Investigating Deep Successor Reinforcement Learning

Johnathan Tucker
Smead Department of Aerospace Engineering Sciences
University of Colorado Boulder
Boulder, CO 80309
`jotu9759@colorado.edu`

1 Introduction

In nature, animals collect experience that they leverage to create a policy which, in turn, will maximize the reward they receive. The reinforcement learning framework seeks to capture this process in a general manner so that it can be applied to a variety of agents. Reinforcement learning algorithms can be broken down into two broad categories: model-based and model-free methods. Agents that learn from a model-based method build a representation of aspects of the environment through interactions with the transition and reward functions. Since the model-based agent either has access to the environment model or learns it, these agents learn a policy that is more robust to changes in the environment. Agents that learn from model-free methods typically learn faster but have a comparatively more compact representation of the environment. The model-free environment representation comes in the form of a value function that is typically approximated with a neural network. A significant limitation of reinforcement learning is that the neural network function approximators require copious amounts of training experience. The successor feature representation of the reinforcement learning problem seeks to provide an alternative to the model-based and model-free paradigm that will allow for efficient transfer between tasks. In particular, the use of the successor feature representation can lead to the extraction of "sub-goals" which can be reused and combined to solve other tasks faster. For example, if an agent is tasked to navigate from one room to another it must pass through a doorway; the doorway is thus identified as a sub-goal that is necessary for task completion.

For this project I sought to investigate the capabilities of the successor feature representation by accomplishing three tiers of practical implementation. First I will implement the "Deep Q-Networks" (DQN) based reinforcement learning method from scratch in Julia on the Atari breakout environment. This is an extension of the work done in homework five that will require a convolutional neural network to process the image inputs from the Atari environment as seen in [6]. Next I will implement the "Deep Successor Reinforcement Learning" (DSRL) algorithm presented by Kulkarni et al. [5]. I will compare the performance of DSRL to DQN on the Atari breakout environment, which will verify that the DSRL agent has learned a policy to consistently achieve a high score. Finally, I will attempt to extract sub-goals from the DSRL agent's policy and analyze the results.

2 Background and Related Work

Understanding the challenge of completing each project sub-task begins with understanding the theory behind them. Prior to developing the theory of Deep Q-Networks, Deep Successor Reinforcement Learning, and sub-goal extraction, the theory behind reinforcement learning (RL) should be understood. The RL process can be understood as a Markov Decision Process which is described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$. Here $\mathcal{S}$ is the set of possible environment states, $\mathcal{A}$ is the set of possible actions the agent can take, $\mathcal{T}$ is the transition function, $\mathcal{R}$ is the reward function, and γ is a discount factor. Through interactions with the environment, the agent seeks to find a policy π that maps environment states to agent actions in such a way that maximizes the discounted future reward.

2.1 Deep Q-Networks

With an understanding of the basic reinforcement learning problem, the theory of Deep Q-Networks can be developed. The goal of DQN is to find the optimal state-action value function $Q*(s,a) = \max_\pi \mathcal{E}[R_t|s_t = s, a_t = a, \pi]$. The state-action value function obeys the Bellman equation seen below in Equation 1. This equation states that the optimal state-action value function is equal to the immediate reward plus the discounted maximally attainable reward according to the policy π.

$$Q^*(s,a) = E_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a\right] \tag{1}$$

Bellman's equation can be written as an iterative updater as seen in Equation 2.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha(R_{t+1} + \gamma \max_{a\in\mathcal{A}} Q(s_{t+1},a) - Q(s_t,a_t)) \tag{2}$$

This allows the optimal state-action value function to be estimated and approximated as the number of iterations approaches infinity. In practical problems the state-action value function is approximated by a neural network whose parameters are updated using the following loss function.

$$L_i(\theta_i) = E_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right] \tag{3}$$

Here the target in the mean squared error loss is $y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \mid s,a\right]$. The Julia package Flux.jl will be used to perform stochastic gradient descent to optimise the loss function and update the parameters of the state-action function.

2.2 Deep Successor Reinforcement Learning

The theory behind Deep Successor Reinforcement Learning seeks to extend Dayan's successor representation [4]. First we suppose the reward can be computed as:

$$r(s,a) = \phi(s,a)^T w \tag{4}$$

Here $\phi(s,a) \in \mathbb{R}^d$ are features of the state and $w \in \mathbb{R}^d$ are weights. Now we can rewrite the definition of the state-action function using the alternative reward representation as so:

$$\begin{aligned}
Q^\pi(s,a) &= E^\pi[r_t + \gamma r_{t+1} + ...|S_t = s, A_t = a] && (5)\\
&= E^\pi[\phi_t^T w + \gamma \phi_{t+1}^T w + ...|S_t = s, A_t = a] && (6)\\
&= E^\pi[\sum_{i=t}^{\infty} \gamma^{i-t}\phi_i|S_t = s, A_t = a]w && (7)\\
&= \psi^\pi(s,a)^T w && (8)
\end{aligned}$$

Here $\psi^\pi(s,a)$ represents the successor features of a state-action pair under a policy π. Furthermore, each component of ψ gives the discounted sum of state features ϕ. Barreto et al.[2] showed that ψ can be updated using the following pseudo-Bellman equation:

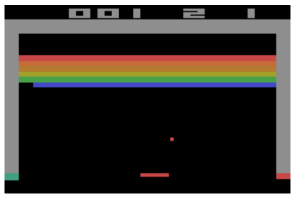
$$\psi^\pi(s,a) = \phi_t + \gamma E^\pi[\psi^\pi(s_{t+1},\pi(s_{t+1}))] \tag{9}$$

Practically, DSRL is broken into four neural network function approximators. The first is an encoder network (f_θ) that will extract the state feature vector ϕ. This state feature vector is fed into a convolutional decoder network($g_{\tilde{\theta}}$), a dense weight network(w), and a dense successor representation network ($u\alpha$). The loss functions for these networks can be seen below and are optimised using stochastic gradient descent. Notice the state feature vector is optimised to be a predictor for the immediate reward in addition to being a discriminator for states. This state feature vector is then used to update the successor representation network.

$$L_t^m(\alpha,\theta) = \mathbb{E}\left[\left(\phi(s_t) + \gamma u_{\alpha_{\text{prev}}}\left(\phi_{s_{t+1}},a'\right) - u_\alpha(\phi_{s_t},a)\right)^2\right] \tag{10}$$

$$L_t^r(\mathbf{w},\theta) = (R(s_t) - \phi_{s_t}\cdot\mathbf{w})^2 \tag{11}$$

$$L_t^a(\tilde{\theta},\theta) = \left(g_{\tilde{\theta}}(\phi_{s_t}) - s_t\right)^2 \tag{12}$$


(a) Image of the breakout game.

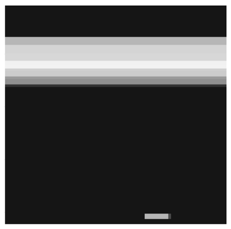

(b) Image showing the cropped grayscale breakout input.

Figure 1: Breakout screen output and input to the model.

2.3 Other Related Work

Outside of the theoretical background presented above that is necessary to grapple with the project sub-tasks; there are other works that have attempted to investigate the successor feature representation. Stachenfeld et al. [8] took a cognitive science based approach and showed that the successor feature representation could explain the spatial predictive map that is in the human hippocampus. Outside of cognitive science, successor features have been applied to task transfer in works that grapple with similar environments [10] as well as dissimilar environments [1]. Finally, the idea of sub-goals naturally leads to the development of multiple policies based on the preference introduced by the environment reward function. When this reward function can change over time, maintaining these policies in the form of options leads to the agents ability to flexibly adapt [7].

3 Problem Formulation

When humans perform a task they consistently take in visual observations and decide which action they would like to take in order to maximize their utility function. In this project, the task will be to complete the Atari 2600 game Breakout seen in Figure 1a. The reinforcement learning agent will take in the state of the environment and act according to it's learned policy to complete the game. This work takes advantage of the Arcade Learning Environment (ALE)[3] which is an Atari 2600 test suite for reinforcement learning. ALE provides two options for the state of game: an image of the screen or the current RAM state. In this project, a cropped grayscale image of the screen was used to represent the state of the environment. Each state image is cropped such that the score, life, and player counts are removed leaving only the bricks and paddle as seen in Figure 1 below. This cropping was done to restrict the image to the features that are most important for the agent to extract.

In addition, the reinforcement learning agent will act in the environment through a maximum possible eighteen actions. These actions replicate a traditional Atari joystick pad where each axis has three positions in addition to a single button. The arcade learning environment provides a method for the extraction of the minimal action set required to play a specific game, which is what will be used in all algorithm implementations. For breakout the minimal action set is:

$$\mathcal{A} = \begin{cases} \text{no-op} \\ \text{fire} \\ \text{right} \\ \text{left} \end{cases} \tag{13}$$

The arcade learning environment defines the reward as the difference between the game score across frames with the initial score being zero. The transition between frames is deterministic and the next state is provided by the game given the action input at the current state. The final component of the five tuple MDP is the discount factor, which was set to 0.99 for breakout.

4 Solution Approach

With the problem and algorithms described in detail, we can fully describe the solution process that will be used to solve the Atari breakout environment. The Deep Q-Networks algorithm was implemented completely from scratch in Julia and the Deep Successor Reinforcement Learning algorithm was implemented using the ReinforcementLearning.jl package [9].

4.1 DQN Model Architecture

The neural network function approximator includes three convolutional layers and two dense layers which break the problem of approximating the state-action function into two distinct components. The first processes the image of the game screen and extracts useful features for value determination and the second uses dense layers to process the extracted features to determine the value that corresponds to each action. The full architecture can be seen below in Table 1 below.

Module	Layers	Channels	Filter	Stride
f_θ	Conv	$4 \rightarrow 32$	(8,8)	4
	Conv	$32 \rightarrow 64$	(4,4)	2
	Conv	$64 \rightarrow 64$	(3,3)	1
	Dense	$7 * 7 * 64 \rightarrow 512$	-	-
	Dense	$512 \rightarrow 4$	-	-

Table 1: Deep Q-Networks Architecture

Algorithm 1 describes the process that was implemented in Julia to train an agent with a neural network state-action function approximator. The method processes 40 million frames in less than 24 hours which yields the results seen in the Figure 2.

Algorithm 1 Deep Q Networks

```
Initialize replay buffer, network parameters, and exploration constant
Initialize game and get initial state
while True do
    for Episode Steps do
        Increment frame count
        According to $\epsilon$-greedy take either a random action or use the argmax Q(s,a)
        Push the transition to the replay buffer
        if Every four frames then
            Sample a mini-batch
            Perform gradient descent on the loss $L_i$ seen in Equation 3.
        end if
    end for
    Anneal exploration variable
    Increment the running reward
    if Running reward is high enough then
        Break while loop
    end if
end while
```

4.2 DSRL Model Architecture

Unlike with Deep Q-Networks, there are not several publicly available implementations of Deep Successor Reinforcement Learning to base my implementation on. Therefore, I needed to create a new encoder-decoder architecture that would allow for the extraction of the state feature embeddings. These embeddings were extracted using the embedder (f_θ) then used as inputs to the successor feature (u_α), decoder ($g_{\tilde{\theta}}$), and weight (w) networks. The architecture can be seen in Table 2 below.

Module	Layers	Channels	Filter	Stride
u_α	Dense	$512 \rightarrow 4$	-	-
$g_{\tilde{\theta}}$	Conv	$1 \rightarrow 64$	(2,2)	2
	Conv	$64 \rightarrow 1$	(7,7)	3
f_θ	Conv	$4 \rightarrow 32$	(8,8)	4
	Conv	$32 \rightarrow 64$	(4,4)	2
	Conv	$64 \rightarrow 64$	(3,3)	1
	Dense	$7 * 7 * 64 \rightarrow 512$	-	-
w	Dense	$512 \rightarrow 512$	-	-

Table 2: Deep Successor RL Architecture

The following algorithm describes the process used to train a DSRL agent on the Atari breakout environment. This was implemented using the ReinforcementLearning.jl package with the hope that it can be contributed to the package as an alternative policy and learner to the existing state-action based policy and learner components of the package.

Algorithm 2 Deep Successor Reinforcement Learning

```
Initialize replay buffer, network parameters, and exploration constant
Initialize game and get initial state
while True do
    for Episode Steps do
        Increment frame count
        According to $\epsilon$-greedy take either a random action or use the argmax $u_\alpha \cdot w$
        Push the transition to the replay buffer
        if Every four frames then
            Sample a mini-batch
            Perform gradient descent on the loss $L^r(w, \theta) + L^\alpha(\tilde{\theta}, \theta)$.
            Perform gradient descent on $L^m(\alpha, \theta)$
        end if
    end for
    Anneal exploration variable
    Increment the running reward
    if Running reward is high enough then
        Break while loop
    end if
end while
```

5 Results

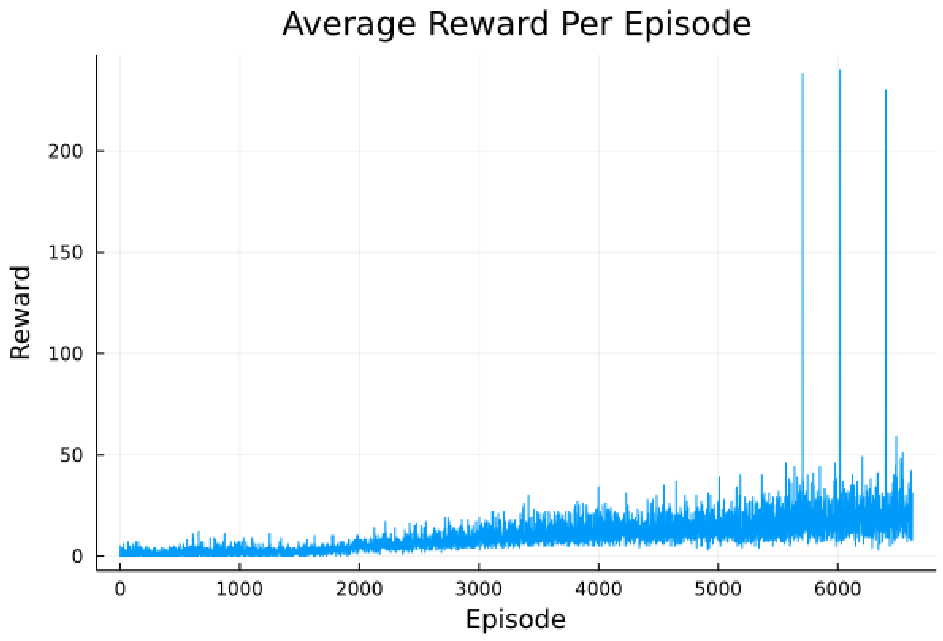


Figure 2: DQN learning curve.

In Figure 2 above, the learning curve for the Double Q-Networks agent is plotted. From this curve it is clear that the agent learns slowly as it does not reach a score plateau until episode 5000 out of 7000. The plateau occurs at a reward of 20 with the maximum reward achieved being 232. The policy that achieved the maximum reward was saved and used for evaluation. The Deep Q-Networks paper reported an average score of 168 and a maximum reward of 225. Ultimately this means the hyperparameters I used in my implementation need to be tuned until the DQN agent is able to consistently reach a score of about 150.

6 Conclusion

The sub-tasks of this project were threefold: first I would implement Deep Q-Networks on the Atari breakout environment and compare the results to the paper, second I would implement the Deep Successor Reinforcement Learning algorithm and compare the performance to the DQN results, and finally I would attempt to extract sub-goals from DSRL policy. Ultimately the goal of the project was to complete an exercise that would allow me to familiarize myself with DQN, DSRL, and the usefulness of sub-goal extraction. Although I only implemented DQN and DSRL and only got results on the breakout environment from DQN I do think this project was instructive. Through the DSRL implementation I was able to create a convolutional neural network architecture from scratch in addition to writing code that could potentially be contributed to the ReinforcementLearning.jl package.

7 Contributions and Release

John is the sole contributor to this project. The author grants permission for this report to be posted publicly.

References

[1] Majid Abdolshah, Hung Le, Thommen Karimpanal George, Sunil Gupta, Santu Rana, and Svetha Venkatesh. A new representation of successor features for transfer across dissimilar environments. In *International Conference on Machine Learning*, pages 1–9. PMLR, 2021.

[2] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

[5] Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Rahul Ramesh, Manan Tomar, and Balaraman Ravindran. Successor options: An option discovery framework for reinforcement learning. *arXiv preprint arXiv:1905.05731*, 2019.

[8] Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643–1653, 2017.

[9] Jun Tian and other contributors. Reinforcementlearning.jl: A reinforcement learning package for the julia programming language, 2020.

[10] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017.