Option Encoder: A Framework for Discovering a Policy Basis in Reinforcement Learning

Arjun Manoharan
Indian Institute of Technology Madras
arjunmanoharan2811@gmail.com

Rahul Ramesh
University of Pennsylvania
rahulram@seas.upenn.edu

Balaraman Ravindran
Indian Institute of Technology Madras
ravi@cse.iitm.ac.in

ABSTRACT

Option discovery and skill acquisition frameworks are integral to the functioning of a hierarchically organized Reinforcement learning agent. However, such techniques often yield a large number of options or skills, which can be represented succinctly by filtering out any redundant information. Such a reduction can decrease the required computation while also improving the performance on a target task. To compress an array of option policies, we attempt to find a policy basis that accurately captures the set of all options. In this work, we propose *Option Encoder*, an auto-encoder based framework with intelligently constrained weights, that helps discover a collection of basis policies. The policy basis can be used as a proxy for the original set of skills in a suitable hierarchically organized framework. We demonstrate the efficacy of our method on a collection of grid-worlds and Deepmind-lab by evaluating the obtained policy basis on downstream tasks.

KEYWORDS

Hierarchical Reinforcement Learning; Policy Distillation

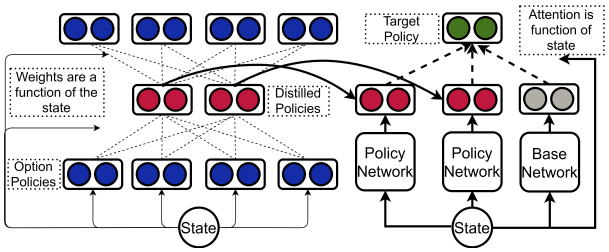

Figure 1: A visual depiction of the Option Encoder Framework. The blue colored layers correspond to the original option policies, and the red layer indicates the distilled policy. Any set of decoder weights connected to the same output sum to 1. The distilled policies are then used in a hierarchical agent that attempts to learn a policy for the target task.

1 INTRODUCTION

Reinforcement learning (RL) [25] deals with solving sequential decision-making tasks and primarily operates through a trial-and-error paradigm for learning. The increased interest in Reinforcement learning can be attributed to the powerful function approximators from Deep learning. Deep Reinforcement Learning (DRL) has managed to achieve competitive performances on some challenging high-dimensional domains [10, 16, 16, 21]. To scale to larger problems or reduce the training time drastically, one could attempt to structure the agent in a hierarchical fashion. The agent hence makes decisions based on abstract state and action spaces, which helps reduce the complexity of the problem. One popular realization of hierarchies is through the options framework [26] which formalizes the notion of a temporally extended sequence of actions.

Discovery of options, particularly in a task agnostic manner often leads to a large number of options. Option discovery methods [13, 14, 22–24] as a result, typically resort to heuristics that help prune this set. In such a scenario, a compression algorithm is of utility, since it would be wasteful to discard these options, but ineffective to use all of them simultaneously. Firstly, the computation expended for determining each option policy is higher, when compared to the smaller set of basis policies. Secondly, we show that a reduced set of basis policies result in better empirical performance on a collection of target tasks. This can be attributed to the fact that the algorithm dedicates less time for determining the relevance of each option to the current task.

Resorting to an existing policy distillation or compression method [6, 7, 17] is one possible alternative. However, these methods distill the options into a single network, resulting in a single policy that captures the behavior of all policies. However, one would like to distill multiple options into a minimal set of policies. In this work, we propose the *Option Encoder*, a framework that attempts to find a suitable collection of basis policies, based on the discovered set of options. We use an auto-encoder based model where an intermediate hidden layer is interpreted as a set of basis policies which we term as *distilled policies*. The options are mixed and are reconstructed back using attention weights. The intermediate hidden layers would hence be forced to capture the commonalities between the various options and potentially eliminate redundancies. The overview of this framework is summarized in Figure 1. The obtained distilled policies can be used to solve a new set of tasks and can be used as a proxy for the original set of options in a new algorithm. This procedure can be understood as a framework that precedes exploiting a collection of policies to transfer to a target task.

Our work also provides a simple mechanism to combine options obtained from different option discovery techniques. This is similar in spirit to [3] but we do not combine the options in a goal-directed manner. Generating options for a certain goal is useful in some scenarios but is difficult to work with in a multi-task setting. Using a task-agnostic approach (like in the Option Encoder) allows the options to be reusable across different tasks. Furthermore, the Option Encoder discards the full set of expert options after distilling to a smaller set, which is not the case in [3]. Another potential extension of this work is to the continual learning framework, where the Option Encoder can be used to handle a set of new policies and integrate the same with policies learnt earlier.

Our contributions in this work are as follows. We describe the Option Encoder framework, which finds a 'basis" for a set of options by compressing them into a smaller set. We empirically demonstrate that the Option Encoder helps improve or retain the performance on downstream tasks when compared to using the raw set of options or when using a reduced number of options. We show results on a few challenging grid-worlds where we achieve a 5-fold and 10-fold reduction in the number of options while retaining or even improving the performance. We also show results in the high-dimensional visual navigation domain of Deepmind-lab.

2 PRELIMINARIES

RL deals with sequential decision making problems and addresses the interaction of an agent with an environment. It is traditionally modeled by a Markov Decision Process (MDP) [18], defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, where $\mathcal{S}$ defines the set of states, $\mathcal{A}$ the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}(S)$ the transition function (that maps to a probability distribution over states), r: $\mathcal{S} \times \mathcal{S}' \times \mathcal{A} \rightarrow \mathcal{R}$ the reward function and γ the discount factor. A parameterized policy $\pi(a|s;\theta)$ maps every state to a probability distribution over the actions. In the context of optimal control, the objective is to learn a policy that maximizes the expected discounted return $R_t = \sum_{i=t}^{T} \mathbb{E}\left[\gamma^{(i-t)} r(s_i, s_{i+1}, a_i) \right]$, where $r(s_i, s_{i+1}, a_i)$ is the reward function. Policy gradient methods [25] attempt to find a parameterized policy that maximizes the expected discounted return. Some popular policy gradient methods include Advantage Actor-critic (A2C) [8] and Proximal Policy Optimization (PPO) [20].

Options: An Option [26] formalizes the notion of a temporally extended sequence of actions and is denoted by the tuple $\langle \mathcal{I}, \beta, \pi_o \rangle$. $\mathcal{I} \subseteq \mathcal{S}$ denotes the initiation set of the option, $\beta : \mathcal{S} \rightarrow [0, 1]$ is the probability that the option terminates in state s and π_o is the option policy. In this work, we assume that the initiation set is the set of all states.

Attend, Adapt and Transfer architecture (A2T):. A2T [19] is a model for utilizing expert policies (or options) from N different source tasks in order to tackle a target task. Consider N expert policies, represented by $\{K_i(s)\}_{i=1}^{N}$. Apart from the expert networks (which are fixed throughout training), A2T has a trainable *base network* represented by $K_B(s)$, which is used to learn in regions of the state space, where the set of experts do not suffice. The target policy $K_T(s)$ is given by:

$$K_T(s) = w_{N+1}(s)K_B(s) + \sum_{i=1}^{N} w_i(s)K_i(s) \tag{1}$$

The set of weights $w_j(s)$ (for $j \in \{1 \cdots N+1\}$) are attention weights and as a consequence, satisfy the constraint $\sum_{i=1}^{N+1} w_j(s) = 1$. $K_T(s)$ is a convex combination of $N + 1$ policies and is hence also a valid policy.

In this work we use a modified version of the A2T algorithm, identical to the version in [5]. Instead of combining the option policies using the attention weights, the A2T agent instead selects a single option policy and persists the same for T steps. We refer to the persistence length T as the *termination limit*. Henceforth, any reference to A2T refers to the modified version.

Actor-Mimic Network (AMN):. Given a set of source tasks, the Actor-mimic framework [17] attempts to learn a network that copies the policies of the various experts. The reconstruction loss corresponding to task i, is given by:

$$\mathcal{L}^{i}_{policy} = \sum_{a \in \mathcal{A}_i} \pi_{E_i}(a|s) \log \pi_{AMN}(a|s;\theta) \tag{2}$$

π_{AMN} is a parameterized network that is trained using the cross-entropy loss. The targets are generated from the expert policy π_{E_i}. In the case where the expert consists of Q-values, the targets are generated from the Boltzmann distribution controlled by a temperature parameter τ. [17] uses an additional feature regression loss which we omit in this work.

3 OPTION ENCODER FRAMEWORK

The Option Encoder attempts to find a collection of basis policies, that can accurately characterize a collection of option policies. Let the policy of an option j, be denoted by π_j. Let the i^{th} policy of the distilled set (intermediate layer) be represented by π_i^d. The set of M "distilled" policies $\pi^d = \{\pi_i^d\}_{i=1}^{M}$ are found by minimizing the objective given in Equation 3.

$$\pi^{d^*} = \arg\min_{\pi^d} \min_{W} \sum_{s \in \mathcal{S}} \sum_{j} \mathcal{L}\left(\pi_j(s), \left(\sum_i w_{ij}(s) \times \pi_i^d(s)\right)\right) \tag{3}$$

W is a weight matrix with the entry in i^{th} row and j^{th} column denoted by w_{ij}. The matrix W is such that $\sum_i w_{ij} = 1 \; \forall j$, i.e the rows of the matrix are attention weights. W can also be a function of the state s. Each w_{ij} is a scalar such that $0 \leq w_{ij} \leq 1$ and it indicates the contribution of distilled element $\pi_i^d(s)$ to the reconstruction of option policy $\pi_j(s)$. The function $\mathcal{L}$ is a distance measure between the two probability distributions (for example Kullback-Leibler divergence or Huber Loss). The objective states that a convex combination of the distilled policies should be capable of reconstructing each of the original set of options with minimal loss.

Equation 3 is realized using an auto-encoder. The encoder is any suitable neural network architecture that outputs M different policies. For example, an encoder in a task with a discrete action space will consist of M different softmax outputs each of size $|\mathcal{A}|$ (size of action space). A continuous action space problem will contain M sets of policy parameters (for example, the mean and variance of a Gaussian distribution). Since the distilled policies are combined linearly, a single layer for the decoder should suffice, since the addition of more layers will not add any more representational power. The entire procedure is summarized in Algorithm 1.

The architecture is an auto-encoder with two key constraints (see Figure 1). The first restriction is that each distilled policy has a single shared weight, i.e., every action of a single policy will have the same weight. This makes the re-constructed expert policies to be convex combinations of the distilled policies.

The second restriction is that the set of weights responsible for reconstructing any option policy must sum to 1. These weights are attention weights and can be agnostic to the current state or be an arbitrary function of it. These restrictions are imposed to respect the objective specified in Equation 3. This constraint ensures that the

distilled policies are coherent since a heavily parameterized auto-encoder will result in a non-interpretable set of distilled policies.

For example, consider a scenario in which all the option policies indicate that the action a, has the highest preference in state s. Let action a be assigned the least probability after passing the options through an encoder. If the weights are allowed to take arbitrary values on the decoder side, the distilled policies can be capable of reconstructing the options by assigning higher weights to action a, even though it has a low probability as per the distilled policies. Alternately, the decoder can make use of negative weights to flip the preference order over the actions dictated by the distilled policies.

Ideally, one would want the distilled policies to capture the fact that action a is preferred in-state s among all options. Hence, the proposed two restrictions ensure that this intended behaviour is achieved. The second restriction also ensures that the output of the decoders are also valid policies since a weighted combination of the policies (with the weights summing to 1) will also result in another policy.

The current framework would require all the option policies to be run to obtain distilled policies. This would, however, defeat the entire purpose of the distillation procedure since we would like to discard the options after the distillation. Hence, the distilled policies can be considered as targets for a procedure like Actor-mimic [17]. Each distilled policy can be potentially transferred to a smaller network using a supervised learning procedure. This can be used during decision-time planning or for policy execution since the network is lightweight by design. The entire procedure is summarized in Algorithm 1.

Algorithm 1: Summary of the Option Encoder Framework

```
L = Number of steps for rollout ;
N = Number of Option policies ;
M = Number of Distilled Policies ;
K = Number of Target Goals ;
T = Number of Steps upto which a option is executed ;
E = Number of Episodes ;
Dataset = Empty list ;
for j in (1..... N) do
    env.reset() ;
    for i in (1..... L) do
        Get option policies (π_1(s), π_2(s), ··· π_N(s)) ;
        Add (s, (π_1(s), π_2(s), ··· π_N(s))) to Dataset;
        a = Sample(π_j(s)) ;
        s = env.step(a) ;
    end
end
DistillPolicies = train_auto-encoder(Dataset) ;
for j in (1..... M) do
    DistillDataset = None ;
    for s in Dataset[0, :] do
        π̂_j(s) = DistillPolicies(j, s) ;
        add π̂_j(s) to DistillDataset ;
    end
    π_j = ActorMimic(DistillDataset) ;
end
for i in (1....K) do
    for j in (1....E) do
        env.reset() ;
        while not done do
            Option_id = AttentionNetwork.Sample(s) ;
            for t in(1...T) do
                a = Option_id.Sample(s) ;
                s = env.step(a) ;
                store_transitions();
            end
            collect_rollout() ;
            UpdateAttentionNetwork(rollout) ;
            UpdateBaseNetwork(rollout)
        end
    end
end
```

4 EXPERIMENTS

In this section, we attempt to answer the following questions:

- How do the distilled policies compare against the option policies on a set of tasks?
- Why do we need certain restrictions on the architecture?
- Is the performance gain solely because of a reduced number of policies?
- Does varying the number of distilled policies effect the performance?
- Does the length of temporal persistence in the hierarchical agent, affect the performance?

4.1 Task description

Grid-world: We consider the grid-worlds depicted in Figure 2. The grid-worlds are stochastic where the agent moves in the intended direction with probability 0.8 and takes a random action (uniform probability) otherwise. The environment has 5 actions available from every state, which are up, down, left and right and a terminate action. Each episode terminates upon taking the terminate action or after 3000 environment steps have been completed. We consider a task where the agent gets a reward of +1 on reaching the designated goal and a reward of 0 for every other transition. Fifty options were learned using the Eigen-options framework [11] for each grid-world which were then used to solve the task of reaching 100 randomly selected goals. These goals are denoted by the yellow dots on the grid-world in Figure 2. Three different grid-worlds GW1, GW2, and GW3 (left to right in Figure 2) were considered. GW1 and GW2 are of sizes 28x31 each and GW3 is of size 41x41.

Deepmind Lab: The Deepmind-Lab domain [4] is a visual maze navigation task where the inputs are images. The images are converted to grayscale before being used. The action space is discretized to 4 actions which are forward, backward, rotate-left and rotate right. Every step receives a reward of -0.1 and the episode ends after reaching a goal or after 3000 steps. Upon reaching the goal

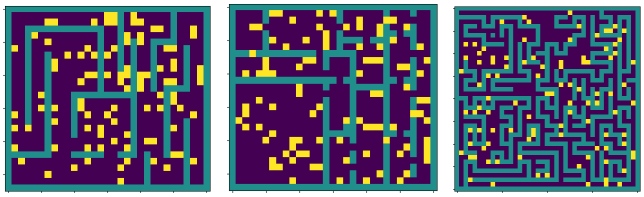

Figure 2: The 3 grid-worlds are tackled in this work. The yellow dots indicate the goals for a collection of tasks that we attempt to solve for in the grids GW1,GW2,GW3 in order.

(collecting apple), the agent receives a reward of 2.0 and receives a reward of -1.0 if it fails to reach the goal during an episode.

4.2 Architecture

Option Encoder: The state in all grid-world experiments is represented as an image of the grid (with depth 1) with all zeros, except at the location of the agent. We impose the encoder to also have shared attention weights (each policy has a single attention weight) like the decoder. This implies that the original set of options are combined using attention weights to yield the distilled policies which are then combined using another set of attention weights to yield the reconstructions.

The encoder and the decoder are comprised of attention weights which are functions of the current state. The state-based attention network consists of two convolution layers (4 filters of size 5x5 and stride 2 and 8 filters of size 3x3 and stride 1) and the fully connected layer with 32 units. These units are used to yield M sets of attention weights (total of $M \times N$ weights) for the encoder and N sets of attention weights (total of $N \times M$ weights) for the decoder, where M is the number of option policies and N is the number of distilled policies. All the option policies corresponding to a given state are fed to the auto-encoder, with its reconstruction guided by the Huber-loss. RMSProp with a learning rate of 10^{-4} and batch size of 4 was used. For the Deepmind-Lab task, the current state is converted into attention weights using a network that contains 3 convolution layers (6x6x8, 4x4x16 and 3x3x32) followed fully connected layer of size 256, an LSTM layer of size 256 which then outputs the attention weights.

Hierarchical Agent: The A2C algorithm with the modified A2T framework (described in Section 2) was used to train the agent (referred to as the A2T + A2C agent). The agent has a base network and is augmented by a collection of option policies. The base network consists of 2 convolution layers (same configuration as earlier) followed by a fully connected layer of size 128 which is projected to yield the policy and the value function heads. The base network policy and the option policies are combined using an attention mechanism to yield the final policy.

The attention network has 2 convolution layers (identical to ones described above) which are followed by a fully connected layer of size 128. These 128 units are projected to obtain the attention weights over the original/distilled option policies and the base network policy. The agent follows a hard-attention mechanism where the selected option policy is persisted for T=20 steps (termination limit is 20). For the Deepmind-lab task, the base agent has

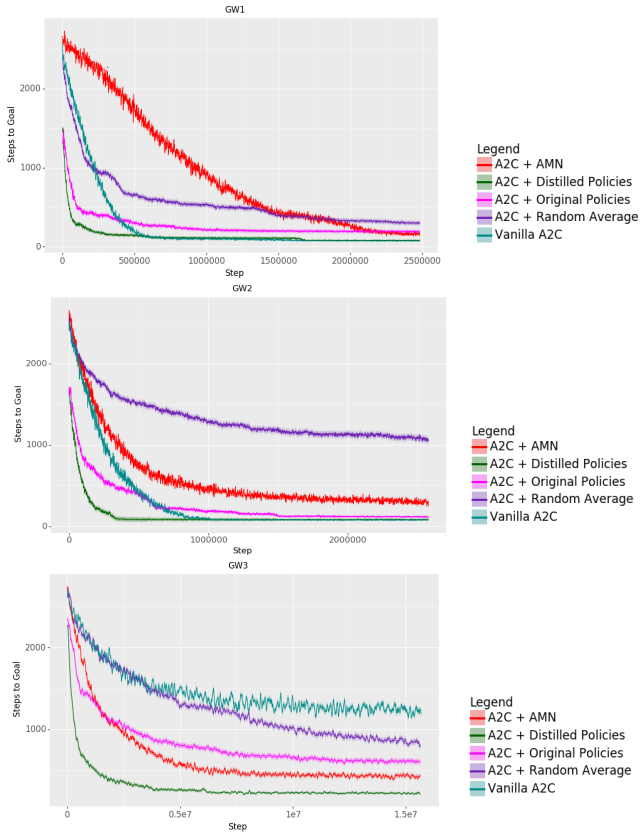

Figure 3: Performance graphs of number of steps to goal (measure of performance) vs. environment steps (training samples) on GW1, GW2, GW3 respectively (left to right). The grpahs compare the distilled policies against the original option policies, a random sample of the option policies, AMN and a vanilla A2C agent.

4 convolution layers of size 3x3x32 followed by a fully connected layer of size 256 which is projected to yield the policy and the value function. A2T + A2C attention network over the option/distilled policies uses the same architecture and the attention weights are optimized using PPO (since the layer is non-differentiable). The option policy is persisted for T=5 steps (termination limit is 5).

4.3 Evaluating on Grid-worlds

50 options were obtained using the Eigen-options framework [11] where the eigen-vectors of the graph laplacian are used to define options. The policies corresponding to each eigen-vector is obtained using a vanilla A2C agent (architecture identical to an A2C+A2T agent barring the attention). This is followed by the Option Encoder framework, which distils the options policies obtained from all states in the grid-world, into 5 distilled policies. The A2C+A2T agent can make use of the original set of options or the distilled set, which we term as A2C + original and A2C + distilled respectively. We also evaluate a vanilla A2C agent. Actor-mimic is another baseline that we consider, where all the options are distilled to one policy which is used in the A2C + A2T setup (we refer to this as

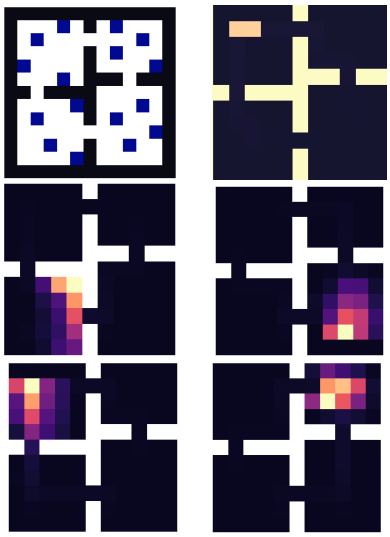

Figure 4: The top-left figure denotes the set of 16 expert policies while the other 4 figures in the middle and bottom row visualize the visitation distribution of the distilled policy. The top-right plot in the top-most row corresponds to the visitation distribution of AMN policy.

A2C+AMN). Finally, we consider the random average agent which consists of an A2T+A2C agent attending to 5 random selected option policies and a base network. Each of the policies are obtained by averaging the policies of 10 randomly chosen option policies.

The agents are periodically evaluated every 500 environment steps and the performance curves are presented in Figure 3. The graphs are clearly indicative of the fact that the A2C+distilled agent outperforms all other baselines. Since we tackle 100 different target tasks, the effort required to obtain the distilled policies (or the options) is negligible when compared to solving the multi-task problem. Hence, the presented performance curves are comparable. We also vary the number of experts and distil to 5 options (Figure 5) and notice that an increased number of experts improve the performance. We also compare to a policy obtained using k-means on the policy space (in Figure 5). In particular, the option policies were clustered and the centroid of each k-means cluster was used as a distilled policy. We observe that our model outperforms this baseline too indicating that the distillation of the experts must be performed intelligently.

The observation that Option Encoder exploits information from a large number of experts is further corroborated by Figure 6. The figure indicates that the top 10 Eigen-options performs worse when compared to using a set of 10 distilled options. In a resource constrained scenario where a limited number of options are required, one can distill knowledge from a plethora of options, to a smaller set.

4.4 Understanding Architecture Constraints

We enforce certain restrictions on the auto-encoder that ensures that the reconstructed policy (and optionally the distilled policy) are convex combinations of the policies in the previous layers. We conduct a qualitative analysis of different architectural variants. Remember that the Option Encoder architectures require the weights to be attention weights and that a policy is a convex combination of the policies from the previous layers. This implies that all the actions of a policy share a single weight.

We consider the grid-world in Figure 7 with four expert options going to the 4 corners of the top-left room. Figure 7 represents a heatmap of the distribution of states visited by the 2 distilled policies generated from the Option Encoder framework. We visualize the same heatmap for other architectures:

- *Removing restriction on attention weights*: In this case, the weights are not fed through a softmax layer to enforce that they sum to 1. Figure 8 highlights that we do not learn an interpretable policy.
- *Removing restriction on weight sharing*: Since the layers enforce the output policies to be convex combination of the input policies, each policy is assigned a single attention weight. We instead assign a single weight for every action in the policy increasing the number of weights by $|\mathcal{A}|$ (size of the action space). Figure 9 again highlights that we do not learn qualitatively useful policies.
- *Removing both restrictions*: This case corresponds to a vanilla Auto-encoder with no constraints on the weights. An intermediate layer is interpreted as a policy and is found to not be qualitatively interpretable (see Figure 10).

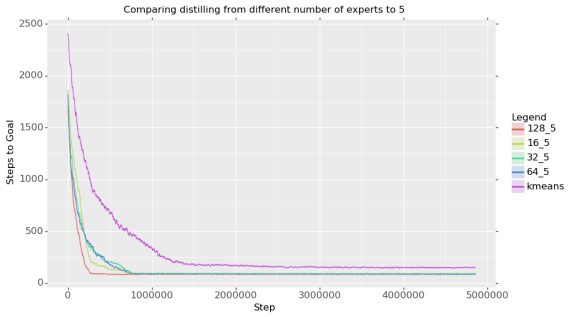

Figure 5: Distilling varying number of experts to 5 options

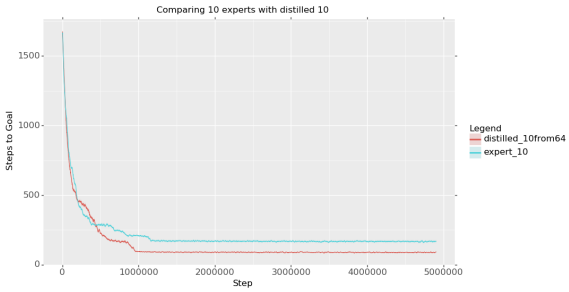

Figure 6: Comparing 10 Eigen-options to 10 options obtained from distilling 64 Eigen-options

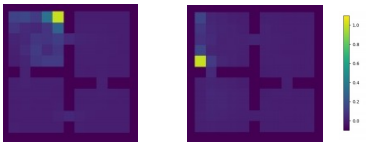

Figure 7: A heatmap of the visitation distribution of the two distilled policies from the Option Encoder framework

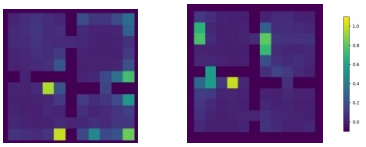

Figure 8: A heatmap of the visitation distribution of the two distilled policies after removing restriction on attention weights

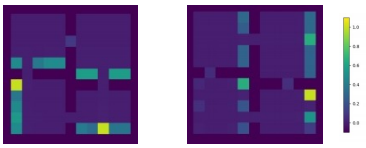

Figure 9: A heatmap of the visitation distribution after removing restriction on weight-sharing for policies

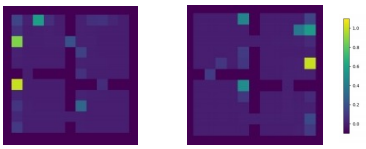

Figure 10: A heatmap of the visitation distribution after removing restrictions on both attention and weight-sharing

.

4.5 Understanding the distilled policies

To understand the Option Encoder framework, we consider 16 policies, each navigating to a specific goal as indicated in the top-left image in Figure 4. Each blue cell denotes a goal towards which the expert is navigated to optimally. These experts are distilled to 4 different policies. In order to visualize these policies, we develop a heatmap of the visitation counts for each policy. The heatmap is obtained by sampling from the respective distilled policy. The agent is executed for 50,000 steps and is reset to a random start state after 100 steps in the environment. Figure 4 demonstrates how the Option Encoder framework captures the commonalities between various policies while the AMN network is unable to capture the diversity in a single policy.

4.6 Randomly sampling a set of options

This analysis on GW2 was conducted to verify that our technique does not derive any advantage, solely based on a reduced number of option policies. Ten sets of options five in each set were considered, and each option appears in any one of the ten sets. Figure 11

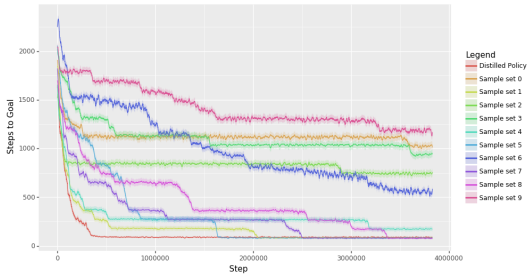

Figure 11: Evaluating a random subsets of options to show that the performance improvement is not due to a reduced number of options

shows that a random sample of options can lead to vastly varying performances (based on the relevance of the options). However, the distilled policies outperform every displayed sample of options, hinting at the fact that all the options are useful on solving a new set of tasks. Each line corresponds to the average performance over *ten randomly selected goals.*

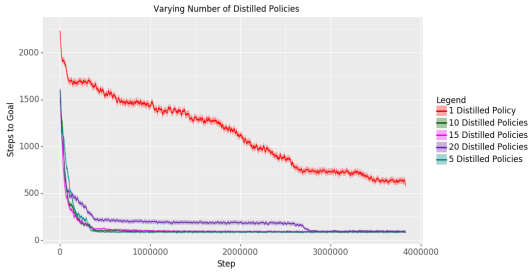

Figure 12: Performance plotted for a varying number of distilled policies.

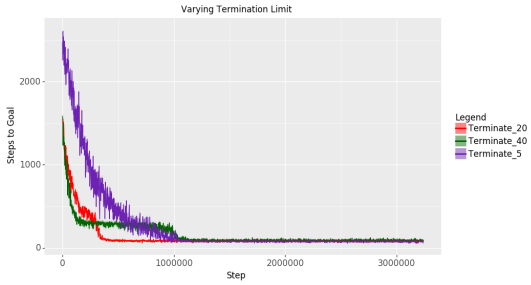

Figure 13: Varying the termination limit of the A2T + A2C setup.

4.7 Varying the number of Distilled policies

An experiment on GW2, was conducted (see Figure 12) to identify the impact of the number of distilled policies on the final performance. The plots are indicative of the fact that the agent is not highly sensitive to this parameter. Distilling to a single policy yield

a poor performance curve since a single policy is incapable of capturing the variety in the various options. Fifty expert policies were used.

4.8 Varying termination limit

We analyze the impact of termination limit T (defined in Section 2) and obtained the performance curves for GW2. When the termination limit is low, the agent fails to leverage the knowledge of a useful sequence of actions since it cycles between the various options. Hence, a higher termination limit results in a persistent strategy for an extended duration, thereby yielding better performance (see Figure 13). However, beyond a certain value for the termination limit, the performance deteriorates, owing to the fact that the agent spends an excessive amount of time on a single option.

4.9 The Deepmind-lab task

An I-room domain (Figure 14) was used for this task. Four option policies were trained using A3C [15] to navigate to four corners in one of the rooms. The agent can start from anywhere in the lower half of the domain. We observe that one distilled option learns to navigate to right corners in the room while the other navigates to the left bottom corner. The target task is to navigate to an unseen location in the maze. The performance was averaged over 6 goals selected from the top room. The final graph (Figure 15) is obtained by repeating the same experiment three times. The figure indicates that there is no degradation in the transfer performance after using the Option Encoder that reduces 4 option policies to 2 with a termination limit of 4. We visualize the options and to get a qualitative understanding of the compression achieved by the Option Encoder. [1]

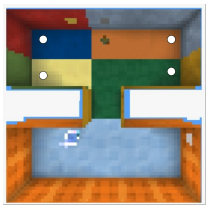

Figure 14: Map used for Deepmind-lab. The white dots are sub-goals for option policies

5 RELATED WORK

Several works have attempted to address the policy distillation and compression scenario. Prior works like Actor-mimic [17] have attempted to compress a collection of policies. This framework can, however, only distill a collection of policies into one single policy. Our method, on the other hand, can distill the expert policies into multiple basis policies. Pathnet [6] utilizes a large network, with weights being frozen appropriately. However, like Actor-mimic, Pathnet only discovers a single policy which may not have sufficient representational power. Pathnet addresses the continual learning

[1] https://www.dropbox.com/sh/soenidh9puhgrq6/AAAj1A91JyPWZax6JgGP7Qhva?dl=0

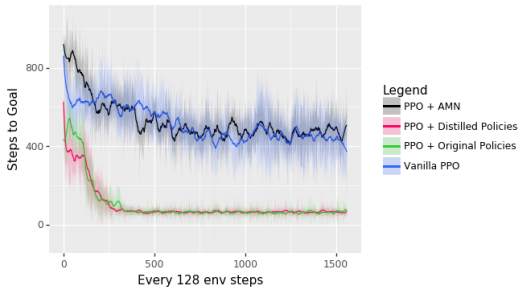

Figure 15: Performance curve for the Deepmind Lab target task. Plot of performance (steps to goal) against the number of samples from the environment.

setup, where there is a sequence of tasks to be solved, which may not be an appropriate approach to option pruning. Overcoming catastrophic forgetting [7], address a scenario identical to that addressed in Pathnet [6] by modifying the gradients in an appropriate manner but suffers from same problems described for the other two methods.

Option pruning has not been addressed in the context of option-discovery in great detail. Option compression is not necessary for works like Option-critic [2], Deep Feudal reinforcement learning [27] or a collection of other works that discover options relevant to the current task. On the other hand, task-agnostic option discovery methods often need a large number of options to capture diverse behaviors. McGovern and Barto use a collection of filters to eliminate redundant options and Machado et al. have observed an improved performance when around 128 options are used in a rather modest 4-room grid-world. Basis functions have been explored in the context of value functions [9, 12], where the structure of the graph is used to define features for every state. On the other hand, our work uses a set of options to define the basis and over policies. Our work shares similarities with the PG-ELLA [1] lifelong learning framework, in that both attempt to discover a shared latent space, from which a set of tasks are solved. In this work, we focus on multi-task learning, where the basis policies are utilized across a set of tasks.

6 CONCLUSION

In this work, we present the Option Encoder framework, which attempts to derive a policy basis from a collection of option policies. The distilled policies can be used as a substitute for the original set of options. We demonstrate the utility of the distilled policies using an empirical evaluation on a collection of tasks. As future work, one could extend the framework to work with value functions. Another area worth exploring is the continual learning setup. This could be set up in a batch-like manner, where the set of basic policies can be constantly refined. The robust learning setup is also an area worth pursuing. The various distilled policies could help generalize across multiple environment configurations. Lastly, this work doesn't attempt to re-orient the action space when compressing policies. One could look to compress policies while also accounting for isomorphism obtained from permuting the action space.

REFERENCES

[1] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. 2014. Online multi-task learning for policy gradient methods. In *International Conference on Machine Learning*. 1206–1214.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The Option-Critic Architecture.

[3] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. 2019. The Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 13031–13041.

[4] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801* (2016).

[5] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070* (2018).

[6] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A Rusu, Alexander Pritzel, and Daan Wierstra. 2017. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734* (2017).

[7] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[8] Vijay R Konda and John N Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[9] George Konidaris, Sarah Osentoski, and Philip Thomas. 2011. Value function approximation in reinforcement learning using the Fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[11] Marlos C Machado, Marc G Bellemare, and Michael Bowling. 2017. A Laplacian Framework for Option Discovery in Reinforcement Learning. *arXiv preprint arXiv:1703.00956* (2017).

[12] Sridhar Mahadevan and Mauro Maggioni. 2007. Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. *Journal of Machine Learning Research* 8, Oct (2007), 2169–2231.

[13] Amy McGovern and Andrew G Barto. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. (2001).

[14] Ishai Menache, Shie Mannor, and Nahum Shimkin. 2002. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *European Conference on Machine Learning*. Springer, 295–306.

[15] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature* 518, 7540 (2015), 529.

[17] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342* (2015).

[18] Martin L Puterman. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. (1994).

[19] Janarthanan Rajendran, Aravind S Lakshminarayanan, Mitesh M Khapra, P Prasanna, and Balaraman Ravindran. 2015. Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain. *arXiv preprint arXiv:1510.02879* (2015).

[20] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[21] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (Jan. 2016), 484–489. https://doi.org/10.1038/nature16961

[22] Özgür Şimşek and Andrew G Barto. 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 95.

[23] Özgür Şimşek and Andrew G Barto. 2009. Skill characterization based on betweenness. In *Advances in neural information processing systems*. 1497–1504.

[24] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. 2005. Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. ACM Press, 816–823. https://doi.org/10.1145/1102351.1102454

[25] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. MIT press.

[26] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.

[27] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. FeUdal Networks for Hierarchical Reinforcement Learning. *arXiv:1703.01161 [cs]* (March 2017). arXiv:cs/1703.01161