Hierarchical Approaches to Reinforcement Learning Using Attention Networks

A Dual Degree Project Report

submitted by

JOE KURIAN EAPPEN

under the guidance of

PROF. BALARAMAN RAVINDRAN

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Hierarchical Approaches to Reinforcement Learning Using Attention Networks**, submitted by **Joe Kurian Eappen (EE13B080)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology** and degree of **Masters of Technology** , is a bonafide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Balaraman Ravindran
Research Guide
Professor
Dept. of Computer Science & Engineering
IIT-Madras, 600036

Place: Chennai

Prof. Aravind R.
Project Co-Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600036

Place: Chennai

Date: June 2018

ACKNOWLEDGEMENTS

I would like to thank Prof. Ravindran for introducing me to and giving me an opportunity to work in the rapidly developing field of deep reinforcement learning. His patient guidance along with his enthusiasm in teaching were a big part of what drew me to this pursuit and inspired me to continue my studies further. I am also indebted to Prof. David Koilpillai for giving me my first major opportunity to conduct research as well as Prof. Kaushik Mitra and Prof. Aravind for their support. Finally, I would like to thank my friends who kept me sane, my parents who were always by my side, and all the professors who taught me during my stay at IIT Madras.

Joe Kurian Eappen

ABSTRACT

While reinforcement learning (RL) has been used to solve sequential decision problems, it has always been hard to solve environments with a sparse reward feedback. This is due to the agent often requiring a very specific series of action selections in order to receive positive reinforcement. Two techniques used to address this issue are to use knowledge from past problems (transfer learning) and building a hierarchy to simplify the decision space (hierarchical reinforcement learning). In this work we ask whether we can use ideas introduced in Rajendran *et al.* (2017) for transfer learning to help create a hierarchical framework for RL. We devise an Actor-Critic based implementation of such an approach along with a feedforward and recurrent architecture. We then examine its strengths and weaknesses in a series of tasks created in Vizdoom and Gridworlds.

KEYWORDS: Reinforcement Learning, Deep Learning, Transfer Learning, Hierarchical Reinforcement Learning

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1 Reinforcement Learning

A key aspect of any intelligent system, is the ability to make decisions based on a given input. In reality, these systems often act in a stochastic environment with a goal in mind. reinforcement learning (Sutton and Barto, 1998) is a method used to approach this problem when the decision-making system (called the *agent*) has access to a scalar signal that captures the desirability of the current scenario with respect to the goal. It models the problem as a Markov decision process(Puterman, 2014) and attempts to use this model to achieve its goal. The main components in the model are the notions of state, action and reward. The agent perceives the *state* (s) of the environment, it decides an *action* (a) to take, then the environment changes based on the action. The end of this change is marked by the agent receiving a scalar value called the *reward* (r). Clearly, the autonomy of the agent lies in the way it selects this action depending on the observed state. This mechanism is called the *policy* of the agent often denoted by π and is conditioned on the state s. The Markov assumption of our model leads us to ignore past states in the agent's history while making the current action decision. Since the agent's goal is captured in the reward signal, one obvious way for the agent to act would be to attempt to maximise the cumulative reward. This cumulative reward is called the *return* and is denoted by G_t. Very often, directly attempting to maximise this cumulative reward leads to instability owing to its unboundedness. A practical method to get around this is to use *discounting* in order to diminish the importance of our estimate on rewards far into the future. Thus we have,

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...$$

where $\gamma \in (0, 1)$. The Value $V(s)$ of state s is the expected return from that state. Thus $V(s) = \mathbb{E}[G_t | s_t = s]$. MDPs have been studied extensively and approaches have been introduced to solve them such as value iteration and policy iteration. These approaches

assume use a tabular estimation approach for the Value of a state and solve the MDP using the Bellman optimality equation, brought about by the Markov property of the MDP.

1.2 Deep Reinforcement Learning

For decades, RL has been studied using toy examples with a finite state space but often failed in several real-world scenarios simply due to the vast state space encountered. This vastness in size was a constant hurdle until the generalization capabilities of deep learning (DL) (LeCun *et al.*, 2015) came to light. Attempts to utilize deep neural networks as function approximators in RL were not widely successful until the Deep Q-network (Mnih *et al.*, 2015) which could learn directly from the raw image of an Atari 2600 game was introduced. Since then several advancements in deep reinforcement learning (DRL) [Guo *et al.* (2014); Schulman *et al.* (2015); Lillicrap *et al.* (2015); Schaul *et al.* (2015); Mnih *et al.* (2016); van Hasselt *et al.* (2016); Vezhnevets *et al.* (2016); Jaderberg *et al.* (2017)] have brought about a new age of RL. The general building blocks for representation learning provided by deep learning provide a differentiable, end-to-end technique which does not require feature engineering. For example, Convolutional neural networks (CNNs) [Krizhevsky *et al.* (2012)] are widely used as a component in image processing pipelines while Long Short-Term Memory Units (LSTMs) [Hochreiter and Schmidhuber (1997)] are used to model sequential data efficiently.

1.3 Contributions

Reinforcement learning (RL) algorithms have the common aim of handling decision-making problems when faced with levels of uncertainty in the outcome. High dimensional and sparse reward environments have been a constant hurdle with several methods coming up in response. One of these is using previous knowledge of a set of problems to solve new problems (transfer learning). It is desired to have an absence of 'negative transfer', the situation that arises when following the source solution is unfavourable in the target domain. Another approach used to handle sparse reward environments is by simplifying the decision space for the agent by adding several levels of action selection

(hierarchical reinforcement learning). These levels may be created by including both spatial and temporal abstractions in the action space.

The major contributions of this work are a transfer learning technique for hierarchical reinforcement learning based on a 'hard' attention model for switching between source task solutions (or sub-policies) and a learnable base network along with an Actor Critic based implementation of this approach. This paper also explores the utility of the A2T framework (Rajendran *et al.*, 2017) in a scalable and parallelizable algorithm like A3C.

A significant advantage of this approach over a soft-attention method is that it can be easily extended to complicated hierarchies with a simple switching structure. We discuss the framework in Chapter 4, describe the environments used along with our experimental setup in Chapter 5, and discuss our observations in Chapter 6.

CHAPTER 2

Background

2.1 Reinforcement Learning

We assume a finite-horizon discounted Markov decision process (MDP) formulation , defined by $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma, T)$, in which S is the state space, A the action space, P a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \rightarrow R$ a reward function, ρ_0 an initial state distribution, $\gamma \in (0, 1]$ a discount factor, The goal of RL is to maximize the total expected discounted return $\mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_{t+1} | s_0 = s\right]$ with T being the horizon and s_0, the starting state being from the distribution ρ_0. This expectation is taken over a policy π, which is a function of the state s that outputs a distribution over the actions $\mathcal{A}$.

2.2 Hierarchical Reinforcement Learning

Given the task, hierarchical reinforcement learning aims to break down the problem into simpler subtasks which make decision making easier for the agent. We can make *macro-actions* that club a series of actions together over time. A more formal definition of this macro-action or skill is an *option* (Sutton *et al.*, 1999*b*). Options are defined by tuple of the form $O =< I, \mu, \beta >$ where I is the initiation set: where the option can begin, μ the option policy: the policy followed when the option is selected, and β the termination condition: a function of state being the probability of the option terminating. While making options adds a level of decision making, one can also think of adding several levels depending on the problem such as in Dietterich (2000) and Parr and Russell (1998). While effective in simpler domains, it gets harder to define the right hierarchy as the problem gets more complex. This has driven research in automatically creating such hierarchies which are then used in solving the problem.

2.3 Asynchronous Advantage Actor Critic

Actor critic algorithms use a parameterized policy $\pi_{\theta_a}(a|s)$ (the actor) and value function $V_{\theta_c}(s)$ (the critic) [Sutton and Barto (1998)] to estimate the policy gradient. Rather than depending on a Monte-Carlo estimate of return, the critic is used to reduce the variance in the policy gradient updates. These algorithms use the stochastic policy gradient theorem (SPGT) [Sutton *et al.* (1999*a*)] to solve the return maximimization problem. According to this theorem, the gradient of the performance objective $J(\pi_\theta)$ of a stochastic policy π with respect to the policy parameters θ is given by:

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \int_{\mathbb{S}} \rho^\pi(s) \int_{\mathbb{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s,a)\, da\, ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)] \end{aligned} \tag{2.1}$$

where the performance objective $J(\pi_\theta)$ is defined as:

$$\begin{aligned} J(\pi_\theta) &= \int_{\mathbb{S}} \rho^\pi(s) \int_{\mathbb{A}} \pi_\theta(s,a) r(s,a)\, da\, ds \\ &= \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}[r(s,a)] \end{aligned} \tag{2.2}$$

where $\rho^\pi(s)$ is the discounted state visitation distribution . The value function $V_{\theta_c}(s_t)$ is updated by using n-step TD error as: $L(\theta_c) = \left(\hat{V}(s_t) - V_{\theta_c}(s_t)\right)^2$ where $\hat{V}(s_t)$ is an n-step estimate of the return from the current state. That means $\hat{V}(s_t)$ is:

$$\hat{V}(s_t) = \sum_{j=t}^{t+n-1} \gamma^{t-j} r_j + \gamma^n V(s_{t+n})$$

In A3C, we represent policy and value functions using deep neural networks.

Asynchronous Advantage Actor Critic (A3C) [Mnih *et al.* (2016)] learns policies based on an asynchronous n-step returns. The k learner threads execute k copies of the policy asynchronously and the parameter updates are sent to a central parameter server at regular intervals. While the DQN addresses the issue of temporal correlations in its updates via Experience replay [Mnih *et al.* (2015)], these parallel workers ensure that temporal correlations are broken between subsequent updates owing to the different threads possibly exploring separate parts of the state space. The objective function for

policy improvement in A3C is:

$$L(\theta_a) = \log \pi_{\theta_a}(a_t|s_t)\,(G_t - V(s_t))$$

where G_t is an estimate for the return at time step t. One can think of $G_t - V(s_t)$ as an estimate for $A(s_t, a_t)$ which represents the advantage of taking action a_t in state s_t. This led to the name asynchronous *advantage* actor critic.

CHAPTER 3

Related Work

Rajendran *et al.* (2017) introduced a soft-attention method for transfer learning by using an adaptive convex combination of multiple task solutions in order to speed up learning in a new scenario. Its robustness to negative transfer was demonstrated mainly in part due to the addition of a base network while keeping subtask solutions fixed. It is not immediately clear how this technique will handle scaling to hierarchies.

The Option-critic architecture Bacon *et al.* (2017) deals with learning options autonomously via gradients and a option selection network. This is a significant step towards automatically learning skills but is still not a perfect way of forming them due to the general nature of the formulation. Florensa *et al.* (2017) is another work that tries to automatically learn useful, diverse skills in a continuous setting with minimal supervision.

Learning from previous knowledge is not trivial in part due to the phenomenon of catastrophic forgetting. A notable recent work addressing this issue is Kirkpatrick *et al.* (2017). Rusu *et al.* (2016) attempts to learn multiple subtasks sequentially while avoiding the pitfalls of this forgetting.

Concurrent work on Meta Learning Shared Hierarchies (Frans *et al.*, 2018) is also closely related with the major differences being its lack of fixed subtasks (and subsequent base network) and the absence of attempts to extending the hierarchies. Another closely related work is Shu *et al.* (2018) which also uses a switching network and reuses previous knowledge by maintaining the entire previous solution as their 'Base Policy'. It also allows better generalization by using natural language instructions to specify current goals.

CHAPTER 4

Hierarchical A2T

4.1 Framework

Borrowing terminology from Rajendran *et al.* (2017), let there be N fixed source tasks solutions $K_1, ..., K_N$ while we require a target task solution K_T. The source tasks are *not* learned online and must be determined apriori. We may refer to these N tasks as 'subtask solutions' or 'sub-policies'. Throughout the K_T training process, these source task solutions are held fixed. Updating them may lead to catastrophic forgetting of the previous source tasks solutions as observed empirically in . We also add one more network K_B called the base network that is trained from scratch along with the learning how to act in the target task. This base network is used to learn behaviour that, when used in tandem with the subtask solutions, will be able to solve the target task. The dashed lines in Fig.4.1 show update paths.

We can think of combining these source task solutions in an additive manner as discussed in Rajendran *et al.* (2017). While such a formulation is widely expressive, it may be the case that mixing several solutions in such a way leads to undesirable intermediate behaviour. One way of maintaining a consistency with the provided skills is by sampling from a learned distribution depending on the current state and picking one of the $N+1$ networks. Assuming we have a multinomial distribution with parameter vector $\bar{w} = \{w_1, .., w_N, w_B\}$, we have the source task as Eq. 4.1. Thus instead of computing all $N+1$ outputs at once and basing the output on a function of the entire set, the decision taken at time t only depends on one out of the $N+1$ networks.

$$K_T = K_i \qquad \text{where } i \sim \text{Multinomial}(\bar{w}) \tag{4.1}$$

Similar to Florensa *et al.* (2017), the chosen network is attended to for k timesteps where k is a tunable hyperparameter called the *commitment period* that changes based on the problem. For example, a task which requires a higher granularity in switching decisions

could be collecting fruits on an open grid while avoiding mines, whereas one where a lower granularity would work could be solving a maze.

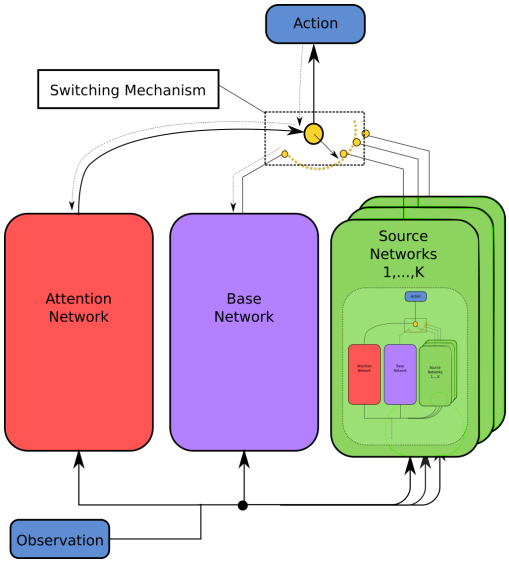

Figure 4.1: Policy Switching Model

A hard attention model is trained to learn how to switch tasks effectively. This can be thought of as a policy with the action space having the dimension $N + 1$ *i.e.* a policy $\pi_a(s)$ to pick one among the set of networks $K_1, ..., K_N$ and K_B. This model needs to estimate $\bar{w}$ given the state observation s such that $\sum_i w_i = 1$. A natural method of modeling this $\bar{w}$ is by a softmax output, thus we have the weights as in Eq.4.3.

$$f(s_t; \theta_a) = \bar{e}_s = (e_{1,s}, e_{2,s}, ..., e_{N+1,s}) \tag{4.2}$$

$$w_{i,s} = \frac{e_{i,s}}{\sum_{j=1}^{N+1} e_{j,s}}, \; i \in \{1, 2, ..., N+1\} \tag{4.3}$$

This extra step of sampling adds a layer of non-differentiability and estimators like REINFORCE (Williams, 1992) are required to learn through this. An Actor-critic approach is followed with the our Critic being an estimate of the Value function (expected return given the current state) of the entire target policy (π_T).

The updates are parallelized in a similar manner to A3C (Mnih *et al.*, 2016). Rollouts of size b are run at every time step t. These rollouts are used to form our estimates for the Advantage function and Critic. Let $d(t)$ be the last 'decision point' at time step t *i.e.* the latest point in the past the Attention network was sampled and a sub-policy was picked. For a constant k if the first time step is $t = 0$, $d(t) = t - (t \mod k)$.

$$\Delta_{\theta_a} = \sum_{t=t'}^{t'+B-1} \mathbf{1}_t(d(t)) \frac{\partial \log \pi_a(s_t, \tau_t)}{\partial \theta_a} \hat{A}_t \tag{4.4}$$

$$\mathbf{1}_t(t') = \begin{cases} 1 & t' = t \\ 0 & \text{otherwise} \end{cases}$$

Eq.4.4 is the attention network update for batch size B where τ_t is network attended to (either the Base or one of the Subtask Solutions). Thus updates to the attention network are not done at all time steps. The estimate $\hat{A}_t = \sum_{l=0}^{b-1} (\gamma)^l \delta_{t+l}$ is independently calculated for each rollout of size b. δ_t is the TD-error,

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

A critic is also maintained separately for the overall policy. The base network update follows a similar Actor-Critic approach with the update in Eq.4.5.

$$\Delta_{\theta_b} = \sum_{t=t'}^{t'+B-1} c_{s,t} \frac{\partial \log \pi_b(s_t, a_t)}{\partial \theta_b} \hat{A}_t \tag{4.5}$$

Critic updates are done via TD-learning with a n-step return target.

Encouraging exploration while learning via policy parameterization methods is often done by adding an entropy penalty. One can do the same here with the entropy term consisting of a linear combination of the attention model's (π_a) entropy and the base network's (π_b) entropy. This will ensure that the exploration occurs while following the base network as well as exploration in the policy switching sense.

$$\begin{gathered} H_\beta = \alpha_B H_B + \alpha_A H_A \\ \text{where } H_A = \mathbb{E}_\tau[-\log(\pi_a(s, \tau))],\ H_B = \mathbb{E}_a[-\log(\pi_b(s, a))] \end{gathered} \tag{4.6}$$

4.2 Updating π_b

Various possibilities exist to update this base network: i) at all times assuming on-policy updates, ii) with correction for the true off-policy nature of the returns and iii) by updating the base only when it is attended to.

1. $c_{s,t} = 1$: Assuming on-policy updates for the Base network like Rajendran *et al.* (2017) is not as safe since the Base network is not used as part of the target solution continuously. One should note that the in their work, the base network learns to copy all the parts of the subtask solutions relevant to the target task. This means that the optimal behaviour for the attention network is to eventually focus on the base network. As observed empirically, such updates are often unstable here.
2. $c_{s,t} =$ Importance Sampling based trace. Off-policy estimators like Retrace(λ)(Munos *et al.*, 2016) would learn with much less instability than (1) but the effect would be the base network copying parts of the subtask solutions entirely.
3. $c_{s,t} = \mathbf{1}_t(d(t))$: Base updates are more stable but infrequent and trajectory information is not used to the fullest extent. The base network only comes into play when required. At all times the network will use the subtask solutions when required instead of eventually shifting attention to the Base network. We use this setting in our experiments.

4.3 Choosing the value of k

The commitment period k is a hyperparameter that can be manually chosen and fixed throughout the training process. We have experimented with $k \in \{1, 3, 10\}$. Another way to set k would be to *adaptively* vary it depending on the current state. This could be achieved in a similar way to FiGAR (Sharma *et al.*, 2017). The Base network will have an added 'head' for a FiGAR policy used to select k from a set of $|k_{fig}|$ choices e.g. $k_{fig} = \{1, 3, 10\}$. Let π_k be the k selection policy. We can update this head using a similar Actor-Critic Approach as used on the Attention head, namely,

$$\Delta_{\theta_k} = \sum_{t=t'}^{t'+B-1} \mathbf{1}_t(d(t)) \frac{\partial \log \pi_k(s_t, k_t)}{\partial \theta_k} \hat{A}_t \tag{4.7}$$

4.4 Architecture

The subtask policies (source networks) are initially learned via a standard RL algorithm (here A3C) and were modeled as a simple feedforward network. While we could use

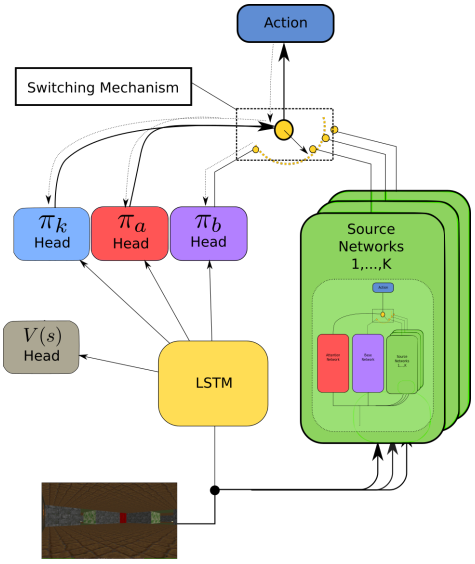

Figure 4.2: Recurrent Model

a similar feedforward network to model the Base network, one notices the possibility to utilize the memory capability of LSTMs here as the Base Policy (such as in Mnih *et al.* (2016)). By sharing this architecture with the Attention network, we turn it into a recurrent meta-controller. While the N source task solutions (or skills) used are simple state functions depending on say n stacked input images (like in Mnih *et al.* (2015)), the meta-controller (Attention network) will contain a memory component. Using a recurrent architecture each individual subtask network would lead to an immense growth in complexity unless a single multi-tasking network is used such as in Shu *et al.* (2018). This drastically reduces the complexity required compared to the subtasks being represented by recurrent networks as well. One should also note that effective use of a hierarchically structured subtask requires use of a recurrent network since the attention network needs to select it for a prolonged duration in order for it to completely carry out the subtask. To an extent this can be mitigated by using a variable k as mentioned in Section 4.3.

4.5 Summary

This architecture is meant to be used with a curricula of environments provided for learning subtasks which can be used effectively in a target task. Essentially, this curricula is expected to equip the agent with a set of skills which can help solve the target task. The Base network is expected to take the agent from partially solving the task using the

skills, to fully solving it.

Allowing a hierarchical network as one of the source tasks is potentially more expressive than a single network as a source task. This can easily be seen by looking at the analogy of a Mixture of Gaussians as being one of the skills. Expecting a feedforward network to model the each of the individual modes of the distribution along with an added network to pick between each of these modes is significantly more simplified than expecting a single network to model the entire distribution.

CHAPTER 5

Experimental Setup

The experiments aim to answer the following,

1. Does the technique perform competitively and is it a better way than using pre-training?
2. Is the base network learning to copy all relevant parts of the subtask solutions as in Rajendran *et al.* (2017)?
3. How important is the attention network in selecting between skills?
4. Is the hard attention update appropriate?
5. Is an off-policy estimator required for the Base network?
6. How useful is the Hard Attention Approach in building hierarchies?

We address (1) by running experiments with pretrained networks on our environments. By having a baseline in our experiments (Fig.6.9) with only the learnt base network we disprove (2). We attempt to solve (3) by removing the attention network and randomly sampling between the learnt base network and source networks (Fig.6.9) as well as randomly sampling the provided sub-polices (Sec.5.2.4). Solutions to (4) and (5) are demonstrated by showing convergence to a close to optimal solution in our experiments. By involving a hierarchically constructed subtask as one of the source networks (Fig. 6.6,6.5,6.11,6.14) (6) is addressed.

5.1 Gridworld Environments

Several Gridworlds[1] were used to show the effectiveness of the suggested approach at an intuitive level.

- **Puddle World:** The agent randomly starts from 4 states and must navigate through a world with a negative reward 'puddle'. It receives a +10 reward when it reaches a specified goal state and the episode ends.

[1] link to the maps on Github

- **Mine World:** The agent randomly starts from 4 states and must navigate through a world with randomly positioned (at every episode) positive and negative reward tiles. The tiles change to zero reward after being traversed upon (akin to being 'fruit being picked up' or 'mine being detonated'). There is no specified goal state and the episode ends after a fixed number of steps.
- **Room World:** This is used to run the hierarchical RL experiments. The objective is to traverse all rooms and pick the object from each one. This is challenging due to the presence of hallways restricting the correct action trajectories. The episode ends after a maximum of 1000 time steps.

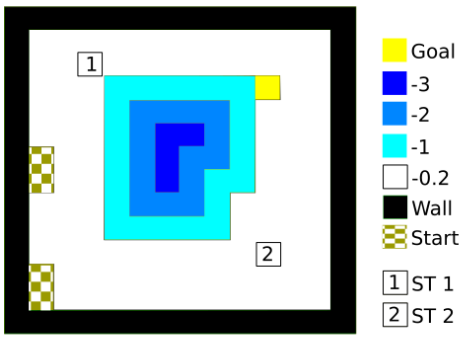

Figure 5.1: Puddle World (ST: subtask)

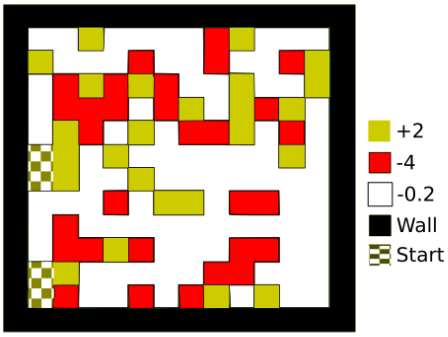

Figure 5.2: Mine World Example (Small)

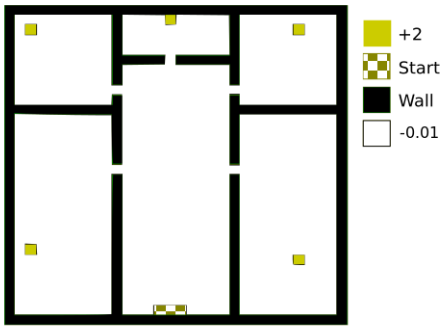

Figure 5.3: 5 room gridworld (Room World)

For the first two worlds the agent receives a 1 step view around it with the total observation size being 3x3. The Room World gave the agent a 3 step view with total size 7x7 due to its added complexity and significant partial observability by limiting the view.

The training sizes of the Puddle and Mine Worlds are 14x14 with the Mine World tests being done on a 28x28 grid. The Room World was a 32x32 sized map.

5.1.1 Architecture

Since the tasks discussed are simple, 2 layer MLPs with hidden layers of size 32 were used for the feature extraction portion without a recurrent archictecture. For the Room World, owing to the partially observable nature of the environment, we append a 32 dimension LSTM layer after the 2 layer MLP. The Room World experiments used a shared architecture with separate heads for π_b, π_a and π_k while the first two environments used separate networks.

5.1.2 Experiments

Incomplete Source Tasks: For the Puddle World, two separate subtasks were trained with the subtask goals being shown in Fig.5.1. Refer A.3 for further details on training the skills. The subtasks were used to show the benefit of the policy switching mechanisms even when the source tasks are not complete solutions.

Versus Pretraining: The Mine World experiments are done in a small training environment while results are reported on a larger test environment. A 'minesweeper' mode is also present where the fruit and mine rewards are swapped. This mode is used to show the switching mechanisms usefulness in handling multiple variations of the target task.

Scaling to hierarchies: The Room World is used to demonstrate the possibility of a hierarchical RL approach using this methodology. Three separate skills were trained via specific pretraining maps. Refer A.3.2 for details.

1. Exit Room : to transition from any point to an exit in the room.
2. Get Object: to pick up an object in the same room.
3. GO & Exit: to pick up an object then exit the room after. This is a hierarchically structured policy of the first two skills.

Intuitively these choices should be most optimal for this task.

5.2 Doom Environments

While Gridworlds helped to verify that our method worked, we wished to demonstrate its effectiveness in a more realistic setting also. This was possible with the help of Vizdoom

(Kempka *et al.*, 2016).

5.2.1 Custom Maps

We devised similar maps[2] to the gridworlds in Doom to show that our method works in the complex 3D setting as well.

1. **Collect Armour**: The agent is placed at a random location and orientation in a square room along with an armour being spawned at a random position. The agent receives a reward of $+1$ upon collecting the armour and the episode ends.
2. **Exit Room**: The agent is placed at a random location and orientation in the room with a corridor on one side. The agent receives a reward of $+1$ upon exiting the corridor and the episode ends.
3. **Collect Armour and Exit Room** (C&E): The agent is placed at the entrance to the corridor while an armour is spawned at a random position in the room. The agent receives a reward of $+0.6$ on picking up the armor $+0.4$ upon subsequently exiting the corridor and the episode ends.
4. n **rooms**: There are n rooms with an armour in each room. The agent receives $+0.5$ on picking up each armour and $+1$ for collecting the last armour followed by the episode ending. We run experiments with $n = \{3, 5\}$. Thus we consider the 3 room and 5 room maps to be solved if the score is greater than 1.5 and 2.5 respectively.

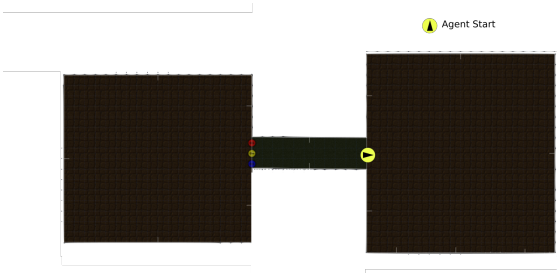

Figure 5.4: C&E Map

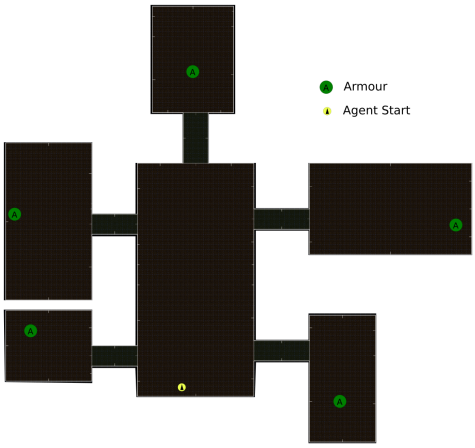

Figure 5.5: 5 Room Map

5.2.2 Common Parameters

The agent was given 4 actions, namely: No-Op, Turn Left, Turn Right, Move Forward. There is a constant time penalty of -0.001 per step and an action repetition of 4. The

[2]link to the custom maps

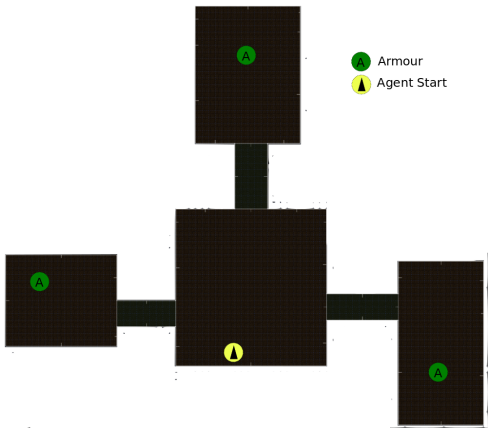

Figure 5.6: 3 Room Map

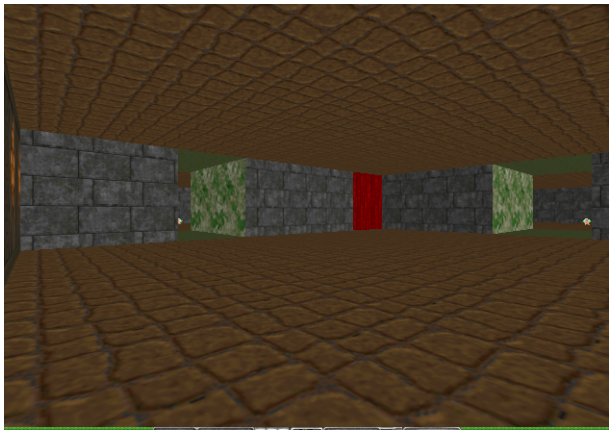

Figure 5.7: 3 Room Agent View

input RGB image of the agent's egocentric view was converted to a 42*42*1 image by taking a maximum over the channels. This was subsquently stacked with the view of three time steps into the past getting a 42*42*4 input similar to Mnih *et al.* (2016, 2015).

5.2.3 Architecture

We require function approximation for the Base Policy, Value function (critic) and Attention Network. In Mnih *et al.* (2016) the Policy Network and Value Network share a common feature representation pipeline with separate 'heads' for the Policy and Value functions. We use a similar architecture here where the three networks share the feature extraction pipeline with three separate heads. Refer A.1 for more details.

5.2.4 Experiments

All experiments had the same two subtasks given; one to collect an armour in the same room, and the other to exit the room and reach the end of the connecting corridor.

Versus Pretraining: We also provide as a baseline the performance of the most important skill ('col': Collect Armour) used as pretraining for a vanilla feed-forward network. One should note that the optimal behaviour for an agent in the n room maps without memory would be to follow the wall of the map along one particular direction until all armours are collected. Thus for a feed-forward policy, convergence would mean a lesser score than for a recurrent policy that can navigate directly to the armour.

Scaling to hierarchies: For the more complicated n room maps, we also included as one of the subtasks a hierarchically structured solution of the C&E map. Thus the optimal strategy for the agent would be to learn a base policy to navigate the connecting area between the rooms, followed by using the hierarchical subtask to pick up the armour and exit the room. Providing the two primary subtasks would help generalize to differently sized rooms.

Importance of Attention: To demonstrate that π_a is an important component of our model, we provide the average reward obtained over 100 episodes for uniformly sampling the provided sub-policies.

CHAPTER 6

Experimental Results

6.1 Grid worlds

We discuss results for the various Grid worlds below. All experiments use a feed-forward architecture unless mentioned otherwise in the legend.

6.1.1 Puddle World

Fig. 6.1 shows the episode lengths while training.

Effect of varying k Changing the commitment period k determined the duration of a skill being followed. Thus the presence of bad skills along with the entropy penalty at the switching level will cause a tendency to perform poorly at first and slowly improve. The larger the value of the fixed k often meant more unstable updates since the environment changes signifcantly after each decision step.

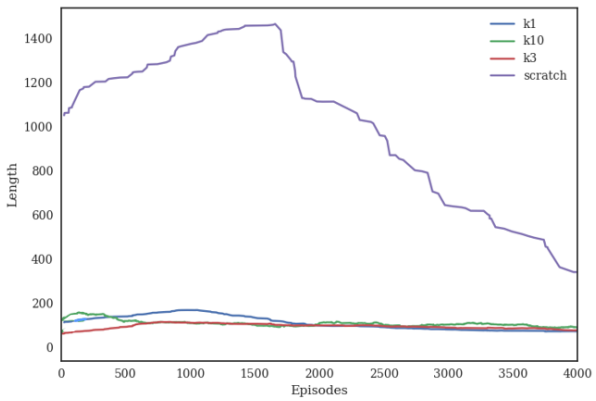

Figure 6.1: Puddle World with k varying

6.1.2 Mine World

The runs shown are:

- f : with the favorable source task (solution in small Mine World)
- u : with the unfavourable source task ('minesweeper' mode solution in small Mine World)
- uf : with both favourable and unfavourable tasks
- scratch : training from scratch on the test environment
- directu : with the unfavourable task as the initialisation (similar to pretraining)

Fig. 6.2 shows the total episode rewards. It is clear that using the set of sub-policies makes the solution robust to the conditions of the test environment.

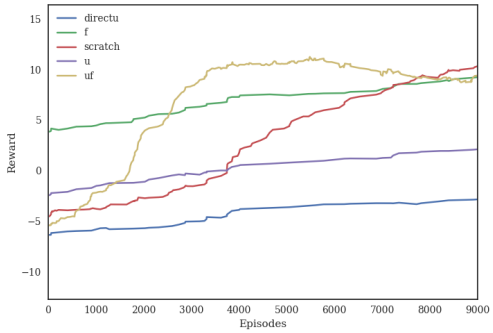

Figure 6.2: Standard Mode

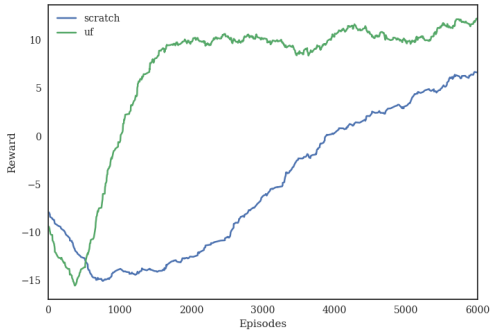

Figure 6.3: 'Minesweeper' Mode

Figure 6.4: Mine World Results

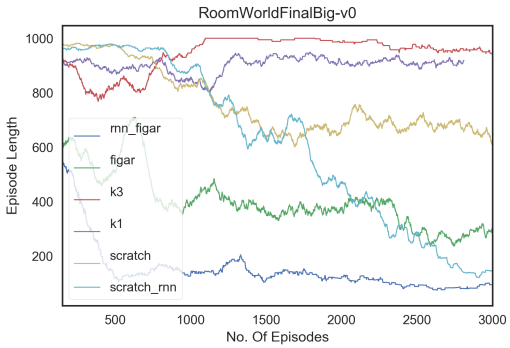

Figure 6.5: RNN + FF

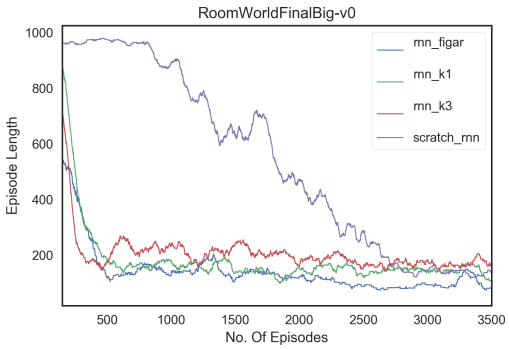

Figure 6.6: RNN

Figure 6.7: Room World experimental results
(FF:Feed forward, RNN: Recurrent

6.1.3 Room World

Scaling to hierarchies: As expected, a recurrent architecture for our controller performed better than a feedforward one. Fig. 6.6 shows the episode length for different

values of k with an RNN architecture. Fig. 6.5 shows the best results with the feed-forward network for comparison. We see that while a variable k via a FiGAR-like method helped the FF architecture by allowing prolonged sub-policy selection, it was not adequate to solve the problem.

6.2 Doom Environments

We discuss results on the Doom environments below. Curves denoted by 'stg' were provided with the hierarchically structured solution of the C&E map as a skill also.

Utility of Attention Network: The dashed lines are 100-episode averages of random selection baselines where :

- '2st' - 2 subtasks
- 'stg' - hierarchical solution of C&E map
- 'base' - trained base network of best performing network (feed-forward)

As mentioned in 5.2.4, these are to show the importance of the attention network in effectively switching between the available networks. From the random selection baselines (Figs.6.11, 6.14) we infer that the guidance by the Attention network is necessary to solve the problem.

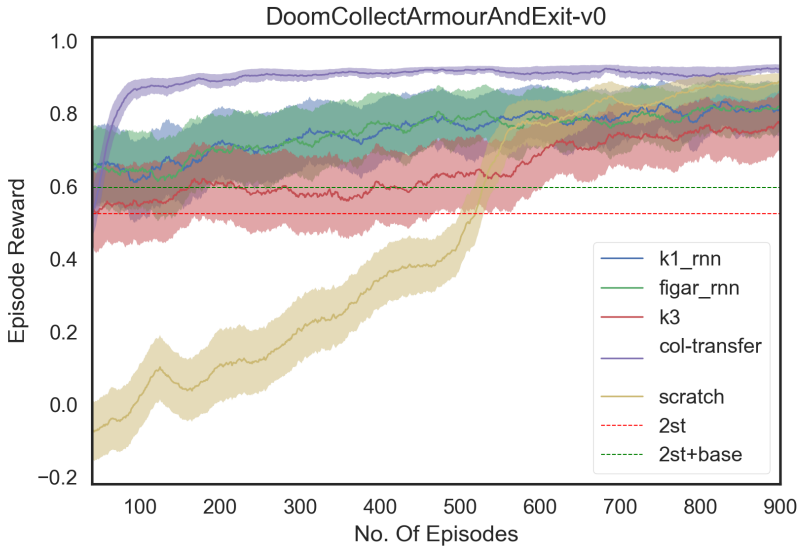

Figure 6.8: Collect & Exit (taking mean over workers, shaded portion being standard error)

Versus Pretraining: Training curves on the C&E map (Fig.6.8) shows us that for the simplest tasks, direct pretraining with the most important skill ('col' - Collect Armour) often performs comparable to our method. As the target task gets more complicated, the benefits of having access to multiple skills at once becomes apparent. This is demonstrated in the 3 room (Fig.6.9) and 5 room (Fig.6.12) experiments.

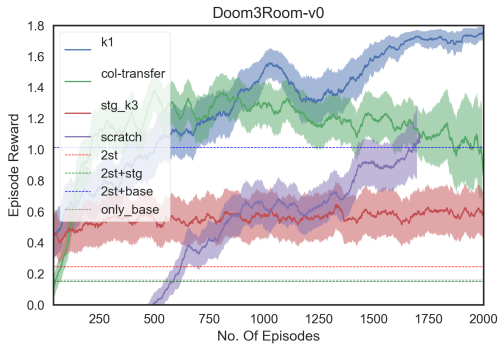

Figure 6.9: FF

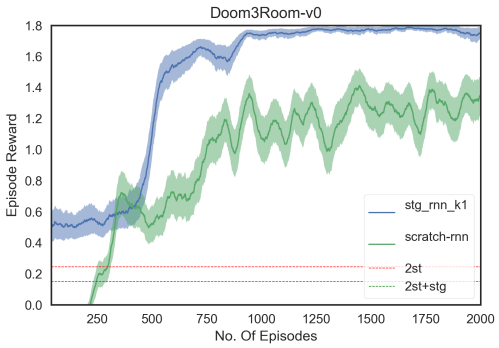

Figure 6.10: RNN

Figure 6.11: 3 room experimental results
(FF:Feed forward, RNN: Recurrent

Scaling to complex scenarios: Providing the hierarchical subtask 'stg' enabled the agent to find the optimal path quicker (Fig.6.10). We see that providing the agent with the decision capability of selecting such a hierarchical subtask for a longer duration via a recurrent architecture is crucial (check 'stg_k1' in Fig.6.12 and 'stg_rnn_k1' in Fig.6.13).

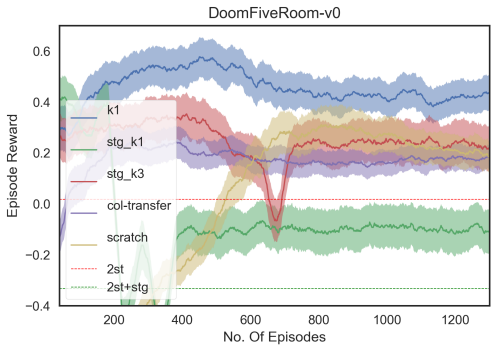

Figure 6.12: FF

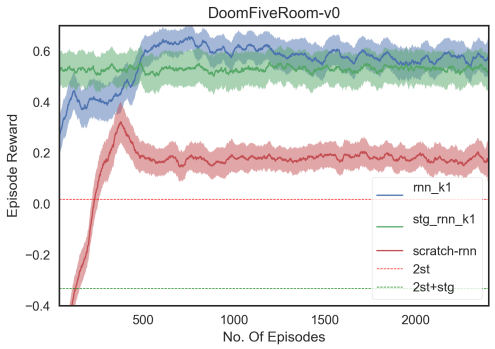

Figure 6.13: RNN

Figure 6.14: 5 room experimental results
(FF:Feed forward, RNN: Recurrent

CHAPTER 7

Shortcomings and Future work

While we have individual networks for each of the skills here, a more generalizable formulation such as in Shu *et al.* (2018) would be beneficial to compare any performance losses due to generalization. Since our action space for the attention network grows linearly with number of subtasks, a single multitasking network would address the problem of an exploding action space. Perhaps using a one hot representation as input for encoding the current goal would help achieve this.

One drawback to the A2T technique is that it does not address the issue where the domain slightly changes for a different agent view. Separating the feature extraction stage and goal directed behaviour stages to reuse high level policy behaviour is a possible area to explore.

While no Importance Sampling was used , it would be interesting to observe the gains of using it to update the base policy at all times.

Currently we expected supervision from humans to decide the order of tasks sampled from. An automated curricula provided by methods such as Svetlik *et al.* (2017) would be a worthy direction to pursue.

CHAPTER 8

Conclusion

Thus we have demonstrated the effectiveness of a policy switching network that learns to attend to the given sub-policies adaptively while learning an auxiliary base network that acts to fill in the gap in order to optimally coordinate between using the skills. We also show this framework's ability to scale to more complicated tasks in a hierarchical manner.

We believe that this provides a baseline for further work in hierarchical and deep reinforcement learning.

REFERENCES

1. **Bacon, P.-L.**, **J. Harb**, and **D. Precup**, The option-critic architecture. *In AAAI Conference on Artificial Intelligence*. 2017.

2. **Dietterich, T. G.** (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research (JAIR)*, **13**, 227–303. URL `http://dx.doi.org/10.1613/jair.639`.

3. **Florensa, C.**, **Y. Duan**, and **P. Abbeel**, Stochastic neural networks for hierarchical reinforcement learning. *In Proceedings of the International Conference on Learning Representations (ICLR)*. 2017.

4. **Frans, K.**, **J. Ho**, **X. Chen**, **P. Abbeel**, and **J. Schulman**, META LEARNING SHARED HIERARCHIES. *In International Conference on Learning Representations*. 2018. URL `https://openreview.net/forum?id=SyX0IeWAW`.

5. **Guo, X.**, **S. P. Singh**, **H. Lee**, **R. L. Lewis**, and **X. Wang** (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 3338–3346. URL `http://papers.nips.cc/paper/5421-deep-learning-for-real-time-atari-game-play-using-offline-mont`

6. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

7. **Jaderberg, M.**, **V. Mnih**, **W. M. Czarnecki**, **T. Schaul**, **J. Z. Leibo**, **D. Silver**, and **K. Kavukcuoglu** (2017). Reinforcement learning with unsupervised auxiliary tasks. *To appear in 5th International Conference on Learning Representations*.

8. **Kempka, M.**, **M. Wydmuch**, **G. Runc**, **J. Toczek**, and **W. JaÅŻkowski**, Vizdoom: A doom-based ai research platform for visual reinforcement learning. *In 2016 IEEE Conference on Computational Intelligence and Games (CIG)*. 2016.

9. **Kirkpatrick, J.**, **R. Pascanu**, **N. Rabinowitz**, **J. Veness**, **G. Desjardins**, **A. A. Rusu**, **K. Milan**, **J. Quan**, **T. Ramalho**, **A. Grabska-Barwinska**, *et al.* (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 201611835.

10. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton** (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems (NIPS)*, 1097–1105.

11. **LeCun, Y.**, **Y. Bengio**, and **G. Hinton** (2015). Deep learning. *Nature*, **521**(7553), 436–444.

12. **Lillicrap, T. P.**, **J. J. Hunt**, **A. Pritzel**, **N. Heess**, **T. Erez**, **Y. Tassa**, **D. Silver**, and **D. Wierstra** (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

13. **Mnih, V.**, **A. P. Badia**, **M. Mirza**, **A. Graves**, **T. P. Lillicrap**, **T. Harley**, **D. Silver**, and **K. Kavukcuoglu**, Asynchronous methods for deep reinforcement learning. *In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016. URL `http://jmlr.org/proceedings/papers/v48/mniha16.html`.

14. **Mnih, V.**, **K. Kavukcuoglu**, **D. Silver**, **A. A. Rusu**, **J. Veness**, **M. G. Bellemare**, **A. Graves**, **M. A. Riedmiller**, **A. Fidjeland**, **G. Ostrovski**, **S. Petersen**, **C. Beattie**, **A. Sadik**, **I. Antonoglou**, **H. King**, **D. Kumaran**, **D. Wierstra**, **S. Legg**, and **D. Hassabis** (2015). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529–533. URL `http://dx.doi.org/10.1038/nature14236`.

15. **Munos, R.**, **T. Stepleton**, **A. Harutyunyan**, and **M. Bellemare**, Safe and efficient off-policy reinforcement learning. *In Advances in Neural Information Processing Systems 29*. 2016.

16. **Parr, R.** and **S. J. Russell**, Reinforcement learning with hierarchies of machines. *In* **M. I. Jordan**, **M. J. Kearns**, and **S. A. Solla** (eds.), *Advances in Neural Information Processing Systems 10*. MIT Press, 1998, 1043–1049. URL `http://papers.nips.cc/paper/1384-reinforcement-learning-with-hierarchies-of-machines.pdf`.

17. **Puterman, M. L.**, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

18. **Rajendran, J.**, **A. S. Lakshminarayanan**, **M. M. Khapra**, **P. Prasanna**, and **B. Ravindran** (2017). Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple source tasks. *To appear in 5th International Conference on Learning Representations*.

19. **Rajendran, J.**, **A. S. Lakshminarayanan**, **M. M. Khapra**, **P. Prasanna**, and **B. Ravindran**, Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain. *In Proceedings of the International Conference on Learning Representations (ICLR)*. 2017.

20. **Rusu, A. A.**, **N. C. Rabinowitz**, **G. Desjardins**, **H. Soyer**, **J. Kirkpatrick**, **K. Kavukcuoglu**, **R. Pascanu**, and **R. Hadsell** (2016). Progressive neural networks. *CoRR*, **abs/1606.04671**.

21. **Schaul, T.**, **J. Quan**, **I. Antonoglou**, and **D. Silver** (2015). Prioritized experience replay. *4th International Conference on Learning Representations*.

22. **Schulman, J.**, **S. Levine**, **P. Moritz**, **M. I. Jordan**, and **P. Abbeel** (2015). Trust region policy optimization. *CoRR, abs/1502.05477*.

23. **Sharma, S.**, **A. S. Lakshminarayanan**, and **B. Ravindran** (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. *To appear in 5th International Conference on Learning Representations*.

24. **Shu, T.**, **C. Xiong**, and **R. Socher**, Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *In International Conference on Learning Representations*. 2018. URL `https://openreview.net/forum?id=SJJQVZW0b`.

25. **Sutton, R. S.** and **A. G. Barto** (1998). Introduction to reinforcement learning. *MIT Press*.

26. **Sutton, R. S.**, **D. A. McAllester**, **S. P. Singh**, and **Y. Mansour**, Policy gradient methods for reinforcement learning with function approximation. *In Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999*a*.

27. **Sutton, R. S.**, **D. Precup**, and **S. Singh** (1999*b*). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, **112**(1-2), 181–211.

28. **Svetlik, M.**, **M. Leonetti**, **J. Sinapov**, **R. Shah**, **N. Walker**, and **P. Stone**, Automatic curriculum graph generation for reinforcement learning agents. *In Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. San Francisco, CA, 2017. URL `http://www.cs.utexas.edu/users/ai-lab/?svetlik:aaai17`.

29. **van Hasselt, H.**, **A. Guez**, and **D. Silver**, Deep reinforcement learning with double q-learning. *In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*. 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389`.

30. **Vezhnevets, A.**, **V. Mnih**, **S. Osindero**, **A. Graves**, **O. Vinyals**, **J. Agapiou**, and **K. Kavukcuoglu**, Strategic attentive writer for learning macro-actions. *In Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016. URL `http://papers.nips.cc/paper/6414-strategic-attentive-writer-for-learning-macro-actions`.

31. **Williams, R. J.** (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, **8**(3-4), 229–256.

APPENDIX A

Training Parameters

A.1 Architectural Choices

The Doom experiments used the following architecture.

Visual Encoder: The 42x42x3 RGB image was changed to 42x42x1 by taking the maximum across the channels. This input was stacked with past frames into a 42x42xn input where n was 4. This was fed into a 4 consecutive convolutional layers (CNNs) with each having 32 filters of size 3x3 and stride 2.

FF Controller: For the feed-forward experiments, the visual encoding was flattened and fed into a 256 dimension feed-forward (fully connected) network layer.

LSTM Controller: For the LSTM experiments, the visual encoding was flattened and fed into a 256 dimension single layer LSTM network.

Attention, Base and FiGAR Policies: Depending on whether the LSTM controller was used, these were modeled by separate Fully-Connected (FC) layers given input as the flattened version of the Feature Extraction pipeline (or controller output) as the selection space ($\pi_b : |\mathcal{A}|$, $\pi_a : K$, $\pi_k : |k_{fig}|$).

Value function: The flattened version of the feature extractor (or controller output) was fed into an FC layer with a scalar output to represent $V(s)$.

A.2 Miscellaneous Stabilizing Tricks

- It was empirically observed that reducing the variance of the policy update by limiting the bootstrap horizon of the Advantage estimate ($\hat{A}_t$) to the next decision point helped in many of the experiments. This slows down learning, but involves more stable updates.
- The LSTM controller is a crucial step in getting the model to work with hierarchically structured subtasks. This is likely due to the need to decide to attend for a prolonged duration to these networks in order to completely carry out the skill.

- When subtasks solutions provided by A3C were seemingly too diffused due to learning with an entropy bonus, it benefitted us to import a greedy version of the subtask solution as a skill.

A.3 Pretraining Environments

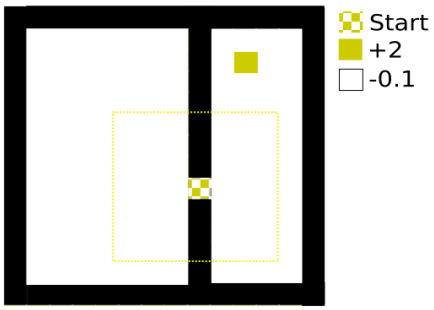

Figure A.1: Gridworld Object Map

A.3.1 Doom Skills

The skills were trained via A3C using the following maps,

1. **Collect Armour**: The agent is placed at a random location and orientation in a square room along with an armour being spawned at a random position. The agent receives a reward of $+1$ upon collecting the armour and the episode ends.
2. **Exit Room**: The agent is placed at a random location and orientation in the right room in Fig.5.4 with a corridor on one side. The agent receives a reward of $+1$ upon exiting the corridor and the episode ends.

A.3.2 Gridworld Skills

The pretraining map used is Fig. A.1 which is randomised at the start of an episode by having the wall and gap randomly placed along with the map rotating by 90°half the time. The agent view is shown by the yellow dashed box.

1. **Collect Object**: The agent spawns at the gap and receives a reward on reaching the object and the episode ends.
2. **Exit Room**: The agent is placed at a random location and orientation in the right room in Fig.A.1 with a corridor on one side. The agent receives a reward upon exiting the corridor and the episode ends.

APPENDIX B

Extended analysis of HA2T

B.1 Major Differences from A2T

The commitment period k introduces a temporal consistency to subtask selections. While A2T (Rajendran *et al.*, 2017) has a convex combination of the source task solutions, this may create some inconsistencies in combining them. For example, consider a policy that learns to move left to avoid an obsstacle versus one that moves right. A convex combination of such a policy distribution could potentially over a few time steps be in the middle, thus encountering the obstacle head on. Picking one or the other for k time steps yields a system that agrees more with either subtask solution. While a recurrent network could in practice learn this behaviour with $k = 1$, providing the network with the option to decide is beneficial (as shown in Fig. 6.5). When $k = 1$ we have,

$$\pi_T(s_t) = w_{0,t}\pi_b(s_t) + \sum_i w_{i,t}\pi_i(s_t) \tag{B.1}$$

In this case our framework is equivalent to the A2T method (Rajendran *et al.*, 2017) with our more stable base update (Eq. 4.5, Sec. A.2) enabling us to use this method for a stable update with A3C (which would not be possible otherwise).

B.2 Similarities with Options

A common way to declare extended actions in RL is by using the Option framework (Sutton *et al.*, 1999*b*). Options are defined by the tuple $< \mu, I, \beta >$. We can attempt to restate our framework in terms of the Option framework as well. The agent is given several options at every state equal in number to the sub-policy set times the number of options for k (that is, $I = \mathcal{S}$). The sub-policies avaiable to the attention network can be thought of as the option policy. The termination condition (β) is simply a function of time where it ends after k steps.

B.3 Importance Sampling to update Base Network

We show a direction to implement importance sampling.

When $k \geq 1$ with t_d the decision point for time step t (this implies $t_d \leq t$):

$$\pi_T(s_t) = w_{0,t_d}\pi_b(s_t) + \sum_i w_{i,t_d}\pi_i(s_t) \tag{B.2}$$

Note that if k is a constant, then $d(t) = t - (t \mod k)$. Now for the target network,

$$P_T(a_{d(t)}, a_{d(t)+1}, ..., a_t) = w_{0,d(t)} \prod_{r=d(t)}^{t} \pi_b(a_r|s_r)p(s_r|a_{r-1}) + \sum_i w_{i,d(t)} \prod_{r=d(t)}^{t} \pi_i(a_r|s_r)p(s_r|a_{r-1}) \tag{B.3}$$

For the base network,

$$P_b(a_{d(t)}, a_{d(t)+1}, ..., a_t) = \prod_{r=d(t)}^{t} \pi_b(a_r|s_r)p(s_r|a_{r-1}) \tag{B.4}$$

Thus the importance sampling factor for an update of trajectory from $d(t)$ to t is,

$$c_s = \frac{P_b(a_{d(t)}, a_{d(t)+1}, ..., a_t)}{P_T(a_{d(t)}, a_{d(t)+1}, ..., a_t)} \tag{B.5}$$

$$= \frac{1}{w_{0,d(t)} + \sum_i w_{i,d(t)} \prod_{r=d(t)}^{t} c_{i,s}(a_r|s_r)} \tag{B.6}$$

where $c_{i,s}(a) = \frac{\pi_i(a)}{\pi_b(a)}$. This is a naive method which we can use to scale updates on each trajectory portion.