Dyna with Options: Incorporating Temporal Abstraction into Planning

by

Gábor Mihucz

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

The motivation to incorporate planning, temporal abstraction and value function approximation in reinforcement learning (RL) algorithms is to reduce the amount of interaction with the environment needed to learn a near-optimal policy. Although each of these concepts has been under intense scrutiny for decades, less is known about their interplay, and specifically: under what circumstances does planning with options provide significant benefits over planning with only primitive actions or model-free alternatives? In this thesis we examine this question by endowing the background planning algorithm, Dyna with access to options with (near)-optimal option policies in two environments: a non-stationary tabular one, in which the changing reward function necessitates rapid value function updates, and in a deterministic, stationary, continuous-state environment that requires value function approximation, a setting in which planning with primitive actions is known to be suboptimal compared to model-free approaches. We find that in the non-stationary environment without a state visitation bonus, all planning algorithms perform significantly better than the model-free Q-learning algorithm; planning with only options (Dyno) performs better than planning with both actions and options (Dyna+options) or planning with actions only (Dyna), the latter two have comparable performance. When a state-visitation bonus is added, each algorithm performs similarly near-optimally, and satisfactory performance can be achieved by restricting the state visitation bonus to goal states. In the value function approximation realm, we find that Dyno outperforms DDQN in terms

of speed and robustness at the beginning of learning, but later on, its performance degrades to that of DDQN's in the instances examined. Dyna+options performs better than Dyna and comparably to DDQN during much of the learning process but with higher variance and occasional dips. We conclude that having access to options with (near-)optimal option policies alone is not sufficient to combat the suboptimality arising from planning with inaccurate primitive models and argue that more sophisticated planning architectures are necessary that bypass the reliance on primitive models.

Preface

The research conducted for this thesis was a team effort led by Professor Martha White with tremendous input and support from Professor Adam White as well. The team also included Chunlok Lo and Farzane Aminmansour. The results in Chapter 4 have been published as Chunlok Lo, Gabor Mihucz, Adam White, Farzane Aminmansour, Martha White, "Goal Space Planning", 2022. The Background section of Chapter 2 is my original work except for the Planning subsection that appears in the aforementioned publication, from which a section features in the Introduction (Chapter 1) in this thesis. Chapter 3 is my original work, with significant help from Martha and Adam White. The experiments in Chapters 4 and 5 and their analyses are my own work, but the codebase for the experiments in Chapter 4 is an adjustment of Dhawal Gupta's codebase used for the publication Dhawal Gupta, Gabor Mihucz, Matthew E. Schlegel, Philip S. Thomas, James E. Kostas, Martha White, "Structural Credit Assignment in Neural Networks using Reinforcement Learning", Neural Information Processing Systems, 2021, while the codebase to run and plot the experiments for Chapter 5 is largely the work of Chunlok Lo.

Success is not final; failure is not fatal: it is the courage to continue that counts.

– Winston Churchill

Acknowledgements

I would like to thank my supervisor, Martha White for giving me this project to work on and for helping me through it over the past year. Without the extent of her support and guidance, this thesis would never have materialized in its current form. I would also like to thank my collaborator, Chunlok Lo, who led a related project: Goal Space Planning, an algorithmic improvement of Dyna with options, the algorithm that is investigated in this thesis. Working on these two projects showed me the current state of empirical reinforcement learning research.

Heartfelt thanks to Adam White, Csaba Szepesvári, and Rich Sutton for our discussions about Artificial Intelligence and research in general, whose influence on me cannot be overstated. Special thanks to Martin Müller for being on my examination committee and providing excellent remarks to finalize this thesis.

Finally, many thanks to all RLAI lab members and the wider Amii community for providing an inspiring and fun environment to learn and create in.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning is a subfield of machine learning concerned with developing agents that learn a policy to behave optimally in an environment by maximizing the expected sum of future rewards via trial and error (R. S. Sutton and A. G. Barto 2018). In relatively simple, unchanging environments correspondingly simple agents, such as Q-learning (Watkins and Dayan 1992) tend to perform very well, with a tractable computational time (Koenig and Simmons 1992). Applications of scientific and economic interest however often require environments with large state and action spaces as well as changing dynamics and rewards (non-stationarity) where such classical methods struggle or are even inapplicable, and hence additional extensions are needed. Some of the proposed extensions that are currently researched with varying degrees of intensity and success are: planning, temporal abstraction and value function approximation.

Planning in reinforcement learning requires access to a model of the environment, which allows the agent to improve its policy without interacting with the environment. Thus, model-based agents should have a significant benefit over model-free ones: their ability to plan with an accurate model should reduce the number of interactions with the environment that is necessary to achieve an optimal policy. This is especially beneficial in environments in which environmental interactions take a longer time/are costlier than interacting with the agent's internal model and/or where the aspects of the environment change faster than the agent can update a corresponding optimal

policy based on direct experience.

Temporal abstraction in reinforcement learning means the notion of actions that span multiple timesteps: most commonly applied in the form of options (R. S. Sutton, Precup, *et al.* 1999). The benefits of having options with a near-optimal policy are similar to having an accurate model: the agent should require fewer interactions with the environment to find the optimal policy and it can update its value function sooner in case the environment dynamics or rewards change.

Value function approximation in reinforcement learning refers to learning a parameterised approximation of (a proxy of) the policy to account for the impracticality or impossibility of calculating it exactly due to a too vast state and/or action space.

Even though these additional components to bare reinforcement learning algorithms have been under intense scrutiny for decades, as of today no empirical analysis has been carried out on their combination, specifically: planning with options, and planning with options with value function approximation, which are the major subjects of this thesis.

The core question we want to answer is: in what circumstances does planning with options with (near-)optimal policies help. To answer this, we examine two settings:

1. a tabular environment with a non-stationary reward function, called GrazingWorld

2. a continuous state, deterministic environment called the PinBall domain, in which value function approximation is necessary

In our experiments, we assume that we have access to options with (near-)optimal option policies in these environments from the start, and do not consider the problem of option discovery - a separate research question beyond the scope of this thesis.

The promise of Dyna is that we can exploit the Markov structure in the RL formalism, to learn and adapt value estimates efficiently, but many open

problems remain to make it more widely useful. These include that (1) one-step models learned in Dyna can be difficult to use for long-horizon planning, and (2) learning probabilities over outcome states can be complex, especially for high-dimensional states.

A variety of strategies have been proposed to improve long-horizon planning. Incorporating options as additional (macro) actions in planning is one approach. Options for planning has largely only been tested in tabular settings (Singh *et al.* 2004; R. S. Sutton, Precup, *et al.* 1999; Wan, Naik, *et al.* 2021). Recent work has considered a mechanism for identifying and learning option policies for planning under function approximation (R. S. Sutton, Machado, *et al.* 2022), but as yet did not consider issues with learning the models.

A variety of other approaches have been developed to handle issues with learning and iterating one-step models. Several papers have shown that using forward model simulations can produce simulated states that result in catastrophically misleading values (Jafferjee *et al.* 2020; Lambert *et al.* 2022; van Hasselt *et al.* 2019). This problem has been tackled by using reverse models (Jafferjee *et al.* 2020; Pan, Zaheer, *et al.* 2018; van Hasselt *et al.* 2019); primarily using the model for decision-time planning (Chelu *et al.* 2020; Silver, R. S. Sutton, *et al.* 2008; van Hasselt *et al.* 2019); and improving training strategies to account for accumulated errors in rollouts (Talvitie 2014; Talvitie 2017; Venkatraman *et al.* 2015). An emerging trend is to avoid approximating the true transition dynamics, and instead learn dynamics tailored to predicting values on the next step correctly (Ayoub *et al.* 2020; Farahmand 2018; Farahmand *et al.* 2017). This trend is also implicit in the variety of techniques that encode the planning procedure into neural network architectures that can then be trained end-to-end (Farquhar *et al.* 2018; Oh, Singh, *et al.* 2017; Schrittwieser *et al.* 2020; Silver, H. Hasselt, *et al.* 2017; Tamar, Wu, *et al.* 2016; Weber *et al.* 2017).

The thesis is organised as follows: Chapter 2 provides the background necessary to understand the rest of the thesis. Chapter 3 introduces Dyna with Options in the tabular case and with value function approximation. Chapters 4 and 5 list the experimental setup and results for the GrazingWorld and PinBall

environments, respectively. Chapter 6 draws the conclusions and discusses potential future work worthy of investigation.

In the non-stationary, tabular environment we find that Q-learning with options with (near-)optimal option policies and especially planning with an adequate state visitation bonus achieves significantly better performance than model-free Q-learning with primitive actions only. Planning with options however, does not provide significant additional benefits over planning with primitive actions only.

In the PinBall Domain we find that access to options with (near-)optimal option policies speeds up the learning progress in the beginning compared to DDQN - with the performance slightly degrading to DDQN's performance in the instances examined. Dyna and Dyna with Options perform similarly in this setting too: both learn faster than DDQN in the beginning and both plateau at a policy that achieves significantly less rewards than their primitive model-free counterparts. Dyna+options in the function approximation regime performs significantly better than Dyna during much of the training regime and comparably to DDQN, albeit with an extended dip in the middle of the learning during which its performance is almost as suboptimal as Dyna's.

1.1 Results and Contributions

To summarize, the key contributions of this thesis are as follows: we

1. propose two novel algorithms to appropriately investigate incorporating temporal abstraction into Dyna: Dyno and Dyna+Options. We additionally provide pseudocode for Dyna with options under value function approximation, previously only specified in the tabular setting.

2. empirically show that Dyna+options provides some benefit over Dyna, but planning with the inaccurate primitive action model still significantly corrupts the value function

3. empirically demonstrate in the investigated environments that Dyno equipped with options with (near-)optimal option policies is both more

effective and robust than Dyna+options due to the idiosyncrasies of planning with both actions and options with a uniformly sampled model

4. we show that in the non-stationary tabular environment all three algorithms can be tuned to perform similarly with a uniform state visitation bonus and near-optimal performance can be achieved when the state visitation bonus is only added to the goal states

Chapter 2

Background

This chapter enumerates the topics that are necessary to understand the rest of the thesis and introduces the notation.

2.1 Reinforcement Learning

This thesis considers the standard reinforcement learning setting, where an *agent* learns to make decisions through interacting with an *environment* without knowledge of the environment's dynamics (R. S. Sutton and A. G. Barto 2018).

2.2 MDPs

Reinforcement Learning problems are typically formulated as Markov Decision Processes (MDP) which are formally defined as a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, in the discounted case, where $\mathcal{S}$ denotes the state and $\mathcal{A}$ the action space. $\mathcal{R}$ and the transition probability $\mathcal{P}$ describe the expected reward and probability of transitioning to a state, for a given state and action. In this thesis, we examine discounted MDPs with discount factor $\gamma \in [0, 1)$ with finite state and action spaces and bounded rewards. We further assume that the agent makes decisions at discrete time steps $t \in \mathbb{N}^+$.

2.3 Episodes, Returns, and Policies

Our experiments consider the episodic RL setting, in which the agent starts each *episode* in the same starting state, on each discrete *timestep* t the agent selects an action A_t in state S_t, the environment transitions to a new state S_{t+1} with probability $P(S_{t+1}|S_t, A_t)$ and emits a scalar reward R_{t+1}, until it reaches the terminal state, where the episode ends, the agent is reset to starting state and a new episode begins for the specified number of episodes.

The agent's objective is to find a *policy* $\pi : S \times A \to [0, 1]$ that maximizes the expected *return*, the future discounted reward $G_t \doteq R_{t+1} + \gamma_{t+1} G_{t+1}$. The state-based discount $\gamma_{t+1} \in [0, 1]$ depends on S_{t+1} (R. S. Sutton, Modayil, *et al.* 2011), which allows us to specify termination. If S_{t+1} is a terminal state, then $\gamma_{t+1} = 0$; else, $\gamma_{t+1} = \gamma_c$ for some constant $\gamma_c \in [0, 1]$.

2.4 Action-Value Functions and Optimal Policies

Action-value functions under policy π map taking action a in state s and then following policy π to the expected return:

$$q^\pi(s, a) \doteq \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} G_t | S_t = s, A_t = a \right], \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

The action-value of the terminal state is always 0.
Action-value functions define a partial ordering over policies (R. S. Sutton and A. G. Barto 2018).

$$\pi \geq \pi' \iff q^\pi(s, a) \geq q^{\pi'}(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

There is a unique action-value function, the optimal action-value function,

$$q^*(s, a) \doteq \max_\pi q_\pi(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

and there is at least one optimal policy π^* associated with this unique action-value function:

$$\pi^*(s) \doteq \operatorname*{argmax}_a q^*(s, a), \forall s \in \mathcal{S}$$

2.5 Q-learning

Action-value functions can be estimated with Dynamic Programming (DP) (Bellman 1957) methods, however, they require knowledge of the environment dynamics $\mathcal{P}$. Temporal-Difference (TD) learning algorithms (R. S. Sutton 1988) remove this strong assumption and estimate value functions with the rest of the components of the MDP. A popular TD-learning algorithm is Q-learning (Watkins and Dayan 1992), which after taking action A_t in state S_t and observing S_{t+1} and R_{t+1} performs the following update:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

where α is the step-size hyperparameter that needs to be tuned for faster learning, and $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ is called the target.

2.6 The Exploration-Exploitation Dilemma

Reinforcement learning requires a balance between exploration and exploitation (R. S. Sutton and A. G. Barto 2018): always executing the greedy action (the one with the greatest action-value) may lead to a suboptimal policy, while always taking random actions is seldom optimal either. Finding the right balance of exploration for optimal learning is an open research question that is beyond the scope of this thesis. A simple and commonly used exploration strategy that we will use in this thesis is called ϵ-greedy: the agent executes a random action with probability ϵ and acts greedily with probability 1-ϵ, usually with $\epsilon \in (0, 0.1]$.

2.7 Planning

We can incorporate models and planning to improve sample efficiency beyond model-free algorithms. In this thesis, we focus on background planning algorithms: those that learn a model during online interaction and asynchronously update value estimates. Another class of planning algorithms is called *model predictive control* (MPC). These algorithms learn a model and use decision-time planning by simulating many rollouts from the current state. Other recent

algorithms using this idea are those doing Monte Carlo tree search (MCTS) online, such as MuZero (Schrittwieser *et al.* 2020).

2.7.1 Dyna

We choose to focus on background planning with Dyna because Dyna (R. Sutton 1990) is a classic example of background planning. On each step, the agent simulates several transitions according to its model and updates with those transitions as if they were real experience. In this thesis, we examine the DynaQ algorithm, which performs Q-learning updates on both real experience and experience simulated from a learned model of the environment.

DynaQ+

DynaQ+ (R. S. Sutton and A. G. Barto 2018) is the DynaQ algorithm with the extension of a state visitation bonus: the algorithm initializes a table $\tau(s, a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ and increments its values by 1 at each step, setting the value of the current state-action pair to 0. In the planning step, the Q-learning update becomes:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[\hat{R}_{t+1} + \kappa \sqrt{\tau(S_t, A_t)} + \gamma \max_a Q(\hat{S}_{t+1}, a) - Q(S_t, A_t) \right]$$

for some small κ that needs to be tuned for optimal performance.

The state-visitation bonus is a form of a somewhat more directed exploration than simple ϵ-greedy: it encourages the agent to explore state-action pairs that it has not tried for a long time since their value will increase during planning when they are sampled from the model.

We provide the pseudocode for DynaQ+ in Algorithm 1 below.

Algorithm 1 DynaQ+ with a Tabular Value Function for Episodic Problems

Initialize $Q(\mathcal{S}, \mathcal{A}), \tau(\mathcal{S}, \mathcal{A}), \kappa, n, \epsilon$,
Sample initial state s_0 from the environment
for $t \in 0, 1, 2, ...$ **do**
  $\xi \sim$unif$(0, 1)$
  if $\xi < \epsilon$ **then**
    $a_t \sim \text{unif}_{discrete}(\mathcal{A})$
  else
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s, a)$
  end if
  $\tau(\cdot, \cdot) = \tau(\cdot, \cdot) + 1$
  Take action a_t, observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
  `DirectExperienceValueUpdate`$(s_t, a_t, s_{t+1}, r_{t+1})$
  `ModelUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$
  for n iterations **do**
    // Sample state and action from the primitive model
    $s, a \sim$ PrimitiveModel()
    `PrimitiveModelValueUpdateKappaUniform`(s, a)
  end for
  $\tau(s, a) = 0$
end for

Algorithm 2 `DirectExperienceValueUpdate`(s, a, s', r)

$\delta \leftarrow r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta$

Algorithm 3 `PrimitiveModelValueUpdateKappaUniform`(s, a)

// Generate experience from the primitive model
$\hat{s'}, \hat{r}, \hat{\gamma} \leftarrow$ PrimitiveModel(s, a)
// Add state visitation bonus to current reward
$\hat{r} \leftarrow \hat{r} + \kappa\sqrt{\tau(s, a)}$
// Update the primitive action-values
$\delta_a \leftarrow \hat{r} + \hat{\gamma} \max_{a' \in \mathcal{A}} Q(\hat{s'}, a') - Q(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_a$

Algorithm 4 `ModelUpdate`(s, a, s', r, γ)

// Update PrimitiveModel
PrimitiveModel$(s, a) \leftarrow s', r, \gamma$

2.8 Temporal Abstraction

Options (R. S. Sutton, Precup, *et al.* 1999) provide temporally-extended ways of behaving, allowing the agent to reason about outcomes further into the future. A Markov option is a 3-tuple $o = \{\mathcal{I}_o, \beta_o, \pi_o\}$, where $\mathcal{I}_o \subset \mathcal{S}$ is the initiation function of an option, which maps states to the probability of the option being started there, $\beta_o : \mathcal{S} \to [0, 1]$ is a function that maps states to the probability of the option's termination and π_o: $\mathcal{S} \to \Delta_{\mathcal{A}}$ is the option's policy that maps states to a distribution over the actions. Based on this definition, it can be inferred that each (primitive) action a can be interpreted as an option with an arbitrary initial state and the termination probability of $\beta_a(s) = 1, \forall s \in \mathcal{S}$, following an arbitrary policy π_a.

In our experiments we will consider the setup in which agents have access to all primitive actions $a \in \mathcal{A}(s), \forall s \in \mathcal{S}$ as well as a set of options with (near-)optimal option policies $o \in \mathcal{O}(s), \forall s \in \mathcal{S}$. While interacting with the environment, if the agent selects an option o_t in state s_t, then $a_t = \pi_{o_t}(s_t)$ is executed and used for the direct experience update. Furthermore, agents only perform primitive exploratory actions. The option-values are learned with the standard Q-learning update but with simulated $\hat{s'}_o, \hat{r}_o$ and $\hat{\gamma}_o$ retrieved from the learned *option model* of the sampled option o. In this thesis, the union of primitive actions and options will be denoted $u \in \mathcal{U} = \mathcal{A} \cup \mathcal{O}$.

There are two main unanswered research questions regarding options: how the agent should discover them and how it should use them to learn more efficiently and effectively. In this thesis, we assume that the agent has already executed a successful option discovery phase and direct our attention to the latter question.

2.9 Value Function Approximation

Until this point, we discussed algorithms with the implicit assumption that their action-value can be stored in a table of size $\mathcal{S} \times \mathcal{A}$. This assumption is untenable in many applications from both a computational space and time

complexity point of view.

One solution to resolve this issue is to approximate the action-value function with function approximation techniques. This can be done with e.g. a parameterised weight vector $\theta \in \mathbb{R}^d : q_\theta(s, a) \approx q^*(s, a)$, which is updated during learning - instead of the Q-table itself - to reduce the approximation error. The parameter update in semi-gradient Q-learning for example, a variant of Q-learning for function approximation with a given transition, $(S_t, A_t, S_{t+1}, R_{t+1})$ is:

$$\theta_{t+1} = \theta_t + \left[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a; \theta_\mathbf{t}) - \hat{q}(S_t, A_t; \theta_\mathbf{t}) \right] \nabla_{\theta_\mathbf{t}} \hat{q}(S_t, A_t)$$

where $\nabla_{\theta_\mathbf{t}} \hat{q}(S_t, A_t)$ is the gradient of the approximate action-value function $\hat{q}(S_t, A_t)$ with respect to θ at timestep t.

A well-understood type of function approximation involves linear functions, where the approximate action-value function is a linear function of the weight vector:

$$\hat{q}(s, a; \theta) \doteq \theta^\top \mathbf{x}(s, a)$$

where $\mathbf{x}(s, a)$ is the feature vector representing the state-action pair s, a and is also the gradient of the approximate action-value function. The performance of algorithms with linear functions depends largely on the feature vectors, which must be specified in advance.

Another alternative is to use non-linear function approximators, such as deep artificial neural networks, which innately find features with the back-propagation algorithm (Rumelhart *et al.* 1986). Disadvantages of neural networks are that they are poorly understood and that they are difficult to optimize, especially on non-stationary, temporally correlated data inherent to Reinforcement Learning. Nevertheless, with a number of techniques that have been developed recently and extensive hyperparameter tuning, deep artificial neural networks enabled RL algorithms to solve problems for which no other approach succeeded.

2.9.1 Deep Q-Networks

Deep Q-Network (DQN) (Mnih *et al.* 2015) was the first widely successful RL algorithm applied to games in the Arcade Learning Environment (ALE) (Bellemare *et al.* 2012). DQN is the moniker for a specific way to approximate action-values using deep neural networks, featuring:

1. Convolutional Neural Networks (CNNs) (LeCun *et al.* 1989) to deal with the large state space of ATARI games,

2. an "experience replay buffer" (L. J. Lin 1993) to store transitions, from which a fixed-length i.i.d. batch is sampled and used to update the parameters at each time-step, helping with temporally decorrelating the data,

3. a "target network" parameterised by the weight vector $\theta^- \in \mathbb{R}^d$, whose values are updated every C steps with the main Q-network's parameters θ, while between updates they are kept fixed, which ensures stationarity.

Double Deep Q-Networks

Both Q-learning and DQN select and evaluate the action using the max operator on the same values, which makes it more likely to result in overestimated values, a source of suboptimality (H. v. Hasselt *et al.* 2016). Double Q-learning (H. Hasselt 2010) avoids this by learning two value functions with two sets of weights θ and θ'. In each update, one set of weights is used for action selection and the other for estimating its value, so Q-learning's target:

$$Y_t^Q \doteq R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta_t); \theta_t)$$

becomes

$$Y_t^{DoubleQ} \doteq R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta_t); \theta'_t)$$

as such, decoupling action selection and evaluation. By alternating the roles of θ and θ' during learning, both sets of weights can be updated.

In the realm of value function approximation, the corresponding target of DQN:

$$Y_t^{DQN} \doteq R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t^-)$$

in Double Deep Q-Networks (DDQN) becomes:

$$Y_t^{DDQN} \doteq R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \theta_t), \theta_t^-)$$

i.e. action selection is done by the main network and evaluated by the target network. The target network is updated periodically, the same way as DQN.

As DDQN empirically dominates DQN, we will use it as the baseline to compare to its extensions: DynaDDQN and DynoDDQN. A slight change to the original algorithm involves updating the target network towards the main network using Polyak averaging (Polyak and Juditsky 1992):

$$\theta_{t+1}^- = \alpha_{Polyak}\theta_t^- + (1 - \alpha_{Polyak})\theta_t$$

i.e. an exponential moving average update between the main and target networks. As such α_{Polyak} is the hyperparameter that needs to be tuned instead of C, the number of steps until the target network is updated with the main network's parameters.

2.9.2 Dyna with Function Approximation

The tabular DynaQ algorithm performs planning updates with exact samples s, a, s', r, hence Experience Replay is essentially a Dyna-update with a perfect model. However, a learned parametric model could simulate novel experiences - never visited states and never taken actions and their corresponding rewards and next states - which could further improve the approximate action-value function. Therefore, in our implementation of DynaDDQN and DynoDDQN, the planning step will use simulated $(\hat{r}, \hat{s}', \hat{\gamma}(s))$ values retrieved from a learned model of the environment. We provide a detailed overview of this algorithm in the chapter 4

2.10 Model Learning

There are three types of environment models: given a current state and action, *distribution models* output a distribution over next states and rewards, *sample models* produce a sample of a next state and reward, while *expectation models* generate the expected next state and reward. Expectation models are unsuitable for stochastic environments as they produce invalid/non-existing states, however, in this thesis, we will consider deterministic environments with fixed transition probability p, for which they are adequate.

2.10.1 Primitive Models

Tabular Models

In the simpler, tabular setup, the model learning involves simply exploring the environment with random actions and storing the state-action pairs and the corresponding (next state, reward, discount factor)-triples in a look-up table.

Approximate Models

Environments with large state and/or action spaces require learning an approximate model. This can be done with supervised learning: as in tabular models, the agent explores the environment with randomized actions, but instead of storing the experienced environment data, it uses them to learn to be able to predict the next states, rewards and state-based discount factors accurately. In this thesis, for learning an approximate environment model, we will use standard deep neural networks techniques. In particular, given the input dataset $\mathcal{D}(s, a, s', r, \gamma)$ we will learn the parameters θ of a function f, that maps (s, a) to $(\hat{s}', \hat{r}, \hat{\gamma})$

$$f^{\theta}((s, a)) \rightarrow (\hat{s}', \hat{r}, \hat{\gamma}),$$

such that the loss function

$$\left[(s', r, \gamma) - (\hat{s}', \hat{r}, \hat{\gamma})\right]^2$$

is minimized. In our experiments, the parameters are the weights and biases of a deep neural network model, and the loss is minimized by updating these

parameters on each iteration with the backpropagation algorithm (Rumelhart *et al.* 1986). As this is a regression problem, the loss function used is the mean squared error (MSE).

2.10.2 Option Model

Tabular Option Model

Learning the option models requires learning $s'(s,o), r(s,o)$ and $\gamma(s,o)$, $\forall s \in \mathcal{S}, o \in \mathcal{O}|I_o(s) > 0$. The latter two quantities will be learned directly as described in (R. S. Sutton, Precup, *et al.* 1999):

$$r(s,o) = r(s,o) + \alpha(r + \gamma(1-\beta(s,o))r(s',o) - r(s,o))$$

$$\gamma(s,o) = \gamma(s,o) + \alpha(\beta(s,o) + \gamma(1-\beta(s,o))\gamma(s',o) - \gamma(s,o))$$

where α is the stepsize hyperparameter that must be tuned for optimal performance. Since in the tabular case the state space is discrete, instead of the expected next state we learn the transition probability:

$$p(s,o) = p(s,o) + \alpha((\beta(s,o) * \mathbb{1}_{|\mathcal{S}|}(s')) + (1-\beta(s,o)) * p(s',o) - p(s,o))$$

and sample the next state according to this $p(s,o)$.

2.10.3 Approximate Option Model

Similarly to the primitive action model, we can learn to predict $\hat{r}(s,o)$ and $\hat{\gamma}(s,o)$, and in deterministic environments we can reliably learn $\hat{s}'(s,o)$ as well. On each timestep, for each option o that is initialized in the current state s_t, we sample a batch of transition data (s,a,r,s',γ) from a buffer B_o and update the option policy and model (the reward and discount factor) with a DDQN-like update as described in Algorithm 5:

Algorithm 5 Option Policy and Model Update

Require: batch of transitions (s, a, r, s', γ) sampled from a buffer B_o for option $o(I_o, \beta_o, \pi_o)$, parameters $\theta^\pi, \theta^r, \theta^\Gamma$
$\gamma_o \leftarrow \gamma(1 - \beta_o(s'))$
// Update option policy
$\delta^\pi \leftarrow \frac{1}{2}(r-1) + \gamma_o \max_{a' \in \mathcal{A}} \tilde{q}(s', a', o; \theta^\pi) - q(s, a, o; \theta^\pi)$
$\theta^\pi \leftarrow \theta^\pi + \alpha^\pi \delta^\pi \nabla q(s, a, o; \theta^\pi)$
// Update reward model and discount model
$a' \leftarrow \text{argmax}_a q(s, o_i; \theta^\pi)$
$\delta^r \leftarrow r + \gamma_o r_\gamma(s', a', o_i; \theta^r) - r_\gamma(s, a, o_i; \theta^r)$
$\delta^\Gamma \leftarrow 1(\gamma_o = 0)\gamma + \gamma_o \Gamma(s', a', o_i; \theta^\Gamma) - \Gamma(s, a, o_i; \theta^\Gamma)$
$\theta^r \leftarrow \theta^r + \alpha^r \delta^r \nabla r_\gamma(s, a, o_i; \theta^r)$
$\theta^\Gamma \leftarrow \theta^\Gamma + \alpha^\Gamma \delta^\Gamma \nabla \Gamma(s, a, o_i; \theta^\Gamma)$

Finally, the option's expected next state can be calculated with a TD-like update for the states in which the option terminates:

if $\beta_o(s') == 1$ **then**
  $\hat{s'}_{s,o} \leftarrow \hat{s'}_{s,o} + \alpha(s' - \hat{s'}_{s,o})$
end if

We can then retrieve $\hat{s'}_{s,o}$ directly, while the $\hat{r}_{s,o}$ and $\hat{\gamma}_{s,o}$ can be calculated as follows:

$\pi_{s,\cdot,o}, r_{s,\cdot,o}, \gamma_{s,\cdot,o} \leftarrow \text{OptionModel}(s; \theta_o)$

$a' \leftarrow \text{argmax}_a \, \pi_{s,\cdot,o}$

$r_{s,o} \leftarrow r_{s,a',o}$

$\gamma_{s,o} \leftarrow \gamma_{s,a',o}$

Chapter 3

Dyna with Options

In this chapter, we present and discuss combining Dyna with options with a tabular value function and with value function approximation. We discuss the major components of the algorithms and we provide their pseudocodes.

3.1 Dyna with Options with a Tabular Value Function

3.1.1 Action Selection, Execution and Direct Experience Update

In each state, the agent takes a random primitive action with probability ϵ, and with probability $1 - \epsilon$ it selects the action or option that is initialized in the current state with the highest action-value. If an option was selected, the action according to the option's policy is executed. The agent then transitions to the next state and receives the corresponding reward and a standard Q-learning update is carried out as in Algorithm 2.

3.1.2 Primitive Action and Option Model Learning

In order to be able to plan, the agent has to create a model of the environment. Thus, at each timestep the agent stores the current state-action pairs and their corresponding next state, reward and state-based discount factor in a look-up table that will serve as the primitive action model of the environment (PrimitiveModel in Algorithm 4). The option models are learned by iterating over each available option in the current state whose policy takes the selected

action in the current state, and storing the reward, state-based discount factor and transition probability models in a lookup-table for each state and option. The reward and discount factor models are the sums of the discounted corresponding quantities starting from the current state until option termination learned via a temporal difference learning update. The transition probabilities are modeled instead of the next states so that by sampling according to this probability we ensure that the option models' next states are states existing in the Q-table.

At each timestep, for a prespecified number of planning steps, the agent updates its value function with values retrieved from their models. As written in Algorithm 8, the agent retrieves the stored reward, next state and state-based discount factor for the sampled state-action pair and performs a standard Q-learning update with the values.

Similarly, Algorithm 11 describes the corresponding Q-learning option-value update with the $\hat{s'}_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o}$ values retrieved from the option model of a random available option o in a sampled state s.

3.1.3 Discussion of Components and Naming

The algorithm discussed thus far has no moniker in the literature, and in the remaining of the thesis, we will refer to it as **Dyna+options**. We provide its pseudocode in Algorithm 13. Leaving out certain components from Dyna+options results in a lesser-known and two well-known algorithms. We will refer to the algorithm that only plans with options as **Dyno**, and provide its pseudocode in Algorithm 10. If we do not have access to options and plan with only primitive actions, we get Dyna (Algorithm 6). Finally, if we do not have access to options, and perform no planning, we get Q-learning. In the rest of the thesis, we will focus on investigating these algorithms and their variants adapted to environments with challenges including non-stationarity, and a state space that necessitates value function approximation.

3.1.4 Pseudocode for Dyna, Dyno and Dyna+options with a Tabular Value Function

In this section, we provide the pseudocode for Dyna+options, Dyno and Dyna, each equipped with a tabular value function.

Algorithm 6 Dyna with a Tabular Value Function for Episodic Problems

Initialize $Q(\mathcal{S}, \mathcal{A} \cup \mathcal{O}), n, \epsilon,$
Sample initial state s_0 from the environment
for $t \in 0, 1, 2, ...$ **do**
  $\xi \sim$unif$(0, 1)$
  if $\xi < \epsilon$ **then**
    $a_t \sim \text{unif}_{discrete}(\mathcal{A})$
  else
    $a_t \leftarrow \text{argmax}_{a \in \mathcal{A}} Q(s, a)$
  end if
  Take action a_t, observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
  `DirectExperienceValueUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \mathcal{U})$
  `PrimitiveModelUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$
  for n iterations **do**
    // Sample state and action from the primitive model
    $s, a \sim$ PrimitiveModel()
    `PrimitiveModelValueUpdate`$(s, a, \mathcal{U})$
  end for
end for

Algorithm 7 `DirectExperienceValueUpdate`$(s, a, s', r, \mathcal{U})$

$\delta \leftarrow r + \gamma \max_{u' \in \mathcal{U}} Q(s', u') - Q(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta$

Algorithm 8 `PrimitiveModelValueUpdate`$(s, a, \mathcal{U})$

// Generate experience from the primitive model
$\hat{s'}, \hat{r}, \hat{\gamma} \leftarrow$ PrimitiveModel(s, a)
// Update the primitive action-values
$\delta_a \leftarrow \hat{r} + \hat{\gamma} \max_{u' \in \mathcal{U}} Q(\hat{s'}, u') - Q(s, u)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_a$

Algorithm 9 `PrimitiveModelUpdate`(s, a, s', r, γ)

// Update PrimitiveModel
PrimitiveModel$(s, a) \leftarrow s', r, \gamma$

Algorithm 10 Dyno with a Tabular Value Function for Episodic Problems

Assume given options $\mathcal{O}(I, \beta, \pi)$
Initialize $Q(\mathcal{S},\mathcal{U}), n, \epsilon,$
Sample initial state s_0 from the environment
for $t \in 0, 1, 2, ...$ **do**
  $\xi \sim$unif$(0, 1)$
  if $\xi < \epsilon$ **then**
    $a_t \sim \text{unif}_{discrete}(\mathcal{A})$
  else
    $u_t \leftarrow \text{argmax}_{u \in \mathcal{U}_s} Q(s, u)$ // $\mathcal{U}_s := \{\mathcal{A} \cup \mathcal{O} | I_{o \in \mathcal{O}}(s) > 0\}$
    if $u_t \in \mathcal{O}$ **then**
      $a_t \leftarrow \pi_{u_t}(s)$
    else
      $a_t \leftarrow u_t$
    end if
  end if
  Take action a_t, observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
  `DirectExperienceValueUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \mathcal{U})$
  `OptionModelUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$
  for n iterations **do**
    // Sample state from the environment model, a model
    // that only keeps track of the states
    $s \sim$ EnvironmentModel()
    `OptionModelValueUpdate`(s)
  end for
end for

Algorithm 11 `OptionModelValueUpdate`(s)

// Generate experience from the option model
$o \sim unif_{discrete}(\mathcal{O}_s)$
$\hat{s'}_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o} \leftarrow$ OptionModel(s, o)
// Update the option-values
$\delta_o \leftarrow \hat{r}_{s,o} + \hat{\gamma}_{s,o} \max_{u' \in \mathcal{U}} Q(\hat{s'}_{s,u}, u') - Q(s, o)$
$Q(s, o) \leftarrow Q(s, o) + \alpha\delta_o$

Algorithm 12 `OptionModelUpdate`(s, a, s', r, γ)

// Update OptionModel for each option whose policy takes action a in current state s
for $o \in \mathcal{O}$ **do**
  if $\pi_o == a$ **then**
    $r_{s,o} \leftarrow r_{s,o} + \alpha(r + \gamma(1-\beta_o)r_{s',o} - r_{s,o})$
    $\gamma_{s,o} \leftarrow \gamma_{s,o} + \alpha(\beta_o + \gamma(1-\beta_o)\gamma_{s',o} - \gamma_{s,o})$
    $p(s'|s,o) \leftarrow p(s'|s,o) + \alpha((\beta_o \mathbb{1}(s')) + (1-\beta_o)p(s''|s',o) - p(s'|s,o))$
  end if
end for
// Update EnvironmentModel
EnvironmentModel() $\leftarrow s$

Algorithm 13 Dyna+options with a Tabular Value Function for Episodic Problems

Assume given options $\mathcal{O}(I, \beta, \pi)$
Initialize $Q(\mathcal{S},\mathcal{U}), n, \epsilon,$
Sample initial state s_0 from the environment
for $t \in 0, 1, 2, ...$ **do**
  $\xi \sim$unif$(0, 1)$
  if $\xi < \epsilon$ **then**
    $a_t \sim \text{unif}_{discrete}(\mathcal{A})$
  else
    $u_t \leftarrow \text{argmax}_{u \in \mathcal{U}_s} Q(s, u)$ // $\mathcal{U}_s := \{\mathcal{A} \cup \mathcal{O} | I_{o \in \mathcal{O}}(s) > 0\}$
    if $u_t \in \mathcal{O}$ **then**
      $a_t \leftarrow \pi_{u_t}(s)$
    else
      $a_t \leftarrow u_t$
    end if
  end if
  Take action a_t, observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
  `DirectExperienceValueUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \mathcal{U})$
  `PrimitiveModelUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1}, \mathcal{U})$
  `OptionModelUpdate`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$
  for n iterations **do**
    // Sample state and action from the primitive model
    $s, a \sim$ PrimitiveModel()
    `PrimitiveModelValueUpdate`(s, a)
    `OptionModelValueUpdate`(s)
  end for
end for

3.2 Dyna+Options with Value Function Approximation

In this section, we discuss the learning components of Dyna with options with value function approximation and provide the corresponding pseudocode.

3.2.1 Action Selection, Execution, Direct Experience Storage

The action selection and execution are equivalent to the algorithm with a tabular value function, but instead of performing an update with the experienced values at each timestep, we store the direct and simulated experiences in buffers that will be used to update the parameters of the approximate primitive action and option models and action- and option-value functions to improve the agent's performance on subsequent iterations (Algorithm 15):

- s, a and the predictions of the Primitive Model with this input $\hat{s'}, \hat{r}, \hat{\gamma}$ are stored in $B_{model\_primitive}$
- for each option o initialized in the current state
  - add the experience (s, a, s', r, γ) to a buffer dedicated to that option (B_{direct_o})
  - generate the option's policy $\hat{\pi}_{s,\cdot,o}$, as well as rewards, discount factors and next states from its model: $\hat{r}(s, \cdot, o), \hat{\gamma}(s, \cdot, o)$, $\hat{s}'_{s,o}$
  - pick $a'_{s,o}$ by choosing the action with the highest probability in $\hat{\pi}_{s,\cdot,o}$
  - add $\hat{s}'_{s,o}, \hat{r}_{s,a',,o}, \hat{\gamma}_{s,a',o}$ to a dedicated buffer of that option (B_{model_o})

3.2.2 Model Learning

Option Models

The option model learning described in Algorithm 19 is carried out as follows: at each timestep, the agent stores its experience in a buffer for each option ($B_{direct\_o_i}$) that is initialized in the current state s. The options' policies, discount factors and expected discounted rewards are then calculated by sampling

from the buffer and performing a DDQN-update on the option model's discount rate and reward. The expected next state of each option is calculated with a TD-like update.

Algorithm 19 includes the steps to learn the primitive model too: at each timestep, a batch of $s, a, \hat{s}', \hat{r}, \hat{\gamma}$ is sampled from $B_{model\_primitive}$, where $\hat{s}', \hat{r}, \hat{\gamma}$ are the outputs of a supervised learning algorithm trained to minimize the loss between the predicted and actual values given the current state and action.

Action- and Option-Value Function Updates

At each timestep, we sample a batch of

1. (s, a, s', r, γ) from $B_{direct\_primitive}$ (Algorithm 16) ,
2. $(s, a, \hat{s}', \hat{r}, \hat{\gamma})$ from $B_{model\_primitive}$ (Algorithm 17)
3. $(s, o, \hat{s'}_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o})$ from the buffer B_{model_o} (Algorithm 18)

and perform a standard DDQN-update.

3.2.3 Naming of Algorithms

Analogously to Algorithm 10, Algorithm 14 without Algorithms 17 and 18 corresponds to DDQN, without Algorithm 18 we get **Dyna** with value function approximation, while without Algorithm 16, we get **Dyno** with value function approximation. Algorithm 14 as a whole constitutes **Dyna+options** with value function approximation.

3.2.4 Pseudocode for Dyna+options with Value Function Approximation

In this section we provide the pseudocode for Dyna+options with value function approximation.

Algorithm 14 Dyna+Options with value function approximation

Assume given options $\mathcal{O}$
Initialize model parameters $\theta = (\theta^r, \theta^\Gamma, \theta^\pi), \theta_{pm} = (\theta^r_{pm}, \theta^{s'}_{pm}, \theta^\gamma_{pm}), \theta_{pn}$, θ_{model_on}
Sample initial state s_0 from the environment
for $t \in 0, 1, 2, ...$ **do**
  $\xi \sim$unif$(0, 1)$
  if $\xi < \epsilon$ **then**
    $a_t \sim \text{unif}_{discrete}(\mathcal{A})$
  else
    $u_t \leftarrow \text{argmax}_u q(s, \mathcal{U}_I)$ // $\mathcal{U}_I := \{\mathcal{A} \cup \mathcal{O} | I_{o \in \mathcal{O}}(s) > 0\}$
    if $u_t \in \mathcal{O}$ **then**
      $a_t \leftarrow \text{argmax}_a q(s, u_t; \theta^\pi)$
    else
      $a_t \leftarrow u_t$
    end if
  end if
  Take action a_t, observe $s_{t+1}, r_{t+1}, \gamma_{t+1}$
  `AddToBuffer`$(s_t, a_t, s_{t+1}, r_{t+1}, \gamma_{t+1})$
  `ModelUpdate`()
  `DirectExperienceParameterUpdate`()
  `PrimitiveModelParameterUpdate`()
  `OptionModelParameterUpdate`()
end for

Algorithm 15 `AddToBuffer`(s, a, s', r, γ)

Add new transition (s, a, s', r, γ) to buffer $B_{direct\_primitive}$
$\hat{s}', \hat{r}, \hat{\gamma} \leftarrow \text{PrimitiveModel}(s, a; \theta_{pm})$
Add new transition/data $(s, a, \hat{s}', \hat{r}, \hat{\gamma})$ to buffer $B_{model\_primitive}$
for $o_i \in \mathcal{O}$ **do**
  // Store sample for o_i if the experience doesn't lead it away
  if $I_{o_i}(s)$ and $(\beta_{o_i}(s)$ or $I_{o_i}(s'))$ **then**
    Add new transition (s, a, s', r, γ) to buffer $B_{direct\_o_i}$
  else
    Add new transition $(s, a, s', 0, 0)$ to buffer $B_{direct\_o_i}$
  end if
  if $I_{o_i}(s)$ **then**
    $\hat{\pi}_{s,a,o_i}, \hat{r}_{s,a,o_i}, \hat{\gamma}_{s,a,o_i}, \hat{s'}_{s,o} \leftarrow \text{OptionModel}(s; \theta_i)$
    $a' \leftarrow \text{argmax}_{a \in \mathcal{A}} \hat{\pi}_{s,a,o_i}$
    $\hat{r}_{s,o_i} \leftarrow \hat{r}_{s,a',o_i}$
    $\hat{\gamma}_{s,o_i} \leftarrow \hat{\gamma}_{s,a',o_i}$
    Add new transition/data $(s, o, \hat{s}'_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o})$ to buffer $B_{model\_o_i}$
  end if
end for

Algorithm 16 `DirectExperienceValueUpdate`()

for n iterations, for multiple transitions (s, a, r, s', γ) sampled from $B_{direct\_primitive}$ **do**
  $\delta \leftarrow r + \gamma \max_{u' \in \mathcal{U}_I} q(s', u'; \theta_{pn}) - q(s, a; \theta_{pn})$
  $\theta_{pn} \leftarrow \theta_{pn} + \alpha \delta \nabla q(s, a; \theta_{pn})$
end for

Algorithm 17 `PrimitiveModelValueUpdate`()

for n iterations, for multiple transitions $(s, a, \hat{r}, \hat{s}', \hat{\gamma})$ sampled from $B_{model\_primitive}$ **do**
  $\delta \leftarrow \hat{r} + \hat{\gamma} \max_{u' \in \mathcal{U}_I} q(\hat{s}', u'; \theta_{pn}) - q(s, a; \theta_{pn})$
  $\theta_{pn} \leftarrow \theta_{pn} + \alpha \delta \nabla q(s, a; \theta_{pn})$
end for

Algorithm 18 `OptionModelValueUpdate`()

for n iterations, for multiple transitions $(s, o, \hat{s'}_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o})$ sampled from $B_{model\_o_i}$ **do**
  $\delta \leftarrow \hat{r}_{s,o} + \hat{\gamma}_{s,o} \max_{u' \in \mathcal{U}_I} q(\hat{s'}_{s,o}, u'; \theta_{model\_on}) - q(s, o; \theta_{model\_on})$
  $\theta_{model\_on} \leftarrow \theta_{model\_on} + \alpha \delta \nabla q(s, o; \theta_{model\_on})$
end for

Algorithm 19 `ModelUpdate()`

for $o_i \in \bar{\mathcal{O}}$, for multiple transitions (s, a, r, s', γ) sampled from $B_{direct\_o_i}$ **do**
  $\gamma_o \leftarrow \gamma(1 - \beta_{o_i}(s'))$
  // Update option policy
  $\delta^\pi \leftarrow \frac{1}{2}(r-1) + \gamma_o \max_{a' \in \mathcal{A}} \tilde{q}(s', a', o_i; \theta^\pi) - q(s, a, o_i; \theta^\pi)$
  $\theta^\pi \leftarrow \theta^\pi + \alpha^\pi \delta^\pi \nabla q(s, a, o_i; \theta^\pi)$
  // Update reward model and discount model
  $a' \leftarrow \text{argmax}_a q(s, o_i; \theta^\pi)$
  $\delta^r \leftarrow r + \gamma_o r_\gamma(s', a', o_i; \theta^r) - r_\gamma(s, a, o_i; \theta^r)$
  $\delta^\Gamma \leftarrow 1(\gamma_o = 0)\gamma + \gamma_o \Gamma(s', a', o_i; \theta^\Gamma) - \Gamma(s, a, o_i; \theta^\Gamma)$
  $\theta^r \leftarrow \theta^r + \alpha^r \delta^r \nabla r_\gamma(s, a, o_i; \theta^r)$
  $\theta^\Gamma \leftarrow \theta^\Gamma + \alpha^\Gamma \delta^\Gamma \nabla \Gamma(s, a, o_i; \theta^\Gamma)$
  if $\beta_{o_i}(s') == 1$ **then**
    $\hat{s'}_{s,o_i} \leftarrow \hat{s'}_{s,o_i} + \alpha(s' - \hat{s'}_{s,o_i})$
  end if
end for
// Update state to next state model (PrimitiveModel)
for n iterations **do**
  sample s, a, r, s' from $B_{direct\_primitive}$
  $\theta^r_{pm} \leftarrow \theta^r_{pm} + \alpha^r_{pm}(r(s, a, \theta^r_{pm}) - r)\nabla\theta^r_{pm}$
  $\theta^{s'}_{pm} \leftarrow \theta^{s'}_{pm} + \alpha^{s'}_{pm}(s'(s, a, \theta^{s'}_{pm}) - s')\nabla\theta^{s'}_{pm}$
  $\theta^\gamma_{pm} \leftarrow \theta^\gamma_{pm} + \alpha^\gamma_{pm}(\gamma(s, a, \theta^\gamma_{pm}) - \gamma)\nabla\theta^\gamma_{pm}$
end for

Chapter 4

An Empirical Evaluation of Dyna with Options in a Non-stationary, Tabular Environment

In this chapter, we examine Dyna with options with a tabular value function. We begin by introducing the environment and motivating why it is an appropriate testbed for investigating the benefits of planning with options. We continue by providing implementation details of the algorithms and listing the experimental design choices. Finally, we present and discuss the results.

4.1 Environment

The environment that we use to evaluate Dyna with Options is called Grazing-World (Figure 4.1) and is designed based on discussions with Rich Sutton and Adam White. It is an 8x12 grid, in which the agent starts from the bottom left corner, and can take four actions (up, down, left, right), each incurring a penalty of 1 per timestep. The environment has three terminal states (G1, G2 and G3), two of which have alternating reward schedules. For the first 250 episodes in the episode both G1 and G2 give 0 reward, then G2 gives a reward of 50 for 500 episodes, after which it reverts to giving a reward of 0 for 500 episodes. In the middle of G2's "on" state, i.e. after 500 episodes, G1 begins to give a reward of 100 for 500 steps, after which the cycle restarts. G3

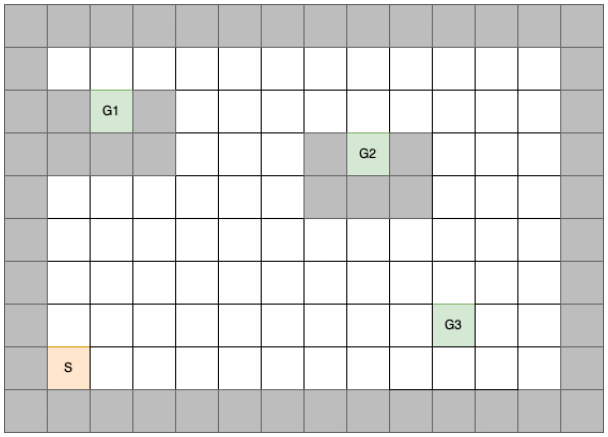


Figure 4.1: The GrazingWorld environment. Grey squares depict walls, white squares depict valid states. Goal states are marked with green and start state is represented by orange.

outputs a reward of 1 constantly. Figure 4.2 provides a visual demonstration of the reward schedule of the three terminal states. As Figure 4.1 depicts, the grid and the two goals with non-stationary rewards are surrounded by walls: if the agent takes a step to bump into them, the agent remains put with a penalty of 1. The discount rate γ is set to 0.95, so that the effective horizon $\frac{1}{1-\gamma} = 20$, well within the longest optimal trajectory length (12 steps).

The -1 reward per timestep was chosen to encourage the agent to terminate, the reward of 1 for G3 was chosen to be a low reward for the agent to revert when the rewards of the other goals are 0. The rewards of 50 and 100 were chosen to be high for the agent to want to go to these goals, and also have a substantial difference between them so that when G1 activates, the agent would switch from a decent policy to an even better one. The reward schedule was chosen to allow enough time steps for the agents to find out if rewards have changed anywhere while exploring.

At first, the GrazingWorld looks like a deceptively simple, small grid world environment. However, as we will show, the non-stationarity of the rewards makes it a challenging environment that necessitates at least one of the following:

- temporal abstraction
- planning

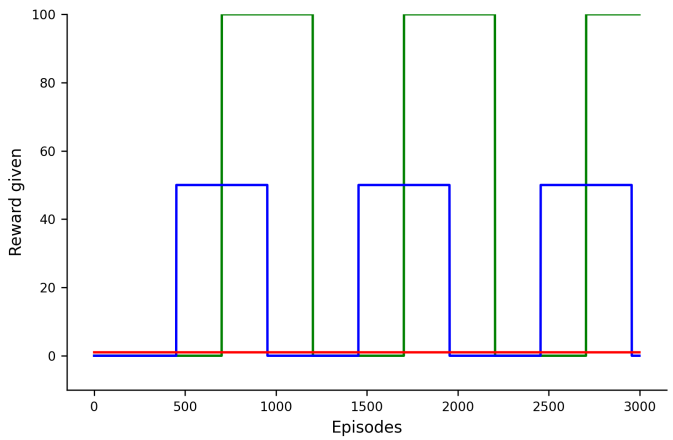


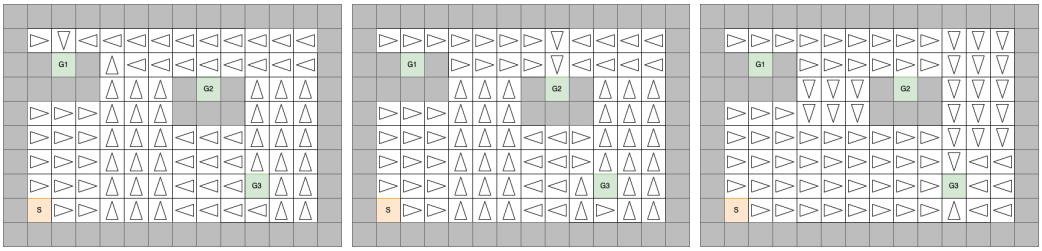
Figure 4.2: The reward schedule of the GrazingWorld environment. G1 (green) alternates between 0 and 100, G1 (blue) does so between 0 and 50, each changing after 500 episodes. G3 (red) outputs a constant reward of 1 in each episode. The plot includes 200 initial exploratory episodes, after which the cycle starts with both G1 and G2 outputting a reward of 0.

- an exploration strategy beyond basic ϵ-greedy

As discussed in Chapter 1, non-stationarity is an essential problem: the real world is non-stationary and hence most applications found in it are as well. Demonstrating the behaviour of the different approaches to tackling non-stationarity in a small environment can aid our understanding of them and thus help us design solution methods for more challenging problems.

4.2 Experiment Details

We divide the problem of finding a close-to-optimal policy in an environment with periodically changing reward function to two phases: first - leveraging the tabular environment - we learn perfect action and option models (according to the current reward function of the environment). Then, making use of the learned models and direct experience, we let the agent learn a policy that optimises for the expected return. We assume that we are given options $\mathcal{O}(I, \pi, \beta)$ where I, β and π are the predefined initiation function, termination function and policy, respectively.



(a) G1's option policy (b) G2's option policy (c) G3's option policy

Figure 4.3: Illustration of the predefined three optimal policies to the three goal states

4.2.1 Agent Details

Learning the model can be done online: as the agent interacts with the environment, it builds and improves its model of the environment and uses this ever-improving model to improve its value function; or the model learning can be executed as a prelearning phase: the agent takes random actions in the environment for a specified number of steps with the sole purpose of learning a model of the next states, rewards and state-based discount factors given the current state-action pairs. As the former setup is more difficult, and our main interest is examining the agent's behaviour of planning with options, we will have a model-learning phase in our experiments.

At the beginning of the algorithm, for a set number of episodes at each timestep the agent takes a random primitive action with probability 1 until termination. The primitive model can be implemented as a lookup-table, in which the agent stores the current state-action pairs and their corresponding next states, rewards and discount rates, and the option models can be implemented as arrays of size $\mathcal{S} \times \mathcal{O}$.

The model learning is followed by the active learning phase, in which in each episode the agent starts in the starting state and takes actions according to its policy, which is improved on each step. A model update is performed at each step just as in the initial exploration phase to keep track of changes in the environment.

The initial exploration phase, during which DynoQ+ learns the primitive action and option models was set to 200 episodes. These models, along with

the Q-table and τ were implemented as simple hash maps (dictionaries). We assumed that we had access to three options with optimal option policies (to each terminal state): these could be initiated in each state, they terminated in each terminal state and their policies were hard-coded with shared paths to their goals as shown in Figure 4.3. The Q-table $Q(\mathcal{S},\mathcal{U})$ was thus of size $(96 \times (4+3))$ for the 4 primitive actions and 3 options.

We set ϵ to a standard 0.1 value, which is empirically generally the highest that encourages the agent to explore the environment, but not as much that would be very detrimental to performance, and given the chosen reward schedule in GrazingWorld, it allowed for seeing the differences in performance between the algorithms. We swept the stepsize parameter $\alpha \in [0.1, 0.3, 0.5, 0.7, 0.9]$, and $\kappa \in [0, 0.01, 0.02, \ldots, 0.1, 0.15, \ldots, 0.4, 0.5, \ldots, 1.2]$, and we tested the algorithm with 5 planning steps per timestep. Each hyperparameter combination was evaluated by averaging the results of 30 different seed values.

4.3 Encouraging Exploration

Since the reward functions of the terminal states change significantly as time passes, the agent must occasionally visit states other than its current policy dictates. To achieve this, we rely on the state visitation bonus component of the original DynaQ+ algorithm. Here, an additional table (τ) of size ($|\mathcal{S}| \times |\mathcal{A} \cup \mathcal{O}|$) keeps being incremented by 1 after each timestep and the entry of the current state and action/option in τ is reset to 0. During each planning step, the reward from the action/option model is incremented by the square root of $\tau(s,u)$ scaled by an additional hyperparameter κ. A uniform increase of τ across states would encourage the agent to explore the environment uniformly, including states that the agent would never need to visit to stay on the given option policies, therefore we will examine three cases:

- $\kappa = 0$ (i.e. no state visitation bonus)
- uniform κ for all $s \in \mathcal{S}$

- $\kappa > 0$ on goal states, $\kappa = 0$ otherwise

The pseudocode for replacing Algorithms 8 and 11 with state visitation bonuses are provided below:

Algorithm 20 `PrimitiveModelValueUpdateKappaOnGoals`(s, a)

// Generate experience from the primitive model
$\hat{s'}, \hat{r}, \hat{\gamma} \leftarrow \text{PrimitiveModel}(s, a)$
// State visitation bonus on states where the options terminate
if $\beta_o(s) == 1$ for any $o \in \mathcal{O}_{I(s)}$ **then**
    $\hat{r} \leftarrow \hat{r} + \kappa\sqrt{\tau(s, a)}$
end if
// Update the primitive action-values
$\delta_a \leftarrow \hat{r} + \hat{\gamma} \max_{a' \in \mathcal{A}} Q(\hat{s'}, a') - Q(s, a)$
$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_a$

Algorithm 21 `OptionModelValueUpdateKappaOnGoals`(s)

// Generate experience from the option model
$o \sim unif_{discrete}(\mathcal{O}_s)$
$\hat{s}'_{s,o}, \hat{r}_{s,o}, \hat{\gamma}_{s,o} \leftarrow \text{OptionModel}(s, o)$
// State visitation bonus on states where the options terminate
if $\beta_o(s) == 1$ for any $o \in \mathcal{O}_{I(s)}$ **then**
    $\hat{r}_{s,o} \leftarrow \hat{r}_{s,o} + \kappa\sqrt{\tau(s, o)}$
end if
// Update the option-values
$\delta_o \leftarrow \hat{r}_{s,o} + \hat{\gamma}_{s,o} \max_{o' \in \mathcal{O}} Q(\hat{s'}_{s,o}, o') - Q(s, o)$
$Q(s, o) \leftarrow Q(s, o) + \alpha\delta_o$

To implement uniform κ for all $s \in \mathcal{S}$, one just needs to remove the if clause in the above two algorithms. Note that $\tau(s, a)$ and $\tau(s, o)$ for the current state-action pair s, a, and state-option pair s, o are reset to 0 at each timestep.

4.4 Results

In this section, we present the results of the experiments carried out with Dyna+options, Dyno, and Dyna in the GrazingWorld environment. We begin by discussing the results when the agents had no state visitation bonus hyper-parameter, then move on to presenting the results with uniform and goal-state visitation bonuses.

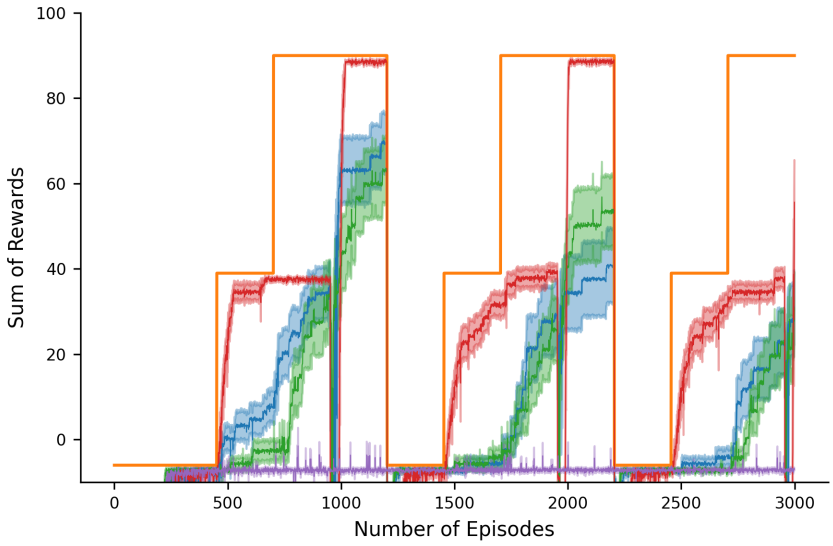


Figure 4.4: Average accumulated rewards over 30 runs per episode on the GrazingWorld environment with no state visitation bonus of Q-learning (purple), Dyna (green), Dyno (red), Dyna+options (blue). The shaded regions represent the standard errors and the orange line depicts the maximum achievable sum of rewards per episode.

4.4.1 Planning with Options without State Visitation Bonus

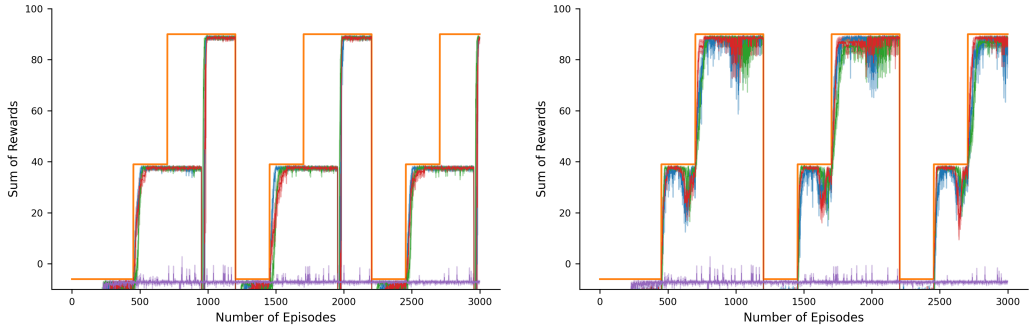
Figure 4.4 depicts the performance of the investigated algorithms with no state-visitation bonus. The initial 200 episodes cannot be seen on the plot as during those episodes all algorithms are in the exploration phase, receiving a lot of negative rewards. Given the non-stationary nature of the reward function and the slow updating of its action-value function, Q-learning fails to consistently go to G1 or G2 when they are giving high rewards: Q-learning resorts to the safe but very suboptimal G3 the majority of the time. Even without any state visitation bonus, all planning algorithms outperform Q-learning significantly. Dyna and Dyna+options perform similarly: much better than the model-free Q-learning, but significantly worse than Dyno. Dyno quickly finds the optimal goal state G2 as it begins to output a reward of 50, but after that does not explore the state space enough to find the meanwhile better G1 as it commences to emit a reward of 100 - only after G1 reverts to giving a reward of 0.

As soon as Dyno visits the current optimal goal state due to a sequence of exploratory actions, the corresponding option's reward model is updated and as the agent plans with options, each s, o pair is quickly updated across the state space. Thus, the option's value will be high in each state, the agent will choose it on each greedy step until the goal state begins to give a low reward again, which will quickly deter the agent from taking that option the same way. In comparison, Dyna plans with primitive actions only, and Figure 4.4 demonstrates that the agent is unable to update its value function to achieve a close-to-optimal policy with the given number of planning steps times the number of timesteps until the reward schedule changes.

Dyna+options performs similarly to Dyna, demonstrating that Dyna+options even with access to options with optimal option policies is severely hindered by planning with primitive actions in the given setup. Similarly to Dyno, once the Dyna+options agent visits the current optimal goal state, and the agent updates the values of the sampled state-option pairs across the state space. However, the agent also samples state-action pairs: some of them will likely be suboptimal, but their corresponding next state may have an option with a high option-value, updated while planning with options. As Q-learning updates with the greatest action-(option)-value of the next state, this suboptimal action will be given a high action-value, which the agent may decide to execute on a greedy step while interacting with the environment. As this keeps occurring throughout the state space, the agent will often go wandering and end up in suboptimal goal states, and similarly to Dyna, the amount of planning steps in the available reward schedule is not sufficient for the agent to converge to a policy that would steadily direct the agent to the then optimal goal state.

4.4.2 Planning with Options with State Visitation Bonus

Once we add uniform state-visitation bonus, Figure 4.5a depicts that all algorithms converge to a policy similar to that of Dyno without exploration bonuses: they quickly find Goal 2 when it provides high rewards, but only venture out to Goal 1 once Goal 2 gives low rewards again. With high κ, we can force the agents to find Goal 1 sooner, however with the price of being



(a) Uniform state visitation bonus (b) Goal state visitation bonus

Figure 4.5: Average accumulated rewards over 30 runs per episode on the GrazingWorld environment with (a) uniform state visitation bonus and (b) state visitation bonus only on the goal states, of Q-learning (purple), Dyna (green), Dyno (red), Dyna+options (blue). The orange line depicts the maximum achievable sum of rewards per episode.

overall more suboptimal, see Appendix A for further details. Finally, Figure 4.5b demonstrates that if we only add state-visitation bonus to goal-states, all planning algorithms perform similarly: they find the goals with the highest rewards fairly soon and continue to visit them across the episodes fairly consistently.

Chapter 5

Examining Dyna with Options with Value Function Approximation

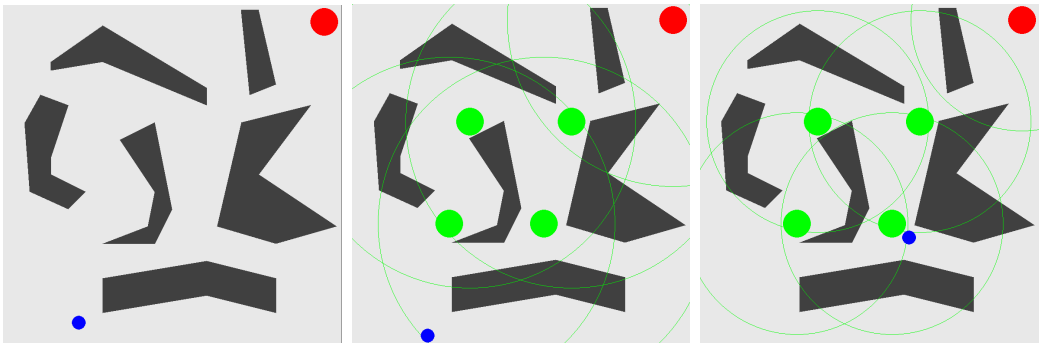
In the previous chapter we investigated Dyna with Options and demonstrated its benefits over algorithms with only its individual components in a non-stationary, tabular environment. In this chapter, we examine Dyna+options and its variants with value function approximation, a setting that is more realistic and challenging, as it can deal with much larger state spaces. As before, we first introduce the environment and motivate why it is an appropriate testbed for comparing these algorithms. Then, we list the experimental design choices, and finally present and discuss the results.

5.1 Environment

The PinBall Domain (Konidaris and A. Barto 2009a) is an RL environment, in which the agent (represented by a small blue dot) must navigate its way from the starting position to the red goal area.The state space of the agent is 4-dimensional and continuous ($x, y, \dot{x}, \dot{y} \in [0, 1]^4$). The agent is dynamic, with drag coefficient 0.995. As Figure 5.1 demonstrates, the environment has obstacles which the agent can learn to use to its advantage to reach the goal faster by bouncing off of them, rather than simply avoiding them. The agent has five primitive actions: increase or decrease the x or y velocity, incurring a penalty of 5 per step or do nothing (penalty of 1 per step). Reaching the

goal state area (irrespective of the speed of the agent) results in a reward of +10,000. The discount factor γ is 0.99.

The PinBall domain has been used as a test environment for many RL algorithms (Bacon *et al.* 2017; Konidaris, Kuindersma, *et al.* 2010; Tamar, Di Castro, *et al.* 2013). It is an influential and popular domain, because due to its dynamic nature, sudden changes make function approximation challenging, more so than e.g. Mountain Car. Given its smaller state space however, it does not require as much computation as for example the ATARI suite (Bellemare *et al.* 2012), allowing for more thorough empirical investigations.



(a) (b) (c)

Figure 5.1: The PinBall Domain a) in a single configuration, without options. The blue dot depicts the agent, and the red dot represents the final termination area, b) with four option termination areas depicted with a green (and a fifth around the final termination area with a red) dot and their initiation radii (green circles around the dots) used for model learning, c) with the reduced initiation radii used while the agent is learning its behaviour policy

5.2 Experiment Details

As in the previous chapter, we divide the problem of finding a near-optimal policy into two main parts: first, we let the agent explore the environment by taking random actions and learn both the primitive action and the option models. Then, leveraging the learned models and direct experience, we let the agent learn a policy that optimises for the return. As before, we assume that we are given options $\mathcal{O}(I, \pi, \beta)$ where I and β are the predefined initiation and termination functions and π is a policy learned via a DDQN update.

5.3 Experiment Design

In this section, we first give details about our specific implementation, followed by a detailed description of the experiments we ran to compare Dyna+options with DDQN, Dyna and Dyno.

5.3.1 Implementation of Model Learning

Environment Exploration Phase

To ensure that the agent gets a roughly uniform coverage of the entire state space while learning the action and option models, instead of letting the agent explore randomly until termination (which would cause the experienced states to be skewed very heavily towards the starting state), we reset the state of the agent according to the following schedule: the agent keeps taking random actions until either it enters the goal state or 10 timesteps have passed. In the former case, the agent starts from the start state and a new episode begins as usual. In the latter case, with probability 0.01 the agent is reset at an area, in which a random option terminates, and with probability 0.99 the agent is reset at a random valid state in the environment (i.e. not on top of an obstacle).

The option model learning described in Algorithm 19 is carried out as follows: at each timestep, the agent stores its experience in a buffer for each option ($B_{direct\_o_i}$) that is initialized in the current state s. If the option terminates in the next state we fill the buffer with $s, a, s', r, \gamma_o = 0, \pi_{cumulant\_o} = \gamma$. If the option does not terminate but is initialized in the next state, we fill the buffer with $s, a, s', r, \gamma_o = \gamma, \pi_{cumulant\_o} = 0$. If the option is not initialized in the current state s, we fill the buffer with $s, a, s', r = 0, \gamma_o = 0, \pi_{cumulant\_o} = 0$ because we want to encourage the agent to learn the policies for the experiences that keep the agent in states in which the option is initialized or terminates.

The options' discount factors and expected discounted rewards are then calculated by sampling from the buffer and performing a DDQN-update on the option discount rate and the reward, with the targets γ_{option_t}, r_{option_t} as:

$$\gamma_{option\_t} = \pi_{cumulant\_o} + \gamma_o \hat{\gamma}(s', a', \theta^-_{\gamma_{option}})$$

$$r_{option\_t} = r + \gamma_o \hat{r}(s', a', \theta^-_{r_{option}})$$

where $a' = \text{argmax}_{a \in \mathcal{A}}(\hat{\gamma}(s', a, \theta_{\gamma_{option}}))$

We implemented the option model learning with two separate neural networks, with layers $[128, 128, 128, 64, 64]$, with ReLU activations and the Adam optimizer with standard β_1 and β_2 hyperparameters, the stepsize parameter α swept in $[2^{-9}, 2^{-10}, (2^{-10} + 2^{-11})/2, 2^{-11}, 2^{-12}, 2^{-13}]$ (we found the best model with $\alpha = (2^{-10} + 2^{-11})/2$) , and the Polyak stepsize for the target network was set to 0.1.

The stepsize parameter $\alpha_{s',o}$ for learning the expected s' of each option was set to 0.005, and learned with the update below, if $\beta_o(s') > 0$

$$s'_o = \alpha_{s',o}(s' - s'_o)$$

The option policies $\pi_o(s)$ were given by $\text{argmax}_{a \in \mathcal{A}} \hat{\gamma}_{s,a,o}$, $\forall s \in \mathcal{S}$ where option o is initialized, as the higher $\hat{\gamma}_{s,a,o}$ the closer the agent is predicted to be to the option's termination area.

The primitive model was implemented similarly: three separate neural networks for the learned parameters $\hat{s}', \hat{r}$ and $\hat{\gamma}$, each with layers $[128, 128, 128, 64, 64]$, with ReLU activations and the Adam optimizer with standard β_1 and β_2 hyperparameters. The stepsize parameter was swept individually for each network: $[2^{-12}, 2^{-13}]$ for both s' and r. The stepsize parameter for learning γ was set to 2^{-13}. To account for class imbalance (all three components are very different in the goal state area than elsewhere in the environment), $\frac{1}{4}$ of the batch was forced to be sampled from the goal state area.

We used the average squared loss as the loss function for each network.

The model learning was run for 300,000 timesteps. We stored the primitive action and option experience at each step in the corresponding buffers. Each buffer was sampled uniformly to get a batch of 16 datapoints (other than the aforementioned goal area sampling) and the networks were updated at each timestep once the stored data in their corresponding buffers exceeded 10,000 datapoints. The options' termination radii were set to 0.04 (the same as the final goal area) (within which $\beta_o(s) = 1$ and 0 for all other $s \in \mathcal{S}$) and their initiation radii were set to 0.48 (within which $I_o(s) = 1$ and 0 for all other

$s \in \mathcal{S}$) during the model learning part, which was reduced to 0.32 during the behaviour policy learning, which we found crucial to get good performance out of Dyna+Options and Dyno.

Since the purpose of the model learning phase is simply to learn an as accurate model as possible, each set of hyperparameters was run with one seed, and we chose the best performing seed. We used the model which appeared to have learned the environment best according to heatmaps of the option models' predicted values and the errors of the primitive model's predictions.

5.3.2 Implementation of the Agent

The exploration strategy we chose was epsilon-greedy, with $\epsilon = 0.1$, and we ran each agent for 300,000 steps. We created two neural networks to learn the action- and option-values. Both had the layer structure: $[128, 128, 64, 64]$, with random initializations, ReLU activation functions and we used the Adam optimizer with standard β_1 and β_2 values. The stepsize parameter was swept: $\alpha \in [2^{-8}, 2^{-9} \dots 2^{-12}]$ and the Polyak stepsize for the target network was set to 2^{-6}, as we found that DDQN performs best with this hyperparameter and to limit computation.

Similarly to the model learning phase, we stored experience at each step in the corresponding buffers. Each buffer was sampled uniformly to get a batch of 16 datapoints and the networks were updated at each step - 4 times for the direct and simulated option experience and once for the simulated primitive experience - once the stored data in their corresponding buffers exceeded 10,000 datapoints, with the buffer size set to 1,000,000 after which the oldest experiences are removed - which would only ever potentially happen to the option buffers as the agents were run for only 300,000 steps.

Each set of hyperparameters was run with 30 different seeds for each agent.

5.4 Results

In this section, we present the results of executing variants of Dyna+options with value function approximation in the PinBall domain. We begin by dis-

cussing the difficulty of learning the primitive and option models, then list the hyperparameters that achieved the highest return in our experiments, and finally, we present and discuss the performance of the algorithms.

5.4.1 Difficulty of Learning the Models

Figures 5.2 and 5.3 provide the $\hat{\gamma}_{s,\cdot,o}$ and $\hat{r}_{s,\cdot,o}$ predictions for each option o for 6400 states scattered uniformly across the state space. On the figures, yellow values signify high, deep blue low values, and white areas mean that the option is not initialized in those states. We can see the shapes of obstacles with blue colours demonstrating the agent learned that those areas have low values, and high values around the options termination areas (the middle white area for each option) and gradually lower values as we move further away from them - indicating decent option models.

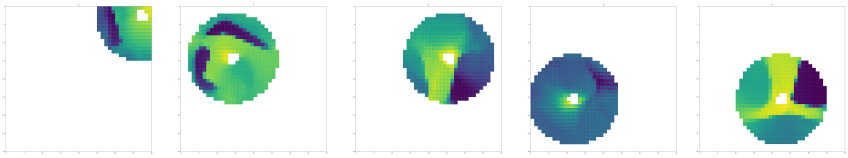


Figure 5.2: Heatmap of the discount factor predictions of option models. $\gamma \in [0, 1]$

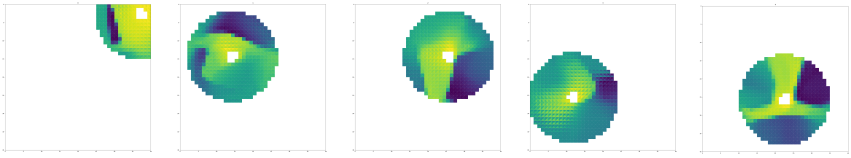


Figure 5.3: Heatmap of the reward predictions of option models. $r \in [-500, 1]$

Learning an accurate primitive model is difficult without additional assumptions about the environment. Highly accurate next state values are difficult to learn due to the set drag factor as well as the elasticity of the agent. Very small inaccuracies can cause big differences in the movement of the agent: e.g. whether an agent bumps into a wall or not. The state-based discount factor γ and the reward r are easier to learn since they are the same across most of the state space except at the final (relatively small) goal state area. However,

Model Component	Min. Error	Max. Error
Next state s'	0.00104	0.38928
Reward r	0.01407	10652.89746
Discount factor γ	0.00000	1.05156

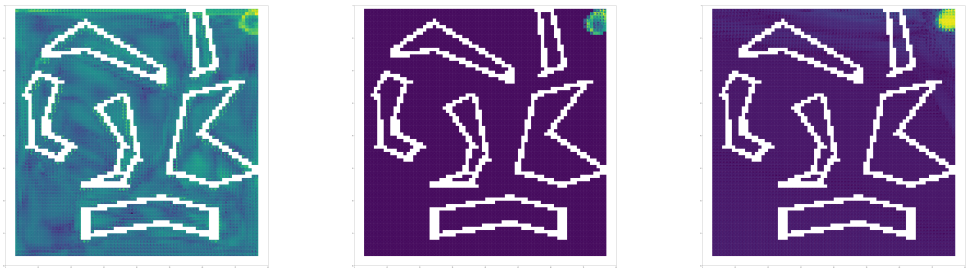
Table 5.1: Minimum and maximum errors of the primitive model's predicted and the environment's actual next state s', r, γ for 6400 states across the entire state space $\times$ 5 actions in the PinBall Domain used for Dyna+Options

the relative difference is significant for γ: 0.99 to 0, and large for the rewards: -5 to +10,000. This class imbalance is accounted for while sampling from the data, but even so, the MSE loss function causes the model to predict values between the two extreme values in most of the states. As Figure 5.4 depicts, all three model components suffer large errors close to the final goal state, but the rest of the states have low errors.

In Table 5.1 we report the minimum and maximum errors across the state space for each primitive model component:

5.4.2 Comparison of Algorithms

Table 5.2 lists the best hyperparameters from the set of hyperparameters tried for Dyna+options, Dyno, Dyna and DDQN. Both DDQN and Dyna achieve the best performance with the stepsize parameter for the primitive network



(a) Next State Errors (b) Reward Errors (c) Discount Factor Errors

Figure 5.4: Heatmap of errors between the PrimitiveModel's predicted and actual next states, rewards and state-based discount factors used for Dyna+Options in the PinBall Domain's 6400 states scattered uniformly across the state space $\times$ 5 primitive actions. Bright yellow depicts the highest, deep blue represents the lowest errors.

Alg./Hyperp.	Stepsize	Polyak Stepsize
Dyna+options	2^{-10}	2^{-6}
Dyno	2^{-10}	2^{-6}
Dyna	2^{-9}	2^{-6}
DDQN	2^{-9}	2^{-6}

Table 5.2: Hyperparameters that achieved the most cumulative rewards (averaged over 30 independent seeds) among the swept hyperparameters of Dyna+options, Dyno, Dyna and DDQN on the Pinball domain with $\gamma = 0.99$

set to 2^{-9}, while Dyno and Dyna+options do so with 2^{-10}, as they also have access to an option network, and as such their functions to be approximated are more complex.

As Figure 5.5 depicts, in the very beginning of learning, all algorithms with access to a primitive and/or option models perform similarly and slightly better than DDQN. However, the inaccuracy of the primitive model causes the performance of Dyna to plateau early and allows DDQN to outperform it already at the 20,000th step. As Dyna+Options has access to option, this allows it to offset the inaccuracy of the primitive model updates and make it perform comparably to DDQN, apart from a dip between the 120,000-220,000th step, when it performs almost as badly as Dyna. Dyno learns the quickest

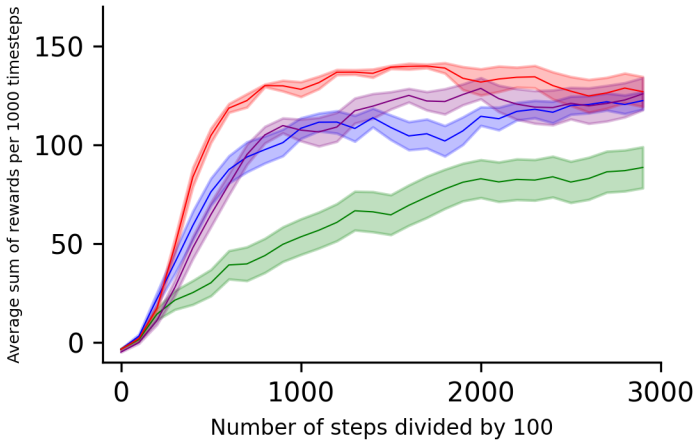


Figure 5.5: Performance of Dyna+options (blue), Dyno (red), Dyna (green) and DDQN (purple) in the Pinball Environment. Results averaged over 30 seeds with the shaded regions representing standard errors.

and the most robustly among the examined algorithms, achieving the best performance around the $150{,}000^{th}$ step, performing significantly better than any of the algorithms. Later on, however, performance degrades a bit and the agent performs comparably to DDQN in this instance of the environment with the option initiation radii, hyperparameters and seeds examined. In Appendix B we provide the results of experiments with a different, slightly more accurate primitive model. We show that Dyna and Dyna+options perform better with a more accurate primitive model, but planning with a primitive action model still hinders performance.

Chapter 6

Conclusion

In this thesis, we investigated Dyna+options in a non-stationary environment and with value function approximation with the assumption of having access to options with (near-)optimal option policies. We broke the learning process into two parts: a model-learning phase in which the agent takes random actions and learns the action- and option-models and an active learning phase, in which the agent gradually improves its policy using direct experience from the environment and simulated experience from its models.

We carried out experiments in a non-stationary, tabular environment called GrazingWorld and in a deterministic, continuous state environment that necessitates value function approximation: the PinBall Domain. We compared against baselines, the individual components of the algorithm: leveraging only direct experience (DDQN), direct experience with simulated experience from the primitive model (Dyna), and direct experience with simulated experience from the option model (Dyno). We found that Dyno performs very well even without the presence of a state visitation bonus. Dyna+options and Dyna performed similarly, hinting that planning with primitive actions can hinder performance even when having access to options with optimal option policies. Furthermore, we found that a state visitation bonus is still crucial in achieving satisfactory performance in our non-stationary environment, diminishing the differences between the examined planning algorithms in the investigated instances. When extending the algorithms with value function approximation, we found that Dyno outperforms DDQN in both speed and robustness

of learning early during training, but its performance may later on degrade to DDQN's. Furthermore, the accuracy of the primitive model is crucial in terms of performance: Dyna exacerbates the performance of DDQN, while Dyna+Options exacerbates that of Dyno's. The access to options with near-optimal option policies improved the performance of Dyna+options compared to Dyna: allowing it to perform comparably to DDQN during a significant part of the training regime, but almost as bad as Dyna during the rest of the training regime in the instances we examined.
Given that

- the Experience Replay Buffer used to train DDQN with direct experience is essentially a perfect Dyna model
- the more inaccurate the primitive model, the more suboptimal the performance
- the more complicated the environment the more difficult it is to learn the model, but
- planning with options only - even if they are suboptimal and their option models inaccurate - can demonstrably help the agent learn faster,

future work should consider whether it makes sense to direct efforts to develop methods to learn more accurate primitive models or instead develop more advanced search control methods and more sophisticated algorithms to plan with options.

Furthermore, future work could examine the crucial questions of option discovery, and learn the models concurrently with policy improvement.

References

[1] Z. Abbas, S. Sokota, E. Talvitie, and M. White, "Selective Dyna-Style Planning Under Limited Model Capacity," in *International Conference on Machine Learning*, PMLR, 2020.

[2] D. Abel, Y. Jinnai, S. Y. Guo, G. Konidaris, and M. Littman, "Policy and Value Transfer in Lifelong Reinforcement Learning," in *International Conference on Machine Learning*, 2018.

[3] D. Abel, N. Umbanhowar, K. Khetarpal, D. Arumugam, D. Precup, and M. Littman, "Value Preserving State-Action Abstractions," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, PMLR, 2020.

[4] S. Adam and L. Busoniu, "Experience Replay for Real-Time Reinforcement Learning Control," *Systems*, 2012.

[5] A. Aubret, L. matignon, and S. Hassas, "DisTop: Discovering a Topological representation to learn diverse and rewarding skills," *arXiv:2106.03853 [cs]*, 2021. arXiv: `2106.03853 [cs]`.

[6] A. Ayoub, Z. Jia, C. Szepesvari, M. Wang, and L. Yang, "Model-Based Reinforcement Learning with Value-Targeted Regression," in *International Conference on Machine Learning*, 2020. 3

[7] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, Feb. 2017. 38

[8] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *IEEE International Conference on Robotics and Automation*, 2001.

[9] G. Baldassarre, "A biologically plausible model of human planning based on neural networks and Dyna-PI models," *Workshop on Adaptive Behaviour in Anticipatory Learning Systems*, 2002.

[10] A. Barreto, R. Beirigo, J. Pineau, and D. Precup, "Incremental Stochastic Factorization for Online Reinforcement Learning.," in *AAAI Conference on Artificial Intelligence*, 2016.

[11] A. Barreto, J. Pineau, and D. Precup, "Policy Iteration Based on Stochastic Factorization," *Journal of Artificial Intelligence Research*, 2014.

[12] A. Barreto, D. Precup, and J. Pineau, "Reinforcement Learning using Kernel-Based Stochastic Factorization.," in *Advances in Neural Information Processing Systems*, 2011.

[13] A. Barreto, D. Precup, and J. Pineau, "On-line Reinforcement Learning Using Incremental Kernel-Based Stochastic Factorization," in *Advances in Neural Information Processing Systems*, 2012.

[14] A. Barreto, D. Borsa, *et al.*, "The Option Keyboard: Combining Skills in Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2019.

[15] A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup, "Fast reinforcement learning with generalized policy updates," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, 2020.

[16] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. Vol. 47, pp. 253–279, 2012. 13, 38

[17] R. Bellman, *Dynamic Programming*. Dover Publications, 1957. 8

[18] E. Brunskill and L. Li, "PAC-inspired Option Discovery in Lifelong Reinforcement Learning," in *Proceedings of the 31st International Conference on Machine Learning*, PMLR, 2014.

[19] W. Caarls and E. Schuitema, "Parallel Online Temporal Difference Learning for Motor Control," *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

[20] V. Chelu, D. Precup, and H. P. van Hasselt, "Forethought and hindsight in credit assignment," in *Advances in Neural Information Processing Systems*, 2020. 3

[21] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning Neuro-Symbolic Relational Transition Models for Bilevel Planning," *arXiv:2105.14074 [cs]*, 2021. arXiv: `2105.14074 [cs]`.

[22] T. Croonenborghs, K. Driessens, and M. Bruynooghe, "Learning relational options for inductive transfer in relational reinforcement learning," in *In Proceedings of the Seventeenth Annual International Conference on Inductive Logic Programming (ILP.*

[23] N. D. Daw and P. Dayan, "The algorithmic anatomy of model-based evaluation," *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 2014.

[24] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *International Conference on Machine Learning*, 2011.

[25] R. K. Dubey *et al.*, "SNAP:Successor Entropy based Incremental Subgoal Discovery for Adaptive Navigation," in *Motion, Interaction and Games*, 2021.

[26] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak, "Sparse graphical memory for robust planning," in *Advances in Neural Information Processing Systems*, 2020.

[27] A.-m. Farahmand, "Iterative Value-Aware Model Learning," in *Advances in Neural Information Processing Systems 31*, 2018. 3

[28] A.-m. Farahmand, A. M. S. Barreto, and D. N. Nikovski, "Value-Aware Loss Function for Model-based Reinforcement Learning," in *International Conference on Artificial Intelligence and Statistics*, 2017. 3

[29] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "TreeQN and ATreeC: Differentiable Tree-Structured Models for Deep Reinforcement Learning," in *International Conference on Learning Representations*, 2018. 3

[30] R. Gieselmann and F. T. Pokorny, "Planning-Augmented Hierarchical Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, 2021.

[31] R. Grande, T. Walsh, and J. How, "Sample Efficient Reinforcement Learning with Gaussian Processes," *International Conference on Machine Learning*, 2014.

[32] S. Grunewalder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton, "Modelling transition dynamics in MDPs with RKHS embeddings," in *International Conference on Machine Learning*, 2012.

[33] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration.," in *International Conference on Machine Learning*, 2016.

[34] W. L. Hamilton, M. M. Fard, and J. Pineau, "Efficient learning and planning with compressed predictive states.," *Journal of Machine Learning Research*, 2014.

[35] A. Harutyunyan, P. Vrancx, P. Hamel, A. Nowe, and D. Precup, "Per-Decision Option Discounting," in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, 2019.

[36] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, vol. 23, Curran Associates, Inc., 2010. 13

[37] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI Press, 2016, pp. 2094–2100. 13

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[39] R. He, E. Brunskill, and N. Roy, "Efficient Planning under Uncertainty with Macro-actions," *Journal of Artificial Intelligence Research*, vol. 40, 2011, ISSN: 1076-9757.

[40] T. Hester and P. Stone, "Real time targeted exploration in large domains," in *International Conference on Development and Learning (ICDL 2010)*, 2010.

[41] C. Hoang, S. Sohn, J. Choi, W. Carvalho, and H. Lee, "Successor Feature Landmarks for Long-Horizon Goal-Conditioned Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2021.

[42] C. Hogg, U. Kuter, and H. Muñoz-Avila, "Learning Methods to Generate Good Plans: Integrating HTN Learning and Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2010.

[43] Z. Huang, F. Liu, and H. Su, "Mapping state space using landmarks for universal goal reaching," in *Advances in Neural Information Processing Systems*, 2019.

[44] T. Jafferjee, E. Imani, E. Talvitie, M. White, and M. Bowling, "Hallucinating Value: A Pitfall of Dyna-style Planning with Imperfect Environment Models," *arXiv:2006.04363 [cs, stat]*, 2020. arXiv: 2006.04363 [cs, stat]. 3

[45] Y. Jinnai, D. Abel, D. Hershkowitz, M. Littman, and G. Konidaris, "Finding Options that Minimize Planning Time," in *International Conference on Machine Learning*, 2019.

[46] N. K. Jong, T. Hester, and P. Stone, "The utility of temporal abstraction in reinforcement learning," in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[47] J. Joseph, A. Geramifard, J. W. Roberts, J. P. How, and N. Roy, "Reinforcement learning with misspecified model classes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[48] K. Khetarpal, Z. Ahmed, G. Comanici, D. Abel, and D. Precup, "What can I do here? A Theory of Affordances in Reinforcement Learning," in *International Conference on Machine Learning*, 2020.

[49] J. Kim, Y. Seo, and J. Shin, "Landmark-Guided Subgoal Generation in Hierarchical Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2021.

[50] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains," Tech. Rep., 1992. 1

[51] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," in *Advances in Neural Information Processing Systems*, 2009. 37

[52] G. Konidaris and A. Barto, "Building portable options: Skill transfer in reinforcement learning," in *International Joint Conference on Artifical Intelligence*, 2007.

[53] G. Konidaris and A. Barto, "Efficient skill learning using abstraction selection," in *International Joint Conference on Artificial Intelligence*, 2009.

[54] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Constructing skill trees for reinforcement learning agents from demonstration trajectories," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23, Curran Associates, Inc., 2010. 38

[55] O. B. Kroemer and J. R. Peters, "A Non-Parametric Approach to Dynamic Programming," *Advances in Neural Information Processing Systems*, 2011.

[56] L. Kuvayev and R. Sutton, "Model-based reinforcement learning with an approximate, learned model," *Proc Yale Workshop Adapt Learn Syst*, 1996.

[57] B. Kveton and G. Theocharous, "Kernel-Based Reinforcement Learning on Representative States.," *AAAI Conference on Artificial Intelligence*, 2012.

[58] N. Lambert, K. Pister, and R. Calandra, "Investigating Compounding Prediction Errors in Learned Dynamics Models," *arXiv:2203.09637 [cs]*, 2022. arXiv: `2203.09637 [cs]`. 3

[59] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems*, 1989. 13

[60] G. Lever, J. Shawe-Taylor, R. Stafford, and C. Szepesvári, "Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning.," in *AAAI Conference on Artificial Intelligence*, 2016.

[61] L. J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Carnegie Mellon University, 1993. 13

[62] L.-J. Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching.," *Machine Learning*, 1992.

[63] T. A. Mann and S. Mannor, "Scaling Up Approximate Value Iteration with Options: Better Policies with Fewer Iterations,"

[64] T. A. Mann, S. Mannor, and D. Precup, "Approximate Value Iteration with Temporally Extended Actions," *Journal of Artificial Intelligence Research*, vol. 53, 2015.

[65] A. McGovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in *International Conference on Machine Learning*, 2001.

[66] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. 13

[67] D. C. Mocanu, M. T. Vega, E. Eaton, P. Stone, and A. Liotta, "Online Contrastive Divergence with Generative Replay: Experience Replay without Storing Data," *arXiv.org*, 2016. arXiv: 1610.05555.

[68] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine learning*, vol. 13, no. 1, 1993.

[69] S. Nasiriany, V. Pong, S. Lin, and S. Levine, "Planning with Goal-Conditioned Policies," in *Advances in Neural Information Processing Systems*, 2019.

[70] J. Oh, X. Guo, H. Lee, R. Lewis, and S. Singh, "Action-Conditional Video Prediction using Deep Networks in Atari Games," in *Advances in Neural Information Processing Systems*, 2015.

[71] J. Oh, S. Singh, and H. Lee, "Value prediction network," *Advances in Neural Information Processing Systems*, 2017. 3

[72] D. Ormoneit and Ś. Sen, "Kernel-Based Reinforcement Learning," *Machine Learning*, 2002.

[73] Y. Pan, H. Yao, A.-M. Farahmand, and M. White, "Hill climbing on value estimates for search-control in Dyna," in *International Joint Conference on Artificial Intelligence*, 2019.

[74] Y. Pan, M. Zaheer, A. White, A. Patterson, and M. White, "Organizing Experience: A Deeper Look at Replay Mechanisms for Sample-Based Planning in Continuous State Domains," in *International Joint Conference on Artificial Intelligence*, 2018. 3

[75] R. Pascanu *et al.*, "Learning model-based planning from scratch," *Journal of Machine Learning Research*, 2017. arXiv: 1707.06170.

[76] J. Peng and R. J. Williams, "Efficient Learning and Planning Within the Dyna Framework," *Adaptive behavior*, 1993.

[77] B. A. Pires and C. Szepesvári, "Policy Error Bounds for Model-Based Reinforcement Learning with Factored Linear Models.," in *Annual Conference on Learning Theory*, 2016.

[78] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992. 14

[79] W. B. Powell, "What you should know about approximate dynamic programming," *Wiley InterScience*, 2009.

[80] S. Ritter, R. Faulkner, L. Sartran, A. Santoro, M. Botvinick, and D. Raposo, "Rapid Task-Solving in Novel Environments," *arXiv:2006.03662 [cs, stat]*, 2021. arXiv: 2006.03662 [cs, stat].

[81] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, MIT Press, 1986, pp. 318–362. 12, 16

[82] T. Schaul, D. Horgan, K. Gregor, and D. Silver, "Universal Value Function Approximators," in *International Conference on Machine Learning*, 2015, ch. Machine Learning.

[83] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," in *International Conference on Learning Representations*, 2016.

[84] J. Schrittwieser *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, 2020. 3, 9

[85] D. Silver, H. Hasselt, *et al.*, "The Predictron: End-To-End Learning and Planning," in *International Conference on Machine Learning*, 2017. 3

[86] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *International Conference on Machine Learning*, 2008. 3

[87] S. Singh, A. Barto, and N. Chentanez, "Intrinsically Motivated Reinforcement Learning.," in *Advances in Neural Information Processing Systems*, 2004. 3

[88] V. Soni and S. Singh, "Using Homomorphisms to transfer options across continuous reinforcement learning domains," in *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, ser. AAAI'06, Boston, Massachusetts: AAAI Press, 2006.

[89] J. Sorg and S. Singh, "Linear options," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, ser. AAMAS '10, Toronto, Canada: International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[90] M. Stolle and D. Precup, "Learning Options in Reinforcement Learning," in *Abstraction, Reformulation, and Approximation*, 2002.

[91] A. Strehl and M. Littman, "An analysis of model-based Interval Estimation for Markov Decision Processes," *Journal of Computer and System Sciences*, 2008.

[92] R. Sutton, C. Szepesvári, A. Geramifard, and M. Bowling, "Dyna-style planning with linear function approximation and prioritized sweeping," in *Conference on Uncertainty in Artificial Intelligence*, 2008.

[93] R. Sutton, "Integrated architectures for learning planning and reacting based on approximating dynamic programming," in *International Conference on Machine Learning*, 1990. 9

[94] R. Sutton, "Integrated modeling and control based on reinforcement learning and dynamic programming," in *Advances in Neural Information Processing Systems*, 1991.

[95] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018. 1, 6–9

[96] R. S. Sutton, A. R. Mahmood, and M. White, "An emphatic approach to the problem of off-policy temporal-difference learning.," *The Journal of Machine Learning Research*, 2016.

[97] R. S. Sutton, J. Modayil, *et al.*, "Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction," in *International Conference on Autonomous Agents and Multiagent Systems*, 2011. 7

[98] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, 1999. 2, 3, 11, 16

[99] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, 1988. 8

[100] R. S. Sutton, M. C. Machado, *et al.*, "Reward-Respecting Subtasks for Model-Based Reinforcement Learning," *arXiv:2202.03466 [cs]*, 2022. arXiv: 2202.03466 [cs]. 3

[101] E. Talvitie, "Model regularization for stable sample roll-outs," in *Uncertainty in Artificial Intelligence*, 2014. 3

[102] E. Talvitie, "Self-Correcting Models for Model-Based Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2017. 3

[103] A. Tamar, D. Di Castro, and S. Mannor, "Temporal difference methods for the variance of the reward to go," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, PMLR, 17–19 Jun 2013, pp. 495–503. 38

[104] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value Iteration Networks," in *Advances in Neural Information Processing Systems*, 2016. 3

[105] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor, "A Deep Hierarchical Approach to Lifelong Learning in Minecraft," *arXiv:1604.07255 [cs]*, 2016. arXiv: 1604.07255 [cs].

[106] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI conference on artificial intelligence*, 2016.

[107] H. P. van Hasselt, M. Hessel, and J. Aslanides, "When to use parametric models in reinforcement learning?" In *Advances in Neural Information Processing Systems*, 2019. 3

[108] H. van Seijen and R. Sutton, "Efficient Planning in MDPs by Small Backups," in *International Conference on Machine Learning*, 2013.

[109] H. van Seijen and R. Sutton, "A deeper look at planning as learning from replay," in *International Conference on Machine Learning*, 2015.

[110] A. Venkatraman, M. Hebert, and J. A. Bagnell, "Improving Multi-Step Prediction of Learned Time Series Models," in *AAAI Conference on Artificial Intelligence*, 2015. 3

[111] T. J. Walsh, S. Goschin, and M. Littman, "Integrating Sample-Based Planning and Model-Based Reinforcement Learning," in *AAAI Conference on Artificial Intelligence*, 2010.

[112] Y. Wan, A. Naik, and R. S. Sutton, "Average-Reward Learning and Planning with Options," in *Advances in Neural Information Processing Systems*, 2021. arXiv: 2110.13855. 3

[113] Y. Wan, M. Zaheer, A. White, M. White, and R. S. Sutton, "Planning with Expectation Models," in *International Joint Conference on Artificial Intelligence*, 2019.

[114] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, 1992. 1, 8

[115] P. Wawrzyński and A. K. Tanwani, "Autonomous reinforcement learning with experience replay," *Neural Networks*, 2013.

[116] T. Weber *et al.*, "Imagination-Augmented Agents for Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems*, 2017. 3

[117] M. White, "Unifying task specification in reinforcement learning," in *International Conference on Machine Learning*, 2017.

[118] D. Wingate, K. D. Seppi, C. B. Edu, and C. B. Edu, "Prioritization Methods for Accelerating MDP Solvers," *Journal of Machine Learning Research*, 2005.

[119] J. Wolfe, B. Marthi, and S. Russell, "Combined Task and Motion Planning for Mobile Manipulation," *International Conference on Automated Planning and Scheduling*, 2010.

[120] H. Yao, S. Bhatnagar, and D. Diao, "Multi-step linear dyna-style planning," in *NIPS*, 2009.

[121] H. Yao, C. Szepesvari, R. S. Sutton, J. Modayil, and S. Bhatnagar, "Universal Option Models," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2014.

[122] H. Yao, C. Szepesvári, B. A. Pires, and X. Zhang, "Pseudo-MDPs and factored linear action models," in *ADPRL*, 2014.

[123] T. Zahavy, A. Hasidim, H. Kaplan, and Y. Mansour, "Planning in Hierarchical Reinforcement Learning: Guarantees for Using Local Policies," in *Proceedings of the 31st International Conference on Algorithmic Learning Theory*, PMLR, 2020.

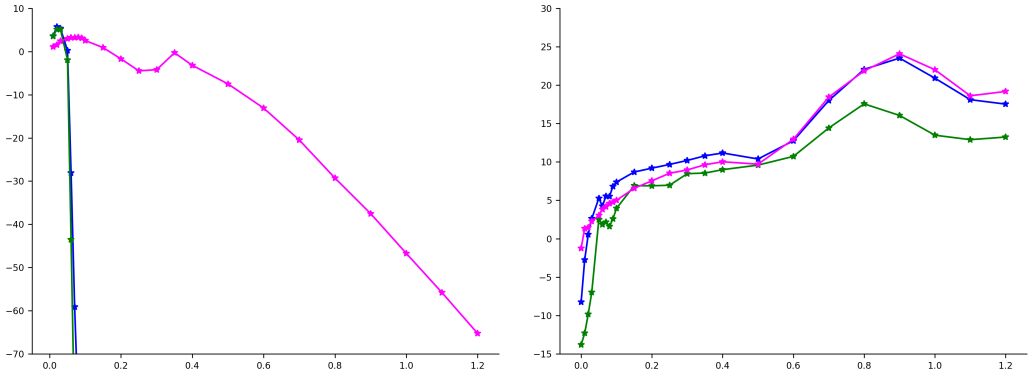
[124] L. Zhang, G. Yang, and B. C. Stadie, "World Model as a Graph: Learning Latent Landmarks for Planning," in *International Conference on Machine Learning*, 2021.

[125] T. Zhang, S. Guo, T. Tan, X. Hu, and F. Chen, "Generating adjacency-constrained subgoals in hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2020.

Appendix A

Hyperparameter Sensitivity in the Non-stationary Tabular Environment

In this section, we provide the plots of sensitivity to the κ hyperparameter of the investigated planning algorithms in the GrazingWorld environment. Figure A.1a depicts that Dyna and Dyna+options are both very sensitive to the κ hyperparameter: the best being 0.02 among the investigated values. Dyno on the other hand is a lot less sensitive: it achieves very similar near-optimal performance between values of 0 and 0.1, and performance degrades much slower as we increase κ.



(a) Uniform state visitation bonus (b) Goal state visitation bonus

Figure A.1: Sensitivity curve of the kappa hyperparameter for Dyna+options (blue), Dyno (magenta), Dyna (green) in the GrazingWorld environment with a) uniform state visitation bonus and b) state visitation bonus only on goal states

Figure A.1b shows the κ hyperparameter sensitivity of the investigated planning algorithms when it is only added to the goal states. In this setup the difference between the algorithms is not as striking, however, Dyna consistently performs worst across all 24 values, and the gaps are greatest at low and high values of κ. Dyna+options performs similarly to Dyno in this setting, but the gaps are greatest at very low and high κ values. As the near-optimal policy in this non-stationary setting is largely dependent on an appropriate exploration strategy, the results may indicate that planning with options with optimal option policies may make finding such an appropriate exploration strategy easier.

Appendix B

Results with Value Function Approximation with a More Accurate Primitive Action Model

In this section, we provide results for the investigated planning algorithms in the PinBall Domain with a more accurate primitive action model. Table B.1 shows the smallest and largest errors between this model's $\hat{s}', \hat{r}, \hat{\gamma}$ predictions for 6400 states $\times$ 5 actions - where the states had been taken uniformly from across the state space - and the environment's actual s', r, γ for these state-action pairs. Figure B.1 shows the heatmap of these errors across the state space.

After having learned this new primitive model, we reran Dyna and Dyna+options with the same settings as in Chapter 5 and swept the same hyperparameter ranges. Dyna performed best with stepsize $\alpha = 2^{-9}$ and Dyna+options with $\alpha = 2^{-9}$. As Figure B.2 demonstrates, both Dyna and Dyna+options perform

Model Component	Min. Error	Max. Error
Next state s'	0.00139	0.50985
Reward r	0.00008	9496.874
Discount factor γ	0.00000	1.01251

Table B.1: Minimum and maximum errors of the primitive model's predicted and the environment's actual next state s', r, γ across the entire state space in the PinBall domain used for Dyna and Dyna+options

(a) Next State Errors (b) Reward Errors (c) Discount Factor Errors

Figure B.1: Heatmap of errors between a more accurate PrimitiveModel's predicted and actual next states, rewards and state-based discount factors used for Dyna+Options. Bright yellow depicts highest, deep blue represents the lowest errors.

significantly better with a more accurate model than with a less accurate one, and Dyna+options achieves better performance than DDQN in the first half of the training steps. However, just as before even one planning step with the primitive action model per timestep is sufficient to exacerbate the performance of Dyna+options compared to Dyno, and that of Dyna compared to DDQN.

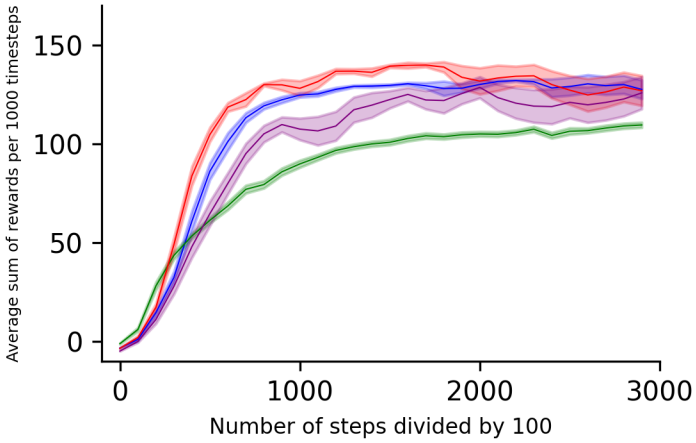


Figure B.2: Performance of Dyna+options (blue), Dyno (red), Dyna (green) and DDQN (purple) in the Pinball Environment. Results averaged over 30 seeds with the shaded regions representing standard errors.