorithm for sample efficiency.

For SAC with discrete actions we did no hyperparameter tuning and instead used a mixture of the hyperparameters found in Haarnoja et al. (2019) and Kaiser et al. (2019). The hyperparameters can be found in Appendix C. The Rainbow results we compare to come from Kaiser et al. (2019) and as they explain were the result of a significant amount of hyperparameter tuning [5]. Therefore we are comparing the tuned Rainbow algorithm to our untuned SAC algorithm and so it is highly likely the relative performance of SAC could be improved if we tuned its hyperparameters.

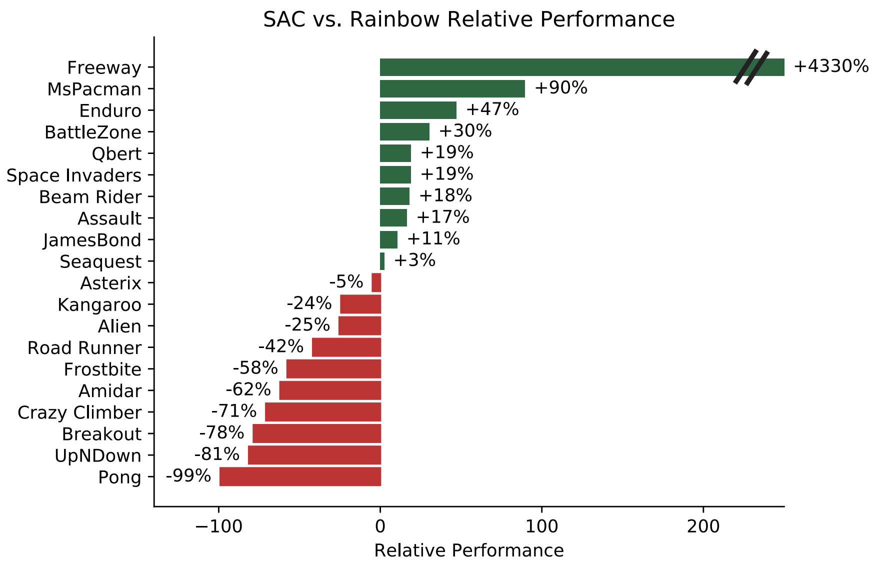

Figure 3: Comparing SAC with discrete actions to Rainbow for 20 Atari games. The graph shows the average relative performance of SAC over Rainbow over 5 random seeds where evaluation scores are calculated at the end of 100,000 steps of training

We find that SAC with discrete actions achieves a better score than Rainbow in 10 out of 20 games with a median performance of -1%, maximum performance of +4330% and minimum of -99% - Figure 3 summarises the results and Appendix A provides them in a table. **Overall, we therefore consider our SAC with discrete actions algorithm as roughly competitive with the model-free state-of-the-art on the Atari suite in terms of sample efficiency**.

[5]For two games (Enduro and SpaceInvaders) Kaiser et al. (2019) provide no results for Rainbow and so for these two games only we ran the Rainbow algorithm ourselves. We used the Dopamine [Castro et al., 2018] codebase to do this (as Kaiser et al. (2019) also did) along with the same (tuned) hyperaprameters used in Kaiser et al. (2019). We share the code used to do this in the colaboratory notebook: https://colab.research.google.com/drive/11prfRfM5qrMsfUXV6cGY868HtwDphxaF

## 5 Habit Reinforcement Learning

### 5.1 Motivation

Before explaining the algorithm, we now expand on the motivation given in the introduction to explain more specifically why we expect Habit Reinforcement Learning to dramatically improve sample efficiency.

Say we are playing a game that lasts 50 timesteps, has action set $A = \{a, b\}$, and where the state is just given by the timestep we are in, i.e. $S = \{1, .., 50\}$. There are therefore $2^{50}$ ways to play this game and so $2^{50}$ represents the size of the space we must search over for a solution (i.e. our hypothesis space).

As an aside note that we can see the curse of dimensionality in action here by observing that if we were to increase the number of timesteps by 2% (from 50 to 51) it would lead to a much larger increase of 100% in the game's hypothesis space to $2^{51} = 2$ x $2^{50}$. A similar argument would also hold if we increased the action space or state space.

Now consider that when playing this game we could always choose to make steps of 2 instead of steps of 1 by changing our action set to all possible 2-step macro-actions - i.e. we could change our action set from $A = \{a, b\}$ to $A = \{aa, ab, ba, bb\}$. Unfortunately this would not combat the curse of dimensionality because our hypothesis space would remain as it was before as there are still $4^{25} = (2^2)^{25} = 2^{50}$ ways of playing the game.

Say instead we somehow knew that the environment was such that the 3 2-step macro-actions $\{aa, ab, bb\}$ were the only relevant 2-step macro-actions and it was never an optimal choice in the environment to pick the other macro-action $ba$. Then by moving to an action set of 2-step macro-actions we could instead change our action space from $A = \{a, b\}$ to $A = \{aa, ab, bb\}$. This would dramatically reduce the complexity of the game as the hypothesis space would fall from $2^{50}$ to $3^{25}$ which represents a drop of over $99.9\%$. This in turn would make learning the solution to the game dramatically faster. We summarise this in Table 2.

Table 2: Shows how using different actions spaces results in differently sized hypothesis spaces in a setting where the action space is given by $A = \{a, b\}$ and the state is the timestep we are in so $S = \{1, .., 50\}$. If we are able to rule out 1 2-step macro-action then moving to an action set of 2-step macro-actions lets us reduce the hypothesis space by over 99.9%.

| **Action Space** | **Hypothesis Space Size** | **Improvement** |
|---|---|---|
| $\{a, b\}$ | $2^{50}$ | N/A |
| $\{aa, ab, ba, bb\}$ | $4^{25}$ | 0.0% |
| $\{aa, ab, bb\}$ | $3^{25}$ | 99.9% |

In general therefore, if we were able to identify the relevant macro-actions we could adapt an agent's action space to dramatically speed up learning. Habit Reinforcement Learning takes inspiration from how the brain works to identify the most relevant macro-actions. It uses an agent's past behaviour to identify its habits and then adapts its action space to make it easier to repeat the habits similarly to how the brain adapts to make it easier for humans to repeat habits. This injects the agent with more of an inductive bias towards the past which we show leads to significantly improved sample efficiency.

### 5.2 Methodology

#### 5.2.1 Basic Setup

At a high-level, Habit Reinforcement Learning will work by iterating through 2 steps similar to those in Lange and Faisal (2018):

(A) **Gather Experience**: the base RL agent plays episodes and stores its experiences [6]

(B) **Identify Habits**: the actions used in the agent's past experiences are used to identify the habits. The habits are then appended to the agent's action set

[6]Note that the base RL agent could be almost any off-policy RL agent that works with discrete actions e.g. DQN, DDQN, Rainbow, discrete SAC etc.

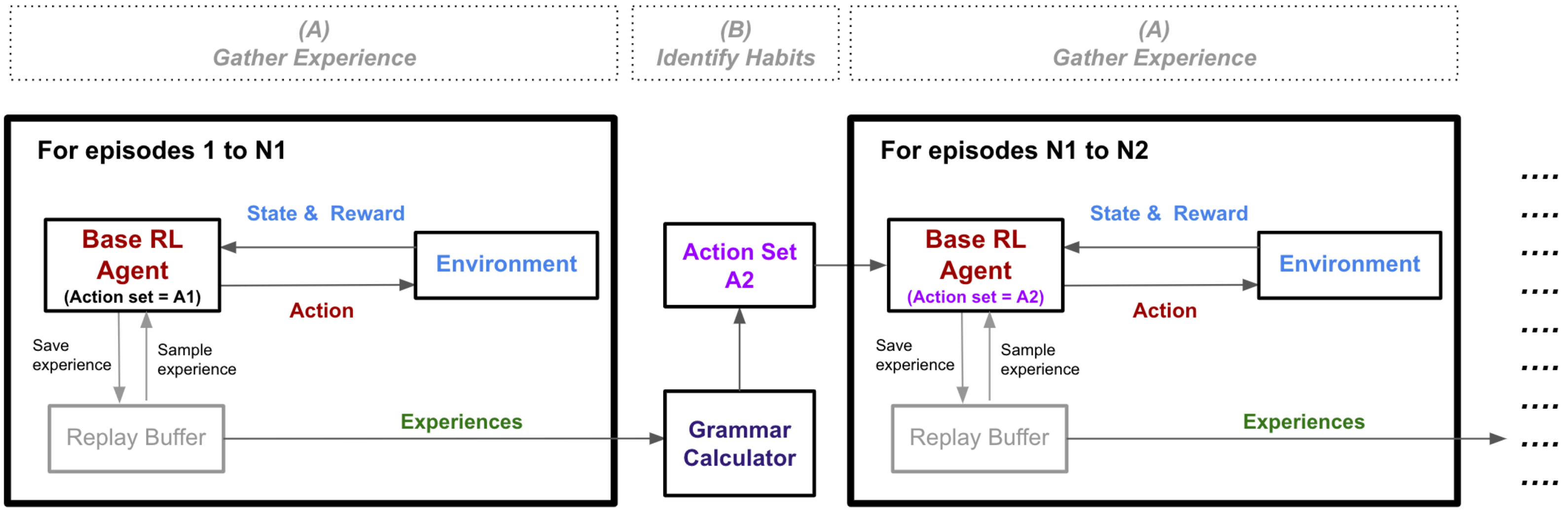

Figure 4: Represents the high-level process of Habit Reinforcement Learning. First the base RL agent plays N1 episodes as normal. Then the experiences are fed into a grammar calculator which identifies the habits. Then we change the agent's action set to a new action set A2 which incorporates the habits and play more episodes. This process can repeat as many times as we like, including only once.

Figure 4 lays out the process in more detail. First, in the gather experience step, the base RL agent plays N1 episodes by choosing an action every timestep as normal from the primitive action set which could be $A1 = \{a, b\}$ for example.

Then, after N1 episodes the experiences seen so far are fed into a grammar calculator (e.g. Sequitur or IGGI) as we enter the habt identification step. The grammar calculator identifies the most notable and common patterns of actions (which we are calling the "habits") in the experiences and uses them to form the new action set A2. For example, the habits identified might be $\{aaa, ababb\}$ and then we would have $A2 = \{a, b, aaa, ababb\}$

Then we update the agent's action set to A2 and return to a gather experience step by having it play more episodes as normal but with the new action set. We can repeat this process as many times as we like including only once.

### 5.2.2 Extensions to Basic Setup

Departing from the work of Lange and Faisal (2019) we extend this basic setup in 7 critical ways to create an algorithm that works consistently and effectively in a wide range of challenging environments.

i) We have the grammar calculator only infer habits from the actions in the **top performing episodes** with exploration turned off rather than all episodes. To collect this data we periodically run episodes of the game with all exploration turned off. Then when it is time to infer the habits (unlike in Lange and Faisal (2019)) we take the actions from the top performing of such episodes and use this to infer the habits. We infer habits from no-exploration episodes because we do not want the random exploration moves to influence the habits. We then only learn habits from the very best performing of those episodes because we want to identify the most helpful habits rather than all habits and it is in those episodes where the agent was most likely to be using its most helpful habits.

ii) We use **IGGI** as our grammar calculator instead of Sequitur which Lange and Faisal (2019) used. We use IGGI because Schoenhense and Faisal (2017) show that it is much less sensitive to noise than Sequitur. We also compare the performance of IGGI and Sequitur in an ablation study later on.

iii) Also, unlike in Lange and Faisal (2019), **we do not restrict number of macro-actions that we can add to the action set at a time** to 2. Instead we allow the agent to add all macro-actions suggested by the grammar calculator (which we find can sometimes be over 50).

iv) To update the agent's action set without destroying what our q-network and/or policy has already learned we use a form of **transfer learning** laid out in Figure 5. First (as in Lange and Faisal (2019)) we simply add new neurons to the final layer of the q-network and/or policy to represent the new macro-actions, leaving the rest of the network as it was. Unlike in Lange and Faisal (2019) however, we then further leverage transfer learning by initialising the weights of a new macro-action's neuron as equal to the weights of the neuron corresponding to the first primitive action of that

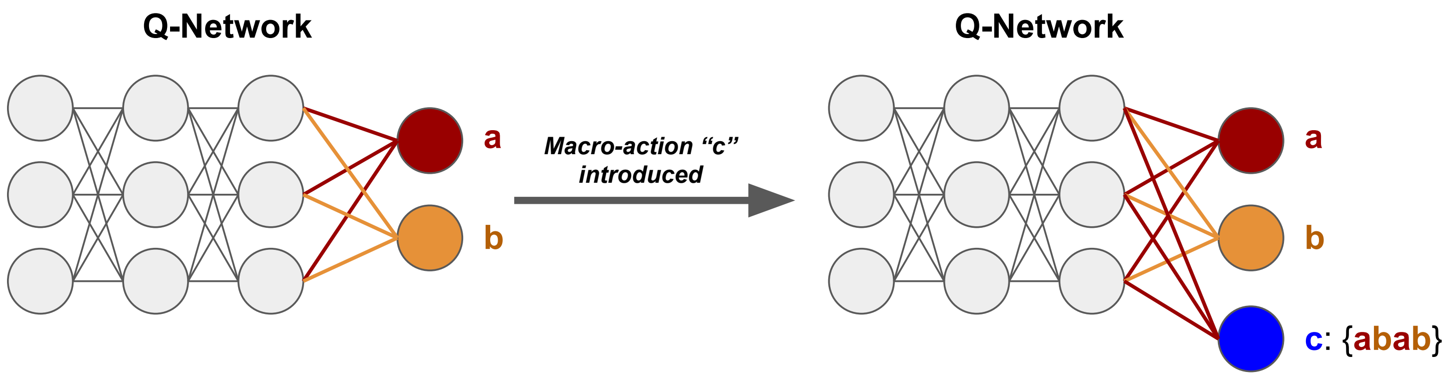

Figure 5: The transfer learning process involved when adding a new macro-action to the action set. The circles represent the nodes and the lines represent the weights in an example Q-network with 2 hidden layers. To add a new macro-action we transfer the weights of all layers except the final layer by simply adding a new node to the final layer for the new macro-action. Unlike in Lange and Faisal (2019) we then also use transfer learning within the final layer itself by initialising the final layer weights for the new node to those weights of the first primitive action in the newly introduced macro-action. In this example the first primitive action of macro-action "c" is primitive action "a" so we initialise the final layer weights of "c" to those of "a".

macro-action. For example, if we are adding a new neuron for the primitive action $abab$ then we would initialise its weights with the weights of its first primitive action: $a$ (in Figure 5 this is represented by the lines going into the "c" node being the same colour as into the "a" node). We transfer over the weights of the first primitive action because a q-value estimates the value of playing a particular move and then following the policy from then onwards, and so this value should be most closely related to the q-value of the macro-action as it too makes the same first move.

v) A problem we then encountered with our method was that, especially with very long macro actions and at the beginning of training, there can sometimes be very few experiences of using macro-actions to learn from. Or, if the macro actions are very long then it can mean the number of experiences using primitive actions suffers instead. In order to deal with this problem and maximise efficiency we develop a new technique we call **Hindsight Action Replay (HAR)**. It is related to Hindsight Experience Replay (HER) explained earlier which creates new experiences by reimagining the goals to match the states achieved. Instead of reimagining the goals though, HAR creates new experiences by reimagining the *actions*. In particular it reimagines them in two ways: [7]

1. If we play a macro-action then we also store the experiences as if we had played the sequence of primitive actions individually
2. If we play a sequence of primitive actions that matches an existing macro-action then we also store the experiences as if we had played the macro-action

Figure 6 lays out an example of the HAR technique. The agent has played the action sequence "acab" where "c" is a macro-action that represents "abab". We then convert all macro-actions to their primitive actions so the sequence becomes "aababab". Then, we store the 7 experiences corresponding to the 7 primitive actions (even though only 3 of them were actually played by the agent in that way). Then we store the 2 experiences corresponding to macro-action "c" even though it was only actually played once by the agent. Note that to store the experience of a macro-action we set the state to be the state at the start of the macro-action execution, the next state and termination information to be the values at the end of execution, and the reward to be the sum of rewards received during execution.

In the end we add 9 experiences to the replay buffer even though only 4 actions were played initially which is an *increase of 125%* in information efficiency. HAR therefore helps us use information much more efficiently when using macro-actions and is one of the novel contributions of this dissertation. It is also not just specific to Habit Reinforcement Learning and could be useful in any off-policy algorithm that uses macro-actions. [8]

[7]Note that this is different to what was done in Lange and Faisal, 2019 where they only reimagine using the first way.

[8]We provide a Python implementation of HAR at https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/utilities/Memory_Shaper.py

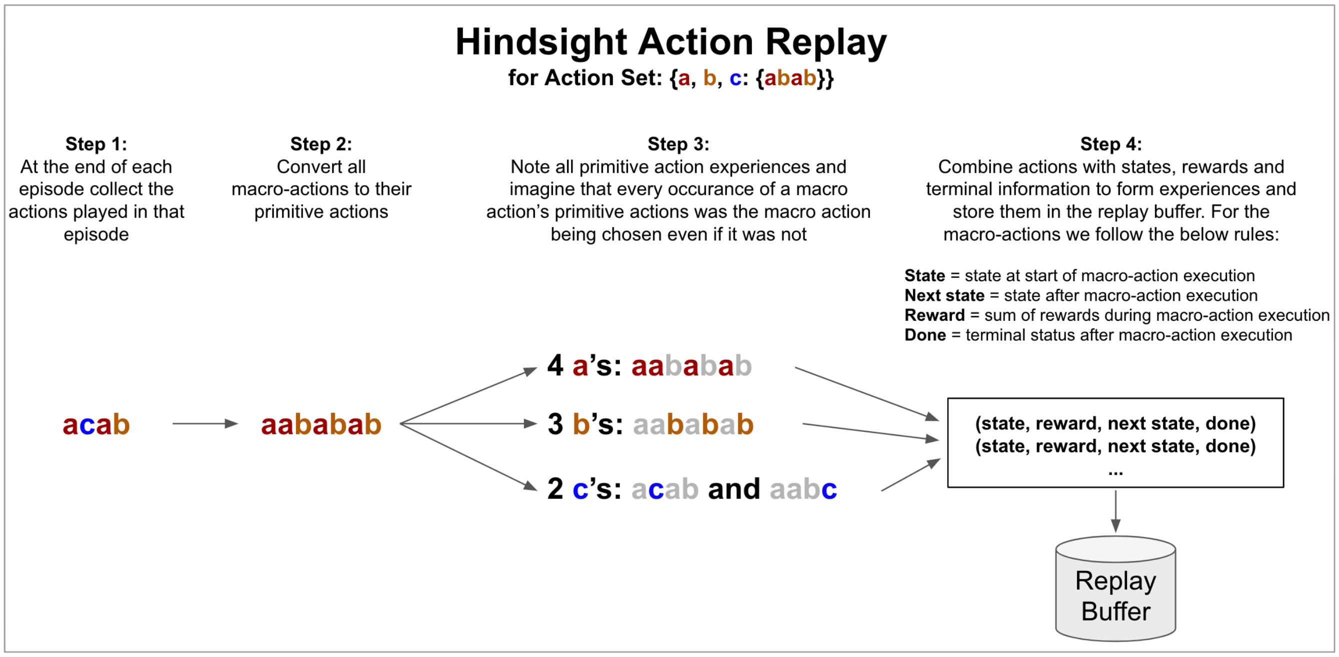
Hindsight Action Replay
for Action Set: {a, b, c: {abab}}
Step 1:
At the end of each episode collect the actions played in that episode
Step 2:
Convert all macro-actions to their primitive actions
Step 3:
Note all primitive action experiences and imagine that every occurance of a macro action's primitive actions was the macro action being chosen even if it was not
Step 4:
Combine actions with states, rewards and terminal information to form experiences and store them in the replay buffer. For the macro-actions we follow the below rules:
State = state at start of macro-action execution
Next state = state after macro-action execution
Reward = sum of rewards during macro-action execution
Done = terminal status after macro-action execution
acab
aababab
4 a's: aababab
3 b's: aababab
2 c's: acab and aabc
(state, reward, next state, done)
(state, reward, next state, done)
...
Replay Buffer

Figure 6: The Hindsight Action Replay (HAR) process with Action Set: {a, b, c:{abab}} meaning that there are 2 primitive actions "a" and "b", and one macro-action "c" which represents the sequence of primitive actions "abab". The example shows how using HAR leads to a sequence of 4 actions producing 9 experiences for the replay buffer instead of only 4.

vi) Even with HAR there can sometimes be many more observations of primitive actions than long macro-actions in our replay buffer. To make sure that the longer macro-actions receive enough attention we also use an **action balanced replay buffer**. [9] An action balanced replay buffer works like an ordinary replay buffer except it returns samples of experiences with equal amounts of each action. This makes sure that longer macro-actions receive enough attention when learning.

vii) Another problem we encountered with long macro-actions is that towards the end of executing a long macro-action we can sometimes end up in situations where it would be very beneficial to abandon the macro-action and change our course of action. This can happen for example if during execution the environment changes in an unexpected way. To deal with this problem we develop another novel technique called **Abandon Ship**. Abandon Ship works as follows: during every timestep of conducting a macro-action we compare the q-value of the primitive action we are about to conduct as part of the macro-action to the q-value of the primitive action with the highest q-value. If the "difference" is greater than some threshold then we abandon the macro-action and instead give control back to the policy to choose the next action as it normally would. We note that this is also similar to how human habits work - when executing a habit we do not finish executing it no matter what, instead we will abandon the habit if we encounter something unexpected.

For the technique we define the "difference" in q-values between action "a" and "b" as:

$$\text{Diff(a, b)} := 1 - \frac{\exp(a)}{\exp(b)} \tag{16}$$

To deal with the changing scale of q-values throughout training we also relate the threshold to the mean and standard deviation of recent *Diff(a, b)* values we have calculated:

$$\text{Threshold} := \text{mean(Diff)} - \text{std(Diff)} * \text{abandon\_ship\_hyperparm} \tag{17}$$

This way our hyperparameter *abandon_ship_hyperparm* can remain constant throughout training while our threshold changes to match the changing scale of the q-values.

[9]We provide a Python implementation of an action balanced replay buffer at https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/utilities/data_structures/Action_Balanced_Replay_Buffer.py.

Table 3: Demonstrates what happens when playing macro-action "aaa" with *abandon_ship_hyperparm* equal to 1 and a primitive action set of $\{a, b\}$. For the first 2 timesteps, even though action "a" has a lower q-value than "b" we still play action "a" because the difference is smaller than the threshold. In timestep 3 though the difference is too large and so we abandon the macro-action and instead choose our next action using our policy as normal. Note that the policy still may not choose "b" as the next action because there could be other macro-actions in the action set with higher q-values.

| | ***t=1*** | ***t=2*** | ***t=3*** |
|---|---|---|---|
| **Q-value of "a"** | **1.0** | **1.0** | **1.0** |
| **Q-value of "b"** | **1.1** | **1.1** | **5.0** |
| **Diff(a, b)** | **-0.11** | **-0.11** | **-53.6** |
| **Rolling mean(Diff)** | **-0.4** | **-0.39** | **-0.38** |
| **Rolling std(Diff)** | **0.1** | **0.09** | **0.09** |
| **Threshold** | **-0.50** | **-0.48** | **-0.47** |
| **Abandon Ship?** | **No** | **No** | **Yes** |
| **Action Taken** | **"a"** | **"a"** | **TBD by Policy** |

Table 3 goes through an example. Say we are conducting macro action $aaa$. First we conduct $a$ without calculating any differences as it is the first action. For our second move of $a$ we calculate *Diff(a, b)* to be -0.11. We then calculate the updated threshold value of $-0.39 - 1 * 0.09 = -0.48$ where the number 1 in the equation represented our hyperparameter *abandon_ship_hyperparm* and so could have been set to some other number. Because $-0.11 > -0.48$ we do not abandon ship and so proceed with playing our second "a" move. For the final "a" move we go through the same process but find that *Diff(a, b)* is now lower than the threshold. We therefore abandon the macro-action at this point and return control back to the policy which picks an action again in its normal way. Note that we only abandon the macro-action when the difference in q-values is large enough rather than there needing to be any difference at all. If we abandoned whenever there is any difference at all we would in effect always be picking the max q-value primitive action and so we effectively would not be using macro actions.

Finally, Table 4 provides a summary of the differences between the methodology of Habit-RL and that of its closest counterparts Lange and Faisal (2018) and Lange and Faisal (2019) to avoid any confusion and illuminate the most novel aspects of Habit-RL. Now we combine all these extensions to form the Habit Reinforcement Learning algorithm.

### 5.2.3 Algorithm

Algorithm 2 lays out the full Habit Reinforcement Learning algorithm. First we play some episodes in the environment with the environment's original action set. While doing so we occasionally play an episode with all exploration turned off - the frequency that this happens is determined by the "no exploration time" condition. At the end of the first set of episodes we feed the no-exploration experiences into the INFER_HABITS method. This method feeds the best performing of such episodes into IGGI which infers the habits of the agent. We then append these habits to the action set in the form of macro-actions. Then we play another set of episodes with the new action set. Because there are now macro-actions in our action set, we now also use HAR and Abandon Ship when storing experiences and conducting actions.

When using DDQN as the base RL algorithm we call the algorithm Habit-DDQN. Habit-DDQN follows the same procedure as in Algorithm 2 except:

- The base RL algorithm is the Double Deep Q-Learning algorithm [Hasselt, Guez, and Silver, 2015]
- When playing a random move (which happens epsilon proportion of the time with a DDQN) we pick macro-actions with a higher probability than primitive actions. The difference in probability is given by the

**Algorithm 2** Habit-RL

```
1:  Initialise environment env, base RL algorithm R, replay buffer D and action set A
2:  for each iteration do
3:      $F \leftarrow$ PLAY_N_EPISODES($A$)
4:      $A \leftarrow$ INFER_HABITS($F$)
5:
6:  procedure PLAY_N_EPISODES(A)
7:      transfer_learning(A)                  ▷ If action set has changed do transfer learning on all networks
8:      $F \leftarrow \emptyset$                                   ▷ Initialise F to store no-exploration episode experiences
9:      for each episode do
10:         if no exploration time then turn off exploration       ▷ Periodically turn off exploration for an episode
11:         $E \leftarrow \emptyset$                                    ▷ Initialise E to store an episode's experiences
12:         while not done do
13:             $ma_t$ = R.pick_action($s_t$)                      ▷ Pick next primitive action / macro-action
14:             for $a_t$ in $ma_t$ do                 ▷ Iterate through each primitive action in the macro-action
15:                 if abandon_ship($s_t$, $a_t$) then break      ▷ Abandon suggested action if too disadvantageous
16:                 $s_{t+1}$, $r_{t+1}$, $d_{t+1}$ = env.step($a_t$)                      ▷ Play action in environment
17:                 $E \leftarrow E \cup \{(s_t, a_t, r_{t+1}, s_{t+1}, d_{t+1})\}$         ▷ Store the episode's experiences
18:                 if enough steps passed then R.learn(D)     ▷ Run learning iteration for the base RL algorithm
19:         $D \leftarrow D \cup$ HAR($E$)                               ▷ Use HAR when updating replay buffer
20:         if no exploration time then $F \leftarrow F \cup E$                   ▷ Store no-exploration experiences
21:     return F
22:
23: procedure INFER_HABITS(F)
24:     $F \leftarrow$ extract_best_episodes(F)          ▷ Keep only the best performing no-exploration episodes
25:     habits $\leftarrow$ IGGI(F)                                ▷ Infer the habits using experiences seen so far
26:     $A \leftarrow A \cup$ habits                                  ▷ Update the action set with identified habits
27:     return A
```

"macro-action exploration bonus" hyperparameter. When macro-actions are chosen randomly in this way we also do not use Abandon Ship and instead let them fully execute

Similarly, when using the discrete action version of SAC derived earlier as the base RL algorithm we call the algorithm Habit-SAC. Habit-SAC follows the same procedure as in Algorithm 2 except:

- The base RL algorithm is the discrete action version of SAC (Algorithm 1)
- Immediately after we introduce new habits we play randomly for a small number of steps given by the "post habit inference random step" hyperparameter. When playing randomly we do not use Abandon Ship and also we pick macro-actions with a higher probability given by the "macro-action exploration bonus".

Our Python implementation of Habit-DDQN and Habit-SAC is provided in the dissertation's GitHub repository.

## 5.3 Results

We first evaluate the ideas of Habit-RL using Habit-DDQN. We chose DDQN as the base algorithm because it is one of the simplest off-policy deep RL algorithms that is able to learn complex environments. We test Habit-DDQN on the 8 Atari games: Qbert, Seaquest, MsPacman, Pong, Space Invaders, Breakout, Beam Rider and Enduro. The games were firstly chosen as they represent a broad range of the Atari games. Secondly, to dramatically speed up training times, for the convolutional layers of the Q-Network we use the frozen convolutional layers from a pre-trained network. This means we only need to learn the weights of the fully connected layers that follow and so allows us to speed up the training time required to get a reasonable score from 12+ hours to 1 hour. Then, the 8 games were also chosen because they were the only games for which pre-trained DDQN agents were available at the RL baselines zoo repository. [10] Also note that the games were chosen before the algorithm was tested so as to avoid any bias in their selection.

We ran the algorithms for 350,000 steps for both DDQN and Habit-DDQN and used the same evaluation procedure as in Hasselt, Guez, and Silver (2015) with 5 random seeds to account for variance in performance between runs. We chose

[10] https://github.com/araffin/rl-baselines-zoo/tree/master/trained_agents/dqn Rainbow etc.

Table 4: Differences between Habit Reinforcement Learning and its closest counterparts Lange and Faisal (2018) and Lange and Faisal (2019). Dark purple indicates a difference and green indicates a similarity.

| | **Habit Reinforcement Learning** | **Macro Deep Q Learning with CFG Production Rules** *(Lange and Faisal 2019)* | **Macro Q Learning with CFG Production Rules** *(Lange and Faisal 2018)* |
|---|---|---|---|
| ***Iterates Between Gathering Experience and Grammar/Habit Learning*** | Yes | Yes | |
| ***Base RL Algorithm*** | DDQN and SAC | Deep Q-Learning | Tabular Q-Learning |
| ***No-Exploration Episodes to Infer Grammar/Habits From*** | Best performing | Any | |
| ***Grammar Calculator*** | IGGI | Sequitur | Sequitur & G-Lexis |
| ***Max Macro-Actions Added at a Time*** | Unlimited | 2 | Unlimited |
| ***Transfer Learning*** | All layers | Only layers before final layer | None |
| ***Use of Hindsight to Store Experiences*** | Both to convert macro-actions to primitive actions and vice versa | Only to convert macro-actions to primitive actions | No |
| ***Abandon Ship*** | Yes | No | |
| ***Replay Buffer*** | Action Balanced Replay Buffer | Ordinary Replay Buffer | |
| ***Demonstrates Improvement in...*** | 27/28 Atari game settings | 0/1 simple games | 2/4 simple games |

350,000 steps because it was enough time for DDQN to make significant progress and achieve scores far above what a randomly playing agent would achieve, but not so much time that it severely restricted the amount of experiments we could run.

For the DDQN related hyperparameters we did no hyperparameter tuning and instead used the same hyperparameters from Hasselt, Guez, and Silver (2015) (including the same CNN architecture) besides i) having a learning rate that was twice as large to account for only having to learn the fully connected layers and ii) using the Adam optimizer instead of RMSProp because the authors found it to be less sensitive to the choice of learning rate in their later paper Hessel et al. (2017). For the Habit-DDQN specific hyperparameters we tried 4 options for the abandon ship hyperparameter (No Abandon Ship, 1, 2, 3) for the game Qbert and chose the option with the highest score. None of the other Habit-DDQN specific hyperparameters were tuned and all games then used the same hyperparameters. The hyperparameters can be found in Appendix B.

Overall therefore only 1 hyperparameter out of 20+ hyperparameters in 1 game out of 8 was tuned and so we consider very little hyperparameter tuning to have been done. This was a conscious decision because we had limited computation power but also because we wanted to discover an algorithm that was more likely to be robust to the choice of hyperparameters. We also note when analysing the results that it is very likely we could have achieved superior results with more hyperparameter tuning.

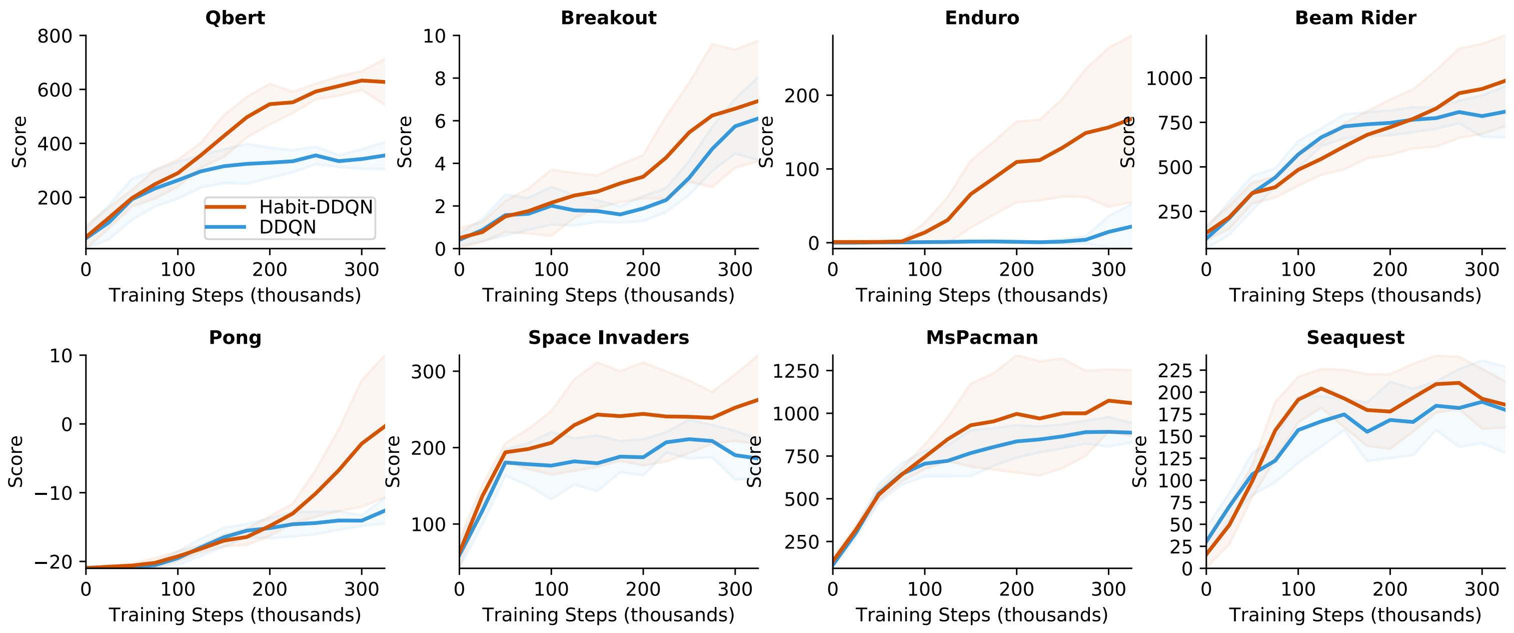

Figure 7: Comparing Habit-DDQN to DDQN for 8 Atari games. Graphs show the average evaluation score over 5 random seeds where an evaluation score is calculated every 25,000 steps and averaged over the previous 3 scores. The shaded area shows $\pm 1$ std.

In terms of results we found that **Habit-DDQN outperformed DDQN in all 8 games with a median final improvement of 31% and a maximum final improvement of 668%** - Figure 7 shows the results.

We note that often the confidence intervals overlap, however we estimate the probability that Habit-DDQN could achieve such higher scores than DDQN in all 8 games without being a superior algorithm for those 8 games as less than 0.00000001%. [11]

To further evaluate Habit-RL we also used the discrete action version of SAC we derived earlier as another base RL algorithm. We chose SAC because DDQN is no longer a state-of-the-art RL algorithm and so we wanted to check that Habit-RL also worked when applied to a more advanced RL algorithm. Additionally, DDQN is Q-Learning based so we wanted to test our algorithm with another type of base algorithm and so we chose SAC which is an actor-critic agent rather than a Q-learning agent. This is also why we did not choose Rainbow which is a Q-Learning agent. Finally, SAC is also particularly interesting because it is part of the growing trend of maximum entropy reinforcement learning.

Habit-DDQN involved using the frozen convolutional layers of a pre-trained q-network which we worried could potentially be skewing the results and so with Habit-SAC we avoided this and instead trained from scratch. Then, because we no longer relied on a pre-trained convolutional base we were able to test Habit-SAC on more games than we were with Habit-DDQN. We kept the 8 games from before and added 12 more games that we felt reflected the diversity of the Atari suite.[12] J. Oh and Lee (2018) suggest a way of categorising Atari games into 3 groups: Easy Exploration, Hard Exploration (Dense Reward) and Hard Exploration (Sparse Reward) - we note that we chose games that fall into all 3 categories.

We ran the algorithms for 100,000 steps because Kaiser et al. (2019) showed that this was enough time for Rainbow to make significant progress on Atari games and we showed earlier that SAC with discrete actions performs similarly to

[11]To estimate this we first assumed the DDQN scores followed a normal distribution. Then we set the null hypothesis to be that the distribution of returns for DDQN and Habit-DDQN were the same. Then we calculated the probability of seeing such differences in scores under the null hypothesis. Note that this is only a rough approximation as our assumption of normality may not hold in reality but we believe it is accurate enough for the point we are trying to make.

[12]We also only chose games where there was a reasonable chance of the agent making progress within 100,000 steps - we excluded Montezuma's Revenge for this reason

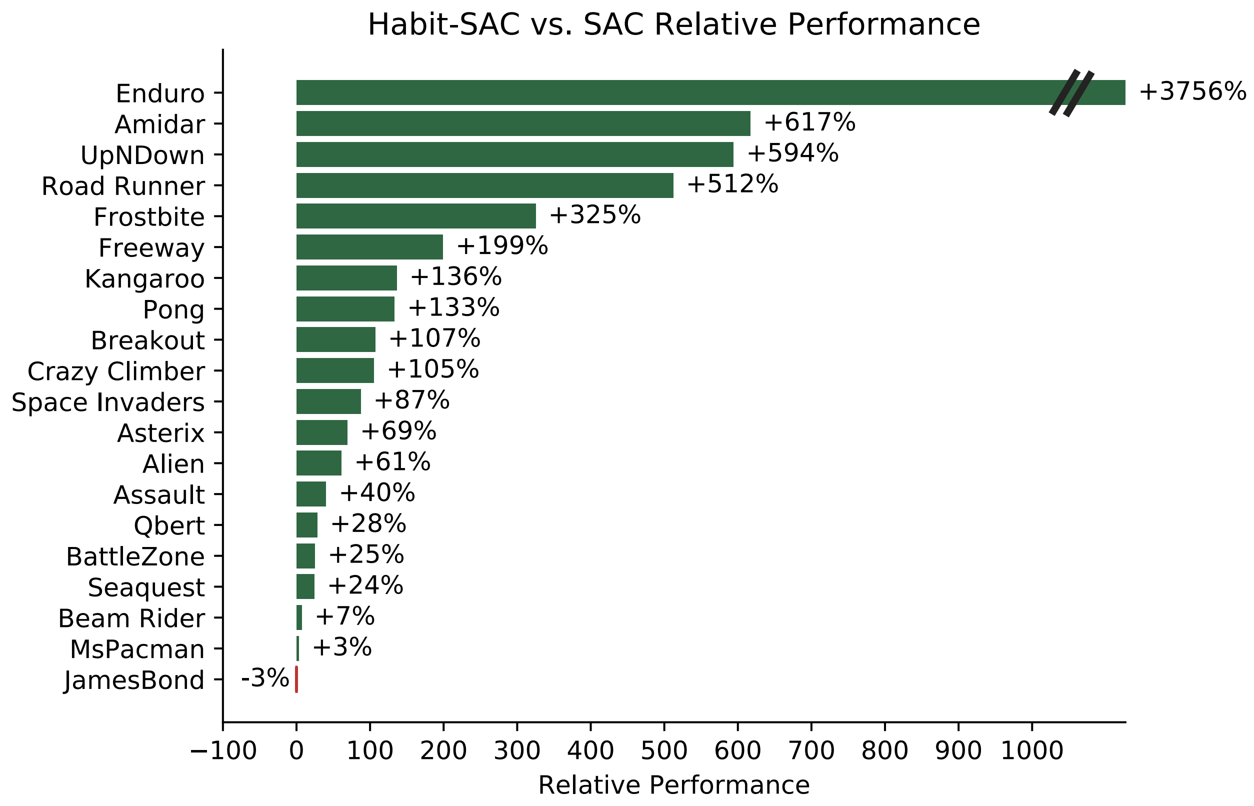

Figure 8: Comparing Habit-SAC to SAC for 20 Atari games. Graphs show the average evaluation score over 5 random seeds where an evaluation score is calculated at the end of 100,000 steps of training

Rainbow. Kaiser et al. (2019) were also exploring the same question of sample complexity as us and this is what they used. We also used the same evaluation procedure as in Kaiser et al. (2019). We ran this for both SAC and Habit-SAC and compared the results.

For the SAC specific hyperparameters we did no hyperparameter tuning and instead used a mixture of the hyperparameters found in Haarnoja et al. (2019) and Kaiser et al. (2019). For the Habit-SAC specific hyperparameters the only tuning we did was amongst 4 options for the abandon ship hyperparameter (No Abandon Ship, 1, 2, 3) on the game Qbert. All games then used the exact same hyperparameters. The hyperparameters can be found in Appendix C. Overall therefore, again almost no hyperparameter tuning was done and so it is highly likely we could have achieved even better results by tuning the hyperparameters.

Remarkably, we found that **Habit-SAC outperformed SAC in 19 out of 20 games with a median improvement of 96%, maximum improvement of 3756%** - Figure 8 summarises the results and Appendix A provides them in a table.

Again we estimate that the probability Habit-SAC would outperform SAC in 19 out of 20 games without it being a superior algorithm for those games as less than 0.00000001%. [13] We also note that for the only game with a negative score, the score (-3%) is well within one standard deviation ($\pm 38\%$) of performance for that game across the 5 random seeds and so there is a significant chance that the negative score was because of our random choice of seeds.

Lastly, even though this was not an aim of the project, we note that **Habit-SAC outperforms the tuned Rainbow scores from Kaiser et al. (2019) in 17 of the 20 games (median improvement 62%, max improvement 13140%)**. This makes Habit-SAC the state-of-the-art model-free RL algorithm in terms of sample efficiency for those 17 games. This all without any substantive hyperparameter tuning compared to the Rainbow scores which were supported by extensive hyperparameter tuning. Figure 9 summarises the results and Appendix A provides them in a table.

Appendix E explains how to replicate the results found in this section and now we move on to discuss the results in more detail.

[13] We calculated this the same way as for Habit-DDQN.

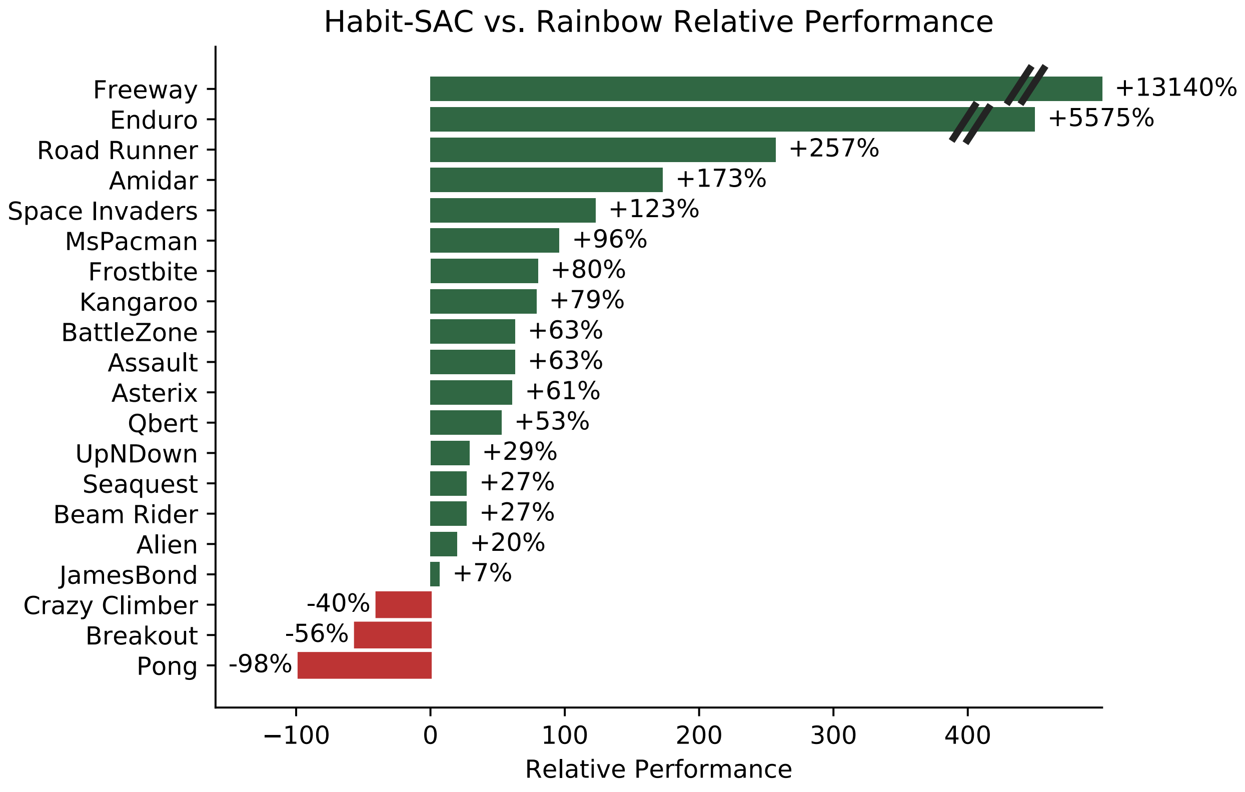

Figure 9: Comparing Habit-SAC to Rainbow for 20 Atari games. Graphs show the average evaluation score over 5 random seeds where an evaluation score is calculated at the end of 100,000 steps of training. Note that the Rainbow scores are as a result of extensive hyperparameter tuning while the Habit-SAC scores involved very little hyperparameter tuning.

### 5.4 Discussion

#### 5.4.1 The Habits

The first thing we explore is what kinds of habits get inferred during the Identify Habits stage and whether the agents use them extensively or not.

Table 5: The final action set in an example Habit-SAC game of Beam Rider. The table contains the game's 9 primitive actions (from 0 to 8) and the agent's 26 inferred habits.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 5 | 44 | 1111 | 55555555 | 1111111111111111 | 44444444444444444444444444444444 |
| 1 | 6 | 55 | 2211 | 77117711 | 6666666666666666 | 66666666666666661111111111111111 |
| 2 | 7 | 66 | 5511 | 77777777 | 7777777711111111 | 77777777111111111111111111111111 |
| 3 | 8 | 77 | 8888 | 88118811 | 8888111188881111 | 77777777777777777777777777777777 |
| 4 | 11 | 88 | 55116611 | 88888888 | 11111111111111111111111111111111 | 88888888888888888888888888888888 |

Table 6: The lengths of moves attempted and executed in the best performing Habit-SAC seed for each game. Executed move lengths being shorter than attempted move lengths indicates that the Abandon Ship technique was used. The games are ordered in terms of the percentage improvement Habit-SAC saw over SAC with the first game being the game that saw the most improvement.

| Game | Attempted Move Lengths | Executed Move Lengths | Executed / Attempted |
|---|---|---|---|
| Enduro | 14.8 | 12.0 | 81.1% |
| Amidar | 3.4 | 2.8 | 82.4% |
| UpNDown | 16.4 | 11.6 | 71.0% |
| Road Runner | 8.2 | 6.8 | 82.9% |
| Frostbite | 2.0 | 2.0 | 100.0% |
| Freeway | 15.8 | 15.2 | 96.2% |
| Kangaroo | 2.1 | 1.5 | 73.8% |
| Breakout | 7.9 | 2.2 | 27.8% |
| Crazy Climber | 7.1 | 4.4 | 62.0% |
| Space Invaders | 8.7 | 6.3 | 72.4% |
| Asterix | 58.3 | 9.8 | 16.8% |
| Alien | 13.1 | 11.6 | 88.5% |
| Assault | 126.9 | 8.1 | 6.4% |
| Qbert | 8.2 | 8.0 | 97.6% |
| Battle Zone | 63.4 | 6.0 | 9.5% |
| Seaquest | 14.0 | 8.6 | 61.4% |
| Beam Rider | 12.2 | 5.8 | 47.5% |
| MsPacman | 8.2 | 7.5 | 91.5% |
| Pong | 8.4 | 6.5 | 77.4% |
| James Bond | 1.3 | 1.2 | 97.8% |

We find that the habits inferred vary significantly between games but as an example we provide the final action set for a Habit-SAC game of Beam Rider in Table 5. In this game there are 9 primitive actions (from 0 to 8) - primitive action 0 has no impact on the game, action 1 means to Shoot, and the rest of the primitive actions are directions of movement (Up, Right, Left, Down, Up-Right, Up-Left, Down-Right, Down-Left respectively). There are 35 actions in the action set therefore 26 habits were inferred. We notice that the habits cover quite a wide range of sizes (from length 2 to length 32). One long habit inferred for example was 8888111188881111 which involves alternating between Down-Right and Shoot - this move seems particularly useful in this game as the game is about shooting enemies whilst avoiding being hit by them.

Then, Table 6 shows the attempted and executed move lengths during the evaluation procedure for the seed that achieved the highest Habit-SAC score for each game. The first thing to notice is that the move lengths are significantly above 1 and so the agents are certainly making use of their habits. During the evaluation steps there is no mechanism that encourages the agent to use the macro-actions and so they are using them because their policy is identifying them as the highest value actions. The highest attempted move length is 127 which shows us that the habits can be extremely long. We also note that the size of macro-actions being used varies wildly between games. The games are extremely varied and so this is to be expected and implies the process of inferring habits is catering to the differences in the games.

Next we note that the Abandon Ship technique gets used very often. The median executed move length as a proportion of attempted move length is 76% and so many macro-actions are being abandoned before they complete. This proportion also varies widely by game, being as low as 6.4% for the game Assault and as high as 100.0% for Frostbite. This variance perhaps reflects the differing levels of predictability across the Atari games - for some games it is very easy to predict future states and so we would expect Abandon Ship to be used less often and for others the opposite is true.

Now we investigate what aspects of Habit-RL in particular are most responsible for the results.

### 5.4.2 Ablation Study

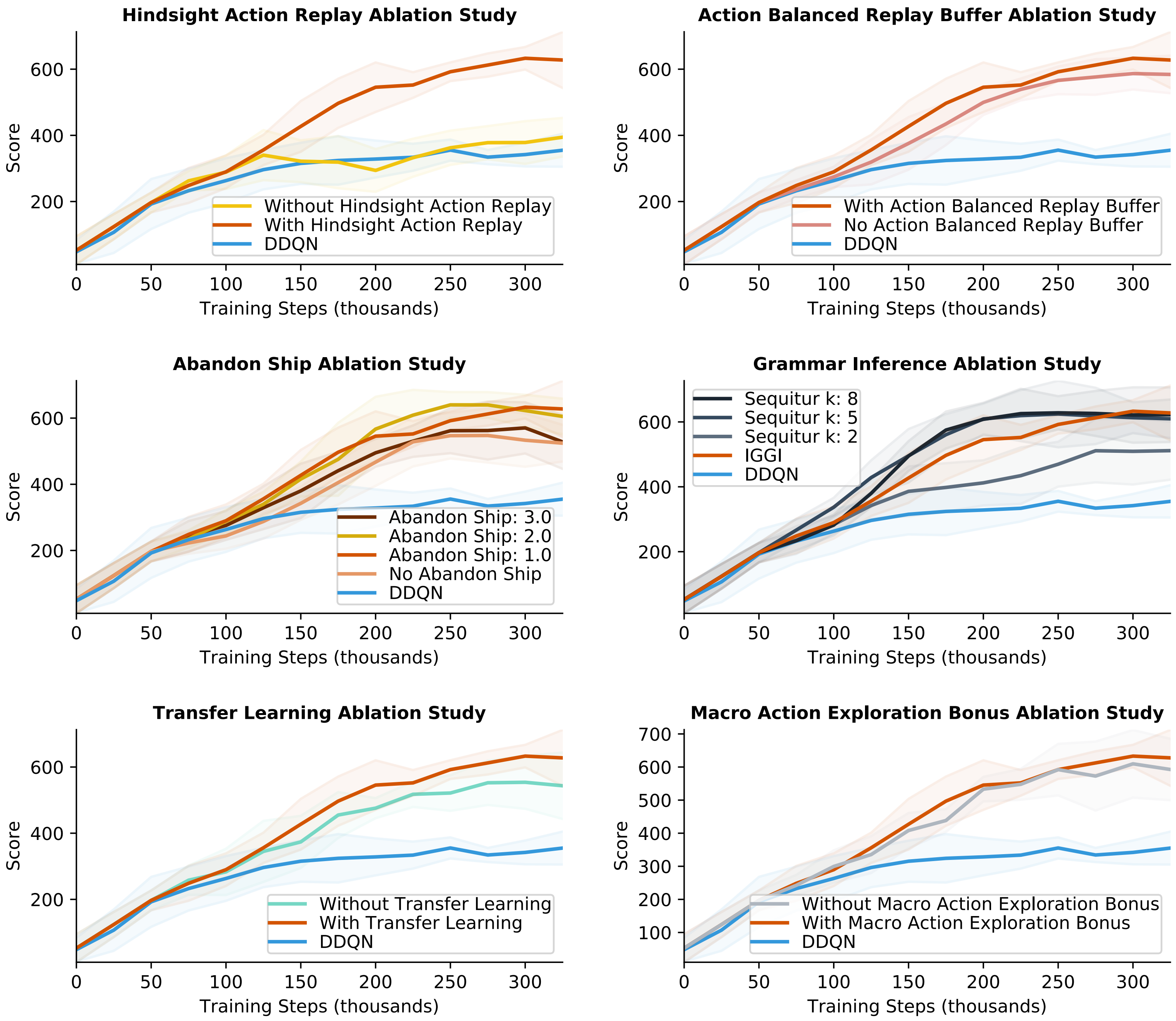

Figure 10: Ablation study analysing the impact of different variables on the performance of Habit-DDQN vs. DDQN in the Atari game Qbert. The thick lines are the average score over 5 random seeds and the shaded areas represent $\pm 1$

Figure 10 shows the results of an ablation study on the key variables involved in Habit-DDQN for the game Qbert.

Firstly, the top-left graph shows that Hindsight Action Replay is crucial for the algorithms performance and without it Habit-DDQN is unable to do better than DDQN. This is most likely because, as mentioned earlier, without HAR we often end up with macro-actions that we have very few observations for. Our estimate of their q-value then has extremely high variance which can drastically skew and disturb training.

Next in the top-right graph we see that using an action balanced replay buffer improved performance slightly. We believe it may have improved performance because even with HAR there is often a lot fewer experiences in the replay buffer for long macro-actions and so using an action balanced replay buffer helps make sure they receive significant attention when learning. However, there is significant overlap in the confidence intervals and so it arguably could have been unnecessary to use an action balanced replay buffer. Also, we did not try this because of a lack of time but suspect that using a prioritised experience replay buffer [Schaul et al., 2016] may instead be the best solution. It would likely be the best solution because it would use TD errors to determine which experiences require more attention rather than assuming that all actions require equal attention.

Thirdly, we in the middle-left graph we looked at how performance changes when we vary the Abandon Ship hyperparameter. We note that when Abandon Ship is turned off, performance is worse than when it is at 1.0 and 2.0 and so we believe Abandon Ship was likely a necessary feature of Habit-RL. Next we note that the scores for hyperparameter choice 1.0 and 2.0 are very similar which indicates performance is not too sensitive to this choice of hyperparameter.

Next we compare the performance when using different grammar calculators in the middle-right graph. We find that IGGI performs the best but that Sequitur-k with k in $\{5, 8\}$ performs roughly as well. We therefore may have been able to use Sequitur instead to get overall similar results. However, Sequitur comes with the extra drawback of having to tune an extra hyperparameter (the k) and so in a practical sense we believe the data still suggests IGGI is the best algorithm to use.

Then in the bottom-left graph we show that transfer learning appeared to improve performance somewhat as was to be expected.

Finally, in the bottom-right graph we see that performance is slightly better when a macro action exploration bonus is used. The confidence intervals overlap a lot however and so it may not have been necessary to include this in the algorithm.

We now make two additional points relating to exploration before moving on to the conclusion and future work.

### 5.4.3 Exploration

First of all, we notice that for both Habit-DDQN and Habit-SAC the game they do best on (compared to the baselines of DDQN and SAC) is Enduro. Enduro is a game where good exploration is very important as the rewards are very sparse. We therefore speculate that one of the reasons Habit-RL does particularly well at this game is that allowing the agents to use long macro-actions helps them explore more effectively. This makes sense because the longer the steps we take the larger the variance in terms of where we can end up when moving randomly, therefore the more of an environment we are likely explore.

Secondly, as mentioned in the methodology section, immediately after updating the action set with new habits, Habit-SAC runs some steps using only randomly selected actions. It is therefore a worry that some of the superior performance of Habit-SAC over SAC could have been because of the extra exploration induced by randomly running these extra steps rather than the habit-related aspects of Habit-SAC themselves. To investigate this we ran a second version of SAC that runs the same number of extra random steps at the same stage in training as Habit-SAC. We ran this version of SAC on Qbert and Enduro and found it had a +4% and -86% impact respectively compared to the original SAC. It therefore does not seem to explain the +28% and +3756% scores Habit-SAC got for the same games and so we do not believe this is a significant factor.

Finally, we spent many hours investigating interesting ideas that did not seem to work and never made it into the final algorithm but that we believe there is value in mentioning. We do this in Appendix D and now we move onto the conclusion and future work.

## 6 Conclusion and Future Work

Motivated by the contrast between the importance of habits to human behaviour and potentially learning, and their lack of representation in RL algorithms, we developed the Habit Reinforcement Learning framework. The Habit Reinforcement Learning framework introduces the concept of habits to the field of Reinforcement Learning in a way that consistently and significantly improves sample efficiency.

We provided two implementations of the framework: Habit-DDQN and Habit-SAC. We showed that Habit-DDQN improves on DDQN in 8 out of 8 Atari settings (median +31%, max +668%) and Habit-SAC improves on SAC in 19 out of 20 Atari settings (median +96%, max +3756%) all without substantive hyperparameter tuning. We also show that Habit-SAC is the model-free state-of-the-art for 17 out of 20 Atari games in terms of sample efficiency, again even without substantive hyperparameter tuning.

As part of Habit-RL we provided two new and generally applicable techniques: Hindsight Action Replay and Abandon Ship. Hindsight Action Replay can be used to drastically improve information efficiency (sometimes by over 100%) in any off-policy setting involving macro-actions. Abandon Ship reduces the variance involved when training macro-actions, making it feasible to train algorithms with very long macro-actions (over 100 steps in some cases). It can also be used in any RL setting with macro-actions.

We also derived a discrete action version of the SAC algorithm that we show is competitive with the model-free state-of-the-art Rainbow algorithm on Atari games even without any hyperparameter tuning. Finally, we also provide public Python implementations in the project's GitHub repository of Habit-RL, Habit-DDQN, Habit-SAC, SAC for discrete actions, an action balanced replay buffer, Hindsight Action Replay, Abandon Ship, IGGI and Sequitur-k.[14]

We believe there are countless promising avenues for future work that we are excited to work on as well as to see others work on. We see some of the most promising avenues as:

- Applying Habit-RL to other off-policy algorithms such as Rainbow to ensure our results were not only specific to DDQN and SAC
- Making Habit-RL compatible and applying it to on-policy algorithms such as A3C
- Extending Habit-RL to environments with continuous actions. We suspect this may involve a form of clustering to discretise actions
- Testing Habit-RL on the other games in the Atari suite we were unable to evaluate this time because of limited resources
- Investigating how robust the benefits of Habit-RL are to longer training times. In particular we would be interested to see whether the quicker initial learning performance of Habit-RL follows through into better performance when training for much longer
- Testing the robustness of Habit-RL on different types of games (e.g. games with high levels of stochasticity) to ensure our results were not only specific to the Atari suite
- Investigating whether it is possible to remove primitive actions from the action set as we append more habits to the action set. Currently the original primitive actions always remain in the action set but it could be beneficial and possible to remove them
- Investigating other ways of incorporating habits into RL. For example, there is likely an alternative method that involves evaluating how close each state is to a past state we have seen and if it is close enough to a commonly seen state then we react by executing the relevant habit

Overall, we believe habits are a surprising but extremely important reason why humans are so efficient at learning and so their lack of representation in RL research so far presents an enormous opportunity to progress the field. We have provided one way of incorporating habits into RL algorithms and shown that it significantly and consistently improves sample efficiency. We do not believe this is the only way of incorporating habits and look forward to investigating and discovering other methods. Eventually we hope that habits can do for RL what CNNs did for image processing and RNNs for sequence processing by dramatically improving RL's sample efficiency in a way that helps make RL a universally practical and useful tool in modern society.

[14]https://github.com/p-christ/Habit-Reinforcement-Learning

## References

Abramova, E., L. Dickens, D. Kuhn, and A. Faisal (2019). *RLOC: Neurobiologically Inspired Hierarchical Reinforcement Learning Algorithm for Continuous Control of Nonlinear Dynamical Systems*. eprint: `https://arxiv.org/abs/1903.03064`.

Andrychowicz, M., B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba (2018). "Learning Dexterous In-Hand Manipulation". In: *CoRR*.

Andrychowicz, M., F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba (2017). "Hindsight Experience Replay". In: *Neural Information Processing Systems*.

Arulkumaran, K., M. P. Deisenroth, M. Brundage, and A. A. Bharath (2017). "A Brief Survey of Deep Reinforcement Learning". In: *IEEE Signal Processing Magazine*.

Bacon, P., J. Harb, and D. Precup (2016). "The Option-Critic Architecture". In: *Association for the Advancement of Artificial Intelligence*.

Bellemare, M., W. Dabney, and R. Munos (2017). "A Distributional Perspective on Reinforcement Learning". In: *International Conference on Machine Learning*.

Bellemare, M., Y. Naddaf, J. Veness, and M. Bowling (2013). "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence*.

Bengio, Y., J. Louradour, R. Collobert, and J. Weston (2009). "Curriculum Learning". In: *International Conference on Machine Learning*.

Beyret, B., A. Shafti, and A. Faisal (2019). *Dot-to-Dot: Achieving Structured Robotic Manipulation through Hierarchical Reinforcement Learning*. eprint: `https://arxiv.org/abs/1904.06703`.

Botvinick, M., S. Ritter, J. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis (2019). "Reinforcement Learning, Fast and Slow". In: *Trends in Cognitive Sciences*.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). *OpenAI Gym*. eprint: `arXiv:1606.01540`.

Castro, P., S. Moitra, C. Gelanda, S. Kumar, and M. Bellemare (2018). "Dopamine: A Research Framework for Deep Reinforcement Learning". In: *arXiv preprint*.

Cohen, N. and A. Shashua (2017). "Inductive Bias of Deep Convolutional Networks Through Pooling Geometry". In: *International Conference on Learning Representations*.

Dezfouli, A. and B. Balleine (2010). "Human and Rodent Homologies in Action Control: Corticostriatal Determinants of Goal-Directed and Habitual Action". In: *American College of Neuropsychopharmacology*.

– (2012). "Habits, Action Sequences, and Reinforcement Learning". In: *The European journal of neuroscience*.

Dickinson, A. (1994). "Instrumental Conditioning". In: *Animal Cognition and Learning*.

Ding, N., L. Melloni, X. Tian, and D. Poeppel (2012). "Rule-based and Word-level Statistics-based Processing of Language: Insights from Neuroscience". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences*.

Duan, Y., X. Chen, R. Houthooft, J. Schulman, and P. Abbeel (2016). "Benchmarking Deep Reinforcement Learning for Continuous Control". In: *International Conference on Machine Learning*.

Florensa, C., Y. Duan, and P. Abbeel (2017). "Stochastic Neural Networks for Hierarchical Reinforcement Learning". In: *International Conference on Learning Representations*.

Fortunato, M., M. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg (2017). "Noisy Networks for Exploration". In: *CoRR*.

Fujimoto, S., H. van Hoof, and D. Meger (2018). "Addressing Function Approximation Error in Actor-Critic Methods". In: *arXiv preprint*.

Haarnoja, T., A. Zhou, P. Abbeel, and S. Levine (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Learning Representations*.

Haarnoja, T., A. Zhou, K. Hartikainen, G. Tucker, J. Tan S. Ha, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine (2019). "Soft Actor-Critic Algorithms and Applications". In: *arXiv preprint*.

Hasselt, H. van, A. Guez, and D. Silver (2015). "Deep Reinforcement Learning with Double Q-learning". In: *Association for the Advancement of Artificial Intelligence*.

Hessel, M., J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver (2017). "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *arXiv preprint*.

Houthooft, R., X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel (2017). "VIME: Variational Information Maximizing Exploration". In: *Neural Information Processing System*.

J. Oh Y. Guo, S. Singh and H. Lee (2018). "Self-Imitation Learning". In: *International Conference on Machine Learning*.

Kaiser, L., M. Babaeizadech, P. Milos, B. Osinski, R. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski (2019). "Model Based Reinforcement Learning for Atari". In: *arXiv preprint*.

Kalman, R. (1960). "Contributions to the Theory of Optimal Control". In: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.4070&rep=rep1&type=pdf`.

Kingma, D. and M. Welling (2013). "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations*.

Konidaris, G., S. Kuindersma, R. Grupen, and A. Barto (2011). "Autonomous Skill Acquisition on a Mobile Manipulator". In: *Association for the Advancement of Artificial Intelligence*.

Kulkarni, T., K. Narasimhan, A. Saeedi, and J. Tenebaum (2016). "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Neural Information Processing System*.

Lange, R. and A. Faisal (2018). "Action Grammars: A Grammar Induction-Based Method for Learning Temporally-Extended Actions". In: `https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1718-pg-projects/LangeR-Action-Grammars.pdf/`.

– (2019). "Action Grammars: A Cognitive Model for Learning Temporal Abstractions". In: *arXiv preprint*.

LeCun, Y. and Y. Bengio (1995). "Convolutional Networks for Images, Speech, and Time Series". In: *The Handbook of Brain Theory and Neural Networks*.

Levy, A., G. Konidaris, R. Platt, and K. Saenko (2019). "Learning with Multi-Level Hierarchies with Hindsight". In: *International Conference on Learning Representations*.

Lewis, M. (2016). "Addiction and the Brain: Development, Not Disease". In: *Neuroethics*.

Lillicrap, T., J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2016). "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations*.

Mnih, V., A. Badia, M. Mirza, A. Graves, T. Harley, T. Lillicrap, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous Methods for Deep Reinforcement Learning". In: *International Conference on Maching Learning*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *Neural Information Processing Systems*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Peterson, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis (2015). "Human-Level Control Through Deep Reinforcement Learning". In: *Nature*.

Nachum, O., S. Gu, H. Lee, and S. Levine (2018). "Data-Efficient Hierarchical Reinforcement Learning". In: *Neural Information Processing Systems*.

Neu, G., A. Jonsson, and V. Gomez (2017). *A Unified View of Entropy-Regularized Markov Decision Processes*. eprint: `https://arxiv.org/abs/1705.07798`.

Nevill-Manning, C. and I. Witten (1997). "Identifying Hierarchical Structure in Sequences: A Linear-Time Algorithm". In: *CoRR*.

Osa, T., V. Tangkaratt, and M. Sugiyama (2019). "Hierarchical Reinforcement Learning via Advantage-Weighted Information Maximization". In: *International Conference on Learning Representations*.

Pashevich, A., D. Hafner, J. Davidson, R. Sukthankar, and C. Schmid (2018). *Modulated Policy Hierarchies*. eprint: `https://arxiv.org/abs/1812.00025`.

Pastra, K. and Y. Aloimonos (2012). "The Minimalist Grammar of Action". In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences*.

Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2016). "Prioritized Experience Replay". In: *International Conference on Learning Representations*.

Schoenhense, B. and A. Faisal (2017). "Data-Efficient Inference of Hierarchical Structure in Sequential Data by Information-Greedy Grammar Inference". In: *arXiv preprint*.

Schulman, J., S. Levine, P. Moritz, M. Jordan, and P. Abbeel (2015). “Trust Region Policy Optimization”. In: *International Conference on Machine Learning*.

Sharma, S., A. Lakshminarayanan, and B. Ravindran (2017). “Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning”. In: *International Conference on Learning Representations*.

Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, and D. Hassabis (2017). “Mastering the Game of Go Without Human Knowledge”. In: *Nature*.

Sing, S., A. Barto, and N. Chentanez (2005). “Intrinsically Motivated Reinforcement Learning”. In: *Advances in Neural Information Processing Systems*.

Siyari, P., B. Dilkina, and C. Dovrolis (2016). “Lexis: An Optimization Framework for Discovering the Hierarchical Structure of Sequential Data”. In: *Knowledge Discovery and Data Mining*.

Stout, D., A. Chaminade, A. Thomik, J. Apel, and A. Faisal (2018). “Grammars of Action in Human Behavior and Evolution”. In: *bioRxiv*.

Sutton, R. (1988). “Learning to Predict by the Methods of Temporal Differences”. In: *Machine Learning*.

Sutton, R. and A. Barto (2017). “Reinforcement Learning: An Introduction”. In: *The MIT Press*.

Sutton, R., D. Precup, and S. Singh (1999). “Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artificial Intelligence*.

Vezhnevets, A., V. Mnih, J. Agapiou, S. Osindero, A. Graves, O. Vinyals, and K. Kavukcuoglu (2016). “Strategic Attentive Writer for Learning Macro-Actions”. In: *Neural Information Processing Systems*.

Vezhnevets, A., S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu (2017). “FeUdal Networks for Hierarchical Reinforcement Learning”. In: *International Conference on Machine Learning*.

Wang, Z., T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas (2016). “Dueling Network Architectures for Deep Reinforcement Learning”. In: *arXiv preprint*.

Watkins, C. (1989). “Learning from Delayed Rewards”. In: *Ph.D thesis, Cambridge University*.

Watson, P., G. van Wingen, and S. de Wit (2018). “Conflicted between Goal-Directed and Habitual Control, an fMRI Investigation”. In: *eNeuro*.

Williams, R., G. Hinton, and D. Rumelhart (1986). “Learning Representations by Back-Propagating Errors”. In: *Nature*.

Wood, W., J. Quinn, and D. Kashy (2002). “Habits in Everyday Life: Thought, Emotion, and Action”. In: *Journal of Personality and Social Psychology*.

Wu, Y., E. Mansimov, S. Liao, R. Grosse, and J. Ba (2017). “Scalable Trust-Region Method for Deep Reinforcement Learning Using Kronecker-Factored Approximation”. In: *Advances in Neural Information Processing Systems*.

Yule, G. (2015). *The Study of Language*. Cambridge University Press.

Ziebart, B. (2010). “Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy”. In: *PhD dissertation, Carnegie Mellon University*.

## Appendix

## A SAC, Habit-SAC and Rainbow Atari Results

Below we provide all SAC, Habit-SAC and Rainbow results after running 100,000 iterations for 5 random seeds. We see that Habit-SAC improves over SAC in 19 out of 20 games (median improvement of 96%, maximum improvement of 3756%) and that Habit-SAC improves over Rainbow in 17 out of 20 games (median improvement 62%, maximum improvement 13140%).

Table 7: The SAC, Habit-SAC and Rainbow results on 20 Atari games referenced in Figure 8 and Figure 9. As a benchmark we also provide a column indicating the score an agent would get if it acted purely randomly.

| Game | Random | Rainbow | SAC | Habit-SAC | Habit-SAC vs. SAC | Habit-SAC vs. Rainbow |
|---|---|---|---|---|---|---|
| Enduro | 0.0 | 0.53 | 0.8 (0.8) | 30.1 (10.1) | +3756.4% | +5575.5% |
| Amidar | 11.8 | 20.8 | 7.9 (5.1) | 56.7 (15.4) | +617.3% | +172.7% |
| UpNDown | 488.4 | 1346.3 | 250.7 (176.5) | 1739.1 (835.8) | +593.8% | +29.2% |
| Road Runner | 0.0 | 524.1 | 305.3 (557.4) | 1868.7 (1658.3) | +512.0% | +256.6% |
| Frostbite | 74.0 | 140.1 | 59.4 (16.3) | 252.3 (79.8) | +324.7% | +80.1% |
| Freeway | 0.0 | 0.1 | 4.4 (9.9) | 13.2 (12.1) | +198.6% | +13140.0% |
| Kangaroo | 42.0 | 38.7 | 29.3 (55.1) | 69.3 (80.4) | +136.4% | +79.2% |
| Pong | -20.4 | -19.5 | $-20.98$ (0.0) | $-20.95$ (0.1) | +133.3% | -98.0% |
| Breakout | 0.9 | 3.3 | 0.7 (0.5) | 1.5 (1.4) | +106.6% | -55.8% |
| Crazy Climber | 7339.5 | 12558.3 | 3668.7 (600.8) | 7510.0 (3898.7) | +104.7% | -40.2% |
| Space Invaders | 148.0 | 135.1 | 160.8 (17.3) | 301.0 (75.1) | +87.2% | +122.8% |
| Asterix | 248.8 | 285.7 | 272.0 (33.3) | 459.0 (104.2) | +68.8% | +60.7% |
| Alien | 184.8 | 290.6 | 216.9 (43.0) | 349.7 (33.4) | +61.2% | +20.3% |
| Assault | 233.7 | 300.3 | 350.0 (40.0) | 490.6 (119.0) | +40.2% | +63.4% |
| Qbert | 166.1 | 235.6 | 280.5 (124.9) | 359.7 (172.8) | +28.2% | +52.7% |
| BattleZone | 2895.0 | 3363.5 | 4386.7 (1163.0) | 5486.7 (1461.7) | +25.1% | +63.1% |
| Seaquest | 61.1 | 206.3 | 211.6 (59.1) | 261.9 (56.3) | +23.8% | +26.9% |
| Beam Rider | 372.1 | 365.6 | 432.1 (44.0) | 463.0 (219.1) | +7.2% | +26.6% |
| MsPacman | 235.2 | 364.3 | 690.9 (141.8) | 712.9 (194.5) | +3.2% | +95.7% |
| JamesBond | 29.2 | 61.7 | 68.3 (26.2) | 66.3 (19.4) | -2.9% | +7.5% |

Note that the scores for Pong are negative and so to calculate proportional improvements we first convert the scores to their increment over the minimum possible score. i.e. in Pong the minimum score is -21.0 and so we first add 21 to all scores before calculating relative performance.

Also note that for Pong both Habit-SAC and SAC perform worse than random. The improvement of Habit-SAC over SAC for Pong therefore could be considered a somewhat spurious result and potentially should be ignored. Note that there are no other games where Habit-SAC performs worse than random though and so this issue is contained to the game Pong.

## B Habit-DDQN Hyperparameters

The hyperparameters used for the DDQN results are given by Table 8. The network architecture was the same as in the original Deepmind Atari paper Mnih et al. (2015).

Table 8: Hyperparameters used for DDQN and Habit-DDQN results

| **Hyperparameter** | **Value** |
|---|---|
| Batch size | 32 |
| Replay buffer size | 1,000,000 |
| Discount rate | 0.99 |
| Steps per learning update | 4 |
| Learning iterations per round | 1 |
| Learning rate | 0.0005 |
| Optimizer | Adam |
| Weight Initialiser | He |
| Min Epsilon | 0.1 |
| Epsilon decay steps | 1,000,000 |
| Fixed network update frequency | 10000 |
| Loss | Huber |
| Clip rewards | Clip to [-1, +1] |
| Initial random steps | 25,000 |

The architecture and hyperparameters used for the Habit-DDQN results that are relevant to DDQN are the same as for DDQN and then the rest of the hyperparameters are given by Table 9.

Table 9: Hyperparameters used for Habit-DDQN results

| **Hyperparameter** | **Value** | **Description** |
|---|---|---|
| Evaluation episodes | 5 | The number of no exploration episodes we use to infer the habits |
| Replay buffer type | Action balanced replay buffer | The type of replay buffer we use |
| Steps before inferring habits | 75,001 | The number of steps we run before inferring the habits |
| Abandon ship | 1.0 | The threshold for the abandon ship algorithm |
| Grammar algorithm | IGGI | The algorithm we use to infer the habits from the actions |
| Macro-action exploration bonus | 4.0 | How much more likely we are to pick a macro-action than a primitive action when acting randomly |

## C Habit-SAC Hyperparameters

The hyperparameters used for the discrete SAC results are given by Table 10. The network architecture for both the actor and the critic was the same as in the original Deepmind Atari paper Mnih et al. (2015).

Table 10: Hyperparameters used for SAC and Habit-SAC results

| **Hyperparameter** | **Value** |
|---|---|
| Batch size | 64 |
| Replay buffer size | 1,000,000 |
| Discount rate | 0.99 |
| Steps per learning update | 4 |
| Learning iterations per round | 1 |
| Learning rate | 0.0003 |
| Optimizer | Adam |
| Weight Initialiser | He |
| Fixed network update frequency | 8000 |
| Loss | Mean squared error |
| Clip rewards | Clip to [-1, +1] |
| Initial random steps | 20,000 |

The architecture and hyperparameters used for the Habit-SAC results that are relevant to SAC are the same as for SAC and then the rest of the hyperparameters are given by Table 11.

Table 11: Hyperparameters used for Habit-SAC results

| **Hyperparameter** | **Value** | **Description** |
|---|---|---|
| Evaluation episodes | 5 | The number of no exploration episodes we use to infer the habits |
| Replay buffer type | Action balanced replay buffer | The type of replay buffer we use |
| Steps before inferring habits | 30,001 | The number of steps we run before inferring the habits |
| Abandon ship | 2.0 | The threshold for the abandon ship algorithm |
| Grammar algorithm | IGGI | The algorithm we use to infer the habits from the actions |
| Macro-action exploration bonus | 4.0 | How much more likely we are to pick a macro-action than a primitive action when acting randomly |
| Post habit inference random steps | 5,000 | The number of random steps we run immediately after updating the action set with new habits |

## D Things That Did Not Work

Here we list some of the things we tried that did not seem to work but that are worth mentioning:

- We tried adding extra negative rewards to the rewards of a macro-action if Abandon Ship had to be used but this did not seem to help
- When calculating the q-value of a macro-action we tried calculating it by adding the q-value of the first primitive action to the output of the neuron instead of the neuron outputting the q-value directly. This did not seem to help.
- We tried using a constant threshold for abandon ship rather than one that changed over time and found that it did not work very well
- We tried running some extra training iterations after new macro-actions were added but this did not seem necessary
- We tried adding only 1 macro-action at a time but then found this to be unnecessary
- We tried inferring the habits from episodes with exploration on rather than only non-exploration episodes and found that the episodes were too noisy and so it was hard for IGGI to infer any habits
- When adding new habits we tried creating a new neural network and then running many training iterations using the same replay buffer but then we found transfer learning to be a better solution instead
- We tried encouraging the use of macro-actions by making them the preferred action if their q-value was within some distance from the max q-value rather than having to be the max q-value, this did not seem to help
- We tried adding intrinsic rewards to the longer macro-actions to encourage their usage but this did not help

## E Replicating the Results

All the code for this project is on the public github: https://github.com/p-christ/Habit-Reinforcement-Learning. To replicate the results you can follow these steps:

- First clone the repository and install the required packages in requirements.txt. For example follow these steps:
  - git clone https://github.com/p-christ/Habit-Reinforcement-Learning.git
  - cd Habit-Reinforcement-Learning
  - pip3 install requirements.txt
- Then for the Habit-DDQN and DDQN results: follow the instructions in the file https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/experiments/Habit_DDQN_Experiment.py
- For the Habit-SAC and SAC results: read the instructions in the file https://github.com/p-christ/Habit-Reinforcement-Learning/blob/master/experiments/Habit_SAC_Experiment.py

## F Legal and Ethical Considerations

The first ethical consideration of this project is with respect to the **environment**. The scientific evidence that human behaviour is damaging the environment is overwhelming. It is therefore our moral duty to minimise the damage we are causing the environment and this is why one of the Royal Academy of Engineering and the Engineering Council's fundamental ethical principles is "Respect for life, law, the environment and public good". At the same time, however, the type of reinforcement learning models trained in this dissertation requires significant amounts of computation which in turn requires a lot of energy which in turn is often generated in ways that damage the environment.

To deal with this issue we did two things. First of all, we only trained a model when we believed there was a significant chance that doing so would identify some new information and progress the dissertation and the subject positively. Second of all, we used Azure and Google Cloud's resources to train our models which both power their data centres using significant amounts of clean energy (in Google's case it is 100% clean energy).

The second ethical consideration of this project is with respect to **automation and jobs**. We believe that, in the long term, strong advances in reinforcement learning will allow many jobs and tasks to be automated. This will improve efficiency and most likely GDP growth but could lead to a short-term rise in unemployment.

With respect to this issue we believe that we should continue to do research as it is overall beneficial for society but that we should encourage and lobby the government to make sure that they support those who do not benefit from advances in the short-term.

The third ethical consideration of this project is with respect to **military applications**. Even though we believe there are no direct military applications of this work at the moment we believe that Reinforcement Learning as a whole has the potential to be used by the military and so anything that contributes to it in some way could be contributing to military applications. This is a complicated issue and we believe the best solution to it is an international agreement to limit the use of artificial intelligence in warfare, similar to the types of international agreements that exist to limit the use of nuclear weapons.

## G Legal and Ethics Checklist

Table 12: Legal and Ethics Checklist

| **Question** | **Yes or No** |
|---|---|
| Does your project involve Human Embryonic Stem Cells? | No |
| Does your project involve the use of human embryos? | No |
| Does your project involve the use of human foetal tissues / cells? | No |
| Does your project involve human participants? | No |
| Does your project involve human cells or tissues? (Other than from “Human Embryos/Foetuses” i.e. Section 1)? | No |
| Does your project involve personal data collection and/or processing? | No |
| Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)? | No |
| Does it involve processing of genetic information? | No |
| Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc. | No |
| Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets? | No |
| Does your project involve animals? | No |
| Does your project involve developing countries? | No |
| If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned? | No |
| Could the situation in the country put the individuals taking part in the project at risk? | No |
| Does your project involve the use of elements that may cause harm to the environment, animals or plants? | No |
| Does your project deal with endangered fauna and/or flora /protected areas? | No |
| Does your project involve the use of elements that may cause harm to humans, including project staff? | No |
| Does your project involve other harmful materials or equipment, e.g. high-powered laser systems? | No |
| Does your project have the potential for military applications? | No |
| Does your project have an exclusive civilian application focus? | No |

| | |
|---|---|
| Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items? | No |
| Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons? | No |
| Does your project have the potential for malevolent/criminal/terrorist abuse? | No |
| Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery? | No |
| Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied? | No |
| Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project? | No |
| Will your project use or produce software for which there are copyright licensing implications? | No |
| Will your project use or produce goods or information for which there are data protection, or other legal implications? | No |
| Are there any other ethics issues that should be taken into consideration? | No |