Multi-policy Attention as a Mechanism for Unsupervised Heirchical Learning

Adam Klein [*1]

Abstract

Traditional reinforcement learning (RL) algorithms often struggle with tasks that require complex planning and forward-thinking. Hierarchical Reinforcement Learning (HRL) tackles this issue by adding layers of multi-action 'choices' over the usual single-step actions. HRL algorithms have the potential to be more powerful, generalized, and explainable and regular RL methods. Unfortunately, most existing HRL implementations require expert-crafted hierarchical reward structures or labelled trajectory information. In this paper, we introduce AttenPi: a fully differentiable gradient-descent method that utilizes attention to learn hierarchical decision making completely unsupervised.

1. Introduction

Reinforcement Learning has made huge strides in training models to perform short-term 'atomic' tasks, such as low-level motor control. Unfortunately, long-horizon tasks requiring high-level reasoning is still very difficult with classical frameworks (1).

Hierarchical reinforcement learning establishes methods for creating agents that can reason on higher abstraction levels, thinking ahead to achieve long-term goals, rather than simply choosing an action at each step. This is done by using a 'hierarchy' of policies, where higher level policies make decisions with decreasing frequency, and lower level policies act according to the higher-level decisions (2). However, with increasing levels of temporal abstraction come increased training difficulty, and true hierarchical learning is rarely done in practice without the input of a human expert.

This paper introduces AttenPi, a method for creating high level policies without supervision. AttenPi operates by training multiple policies simultaneously, and using a higher level decision maker to choose which policy will be applied for a given period. By superimposing the indiviual policies in an attention mechanism, AttenPi is end-to-end differentiable, without the requirement for intermediate reward functions.

We show that, while AttenPi requires careful tuning and regularization to avoid convergence to a non-hierarchical policy, it is capable of performance that is as good, if not better, than single policy models.

2. Related Work

The framework for hierarchical reinforcement learning was formalized in (3), which introduces the notion of 'options'. Options can be described as multi-step abstractions of actions. Options are chosen periodically, and us used to choose actions for the following steps.

Optimizing option generation and optimization has proven to be a difficult task without the help of human intervention. One of the most popular algorithms, Feudal Learning, relies on experts to design reward methods for different choices. Then, lower-level agents are trained to optimize the higher-level rewards for their designated choice. This limits practicality, as many environments lack good choice options (2).

Another emerging method is the use of natural-language models to cluster example trajectories. Using natural English descriptions of expert trajectories, trajectories can be clustered into a finite set of 'skills'. Models can then be trained to imitate the actions for a given skill. While promising for environments that involve human interaction, this method is limited by the data requirements of the descriptions (4). ' A recent development in HRL involves the use of 'target states' to act as options. Here, a high-level option generator outputs a state that it wants to achieve. Then, a lower-level policy is trained to achieve that target state. This is arguably the most promising unsupervised HRL method so far, as it has been shown to outperform regular RL algorithms in environments that require long-term planning (1).

This paper builds directly upon HiPPO architecture (5), which maintains a set of sup-policies, and uses a choice agent to discretely choose a sub-policy to follow for the

[*]Equal contribution [1]Department of Computer Science, Stanford University. Correspondence to: Skanda Vaidyanath <svaidyan@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

given sequence. HiPPO trains the choice agent by using conditional probability across the sub-policies, and directly trains the sub-policies using the policy-gradient methods. AttenPi iterates on this idea by choosing sub-policies with a continous attention mechanism, rather than discrete sampling, and avoids the need for reformulation of the loss function.

3. Approach

To tackle the challenge of simultaneously optimizing a high-level option generator and low-level policies, AttenPi uses a fully differentiable architecture that allows gradients to flow from low level policies into the option generator. Then, by applying a policy-gradient algorithm, the entire system can be treated as a single model that is automatically optimized by gradient descent.

3.1. Model Architecture

AttenPi is composed of 2 components: a set of k policies, Π, and an option generator, δ. Every n steps, δ takes in the current state, and outputs an L1-normed attention mask μ over the policies. Then, for the next n steps, the overall policy is computed as a superposition of the policies in Π, weighted by μ. Formally, if the environment is modeled by a Markov Decision Process (MDP) with states $s \in S$ and actions $a \in A$, we have:

$$\Pi = \{\pi_1, \pi_2 ... \pi_k\}, \;\; \pi_i(s) \in \Re^{|A|}, \;\; \pi_{i,j}(s) = P(a_j|s)$$

$$\delta(s_c) = \mu \in \Re^k, \quad ||\mu||_1 = 1$$

$$\pi(s_{c:c+n}) = [pi_1; pi_2; ...pi_k]\mu$$

3.2. Training

Since the attention mechanism is modelled by a matrix-vector product, we can calculate the gradients for both δ and Π by propagating from the final output policy π. To calculate the loss, we use the popular REINFORCE gradient algorithm, which maximizes the log-probability of taking actions that have a higher observed advantage. Here, we calculate the advantage using a separate network $b(s)$ that is trained simultaneously to the policy to predict the value of a given state, $V(s)$.

$$\mathcal{L}_r = -\sum_{t=0}^{T} \log[\pi(s_t)] A_t, \quad A_t = G_t - b(s_t)$$

This allows the gradients of each π_i to be calculated as:

$$\nabla_{\pi_i} \mathcal{L}_r = \sum_{t=0}^{T} \frac{\mu_i \nabla \pi_i(s_t)}{\pi(s_t)} A_t$$

While the gradient of δ can be calculated as:

$$\nabla_{\delta} \mathcal{L}_r = \sum_{t=0}^{T} \sum_{i=0}^{k} \frac{\pi_i(s_t) \nabla \mu_i}{\pi(s_t)} A_t$$

3.3. Evaluation

In order to evaluate the performance of the described model, 3 metrics are used to measure not only the performance, but also how 'hierarchically' the model is performing:

(i) *Performance*: This measures how well the model is able to achieve rewards in the given environment. This is quantified as the expected return of a trajectory $E_{\sim\tau}[\sum_{t=0}^{T} \gamma^t r(s_t, a_s)]$ (The return is discounted due to the nature of the test environment, described below).

(ii) *Choice Variance*: This measures how much μ varies across different states. This is monitored to avoid outcomes where δ converges to always heavily attending to the same pi_i, which is undesirable because it reduces the model to a classical single-policy model represented by that π_i, instead of being truly heirchical. This is measured as the Jenson-Shannon divergence between different μ outputs, where a higher value indicates more varience:

$$\sigma_{\mu} = \frac{1}{T} \sum_{t=0}^{T} KL(\mu_t || \bar{\mu})$$

(iii) *Policy Variance*: This measures how much the π_i vary between themselves. This is monitored to avoid outcomes where Π all converge to the same distribution. This is problematic because it means that π is always the same for a given state, regardless of μ. In that situation, δ could be disregarded and the model would again reduce to a classical single-policy model. This is quantified by the Jenson-Shannon average divergence between policies across states:

$$\sigma_{\pi} = \frac{1}{kT} \sum_{t=0}^{T} \sum_{i=1}^{k} KL(\pi_i(s_t) || \bar{\pi}(s_t))$$

3.4. Regularization

It turns out that the vanilla algorithm described above is not sufficient to induce hierarchical behavior, as the choice

variance will almost always reduce to zero. This is likely to due positive feedback in the attention mechanism: if one policy becomes better than its peers, δ becomes more likely to attend to it, so it will receive larger gradient updates from the attention mechanism, causing it to outpace its peers further, and so on. The most obvious way to prevent this would be to directly optimize σ_μ as a loss term, but the Jenson-Shannon divergence has a positive second derivative, causing it to become the dominating loss term and preventing the model from learning a good policy. Therefore, we propose 2 potential methods of regulating the choice variance: reconstruction error and gradient masking.

3.4.1. Reconstruction Error

One can frame the problem of optimizing the Choice Varience as a problem of optimizing the 'information retention' going from the state into the choice. The idea is that if μ is always the same, then an observer could never correlate μ with the state that it came from. If we can encourage δ to construct a μ such that an observer can reconstruct properties of the its state, then μ must have some variance in its values for the observer to measure.

To take advantage of this theory, we introduce a state encoder, ϕ_s, that takes in the state and maps it to some latent vector l_s. We also have a choice encoder ϕ_μ that takes in μ and produces a latent vector l_μ. We then want to maximize the similarity between latent vectors from the same state, $l_s(s_i)$ and $l_\mu(s_i)$, and minimize the similarities between latent vectors from different states, $l_s(s_j)$ and $l_\mu(s_i)$. Using dot product similarity with L1-normed latent vectors, this gives us the following regulatory loss function, where s_x are randomly sampled from different trajectories. Note that modifications to this equation, such as different similarity measures or sigmoid activations may have a drastic effect on outcomes.

$$\mathcal{L}_\mu = -\sum_{t=0}^{T}\left[l_\mu(s_t)^T l_s(s_t) - \sum_{x=1}^{m} l_\mu(s_t)^T l_s(s_x)\right]$$

To optimize this loss function, we can again directly backpropagate into ϕ_s and ϕ_μ, and into δ from ϕ_μ.

3.4.2. Gradient Masking

As described above, there is a positive-feedback loop where better policies receive larger gradient updates. To prevent this, we can create a modified μ' that is less extreme, and use that for the Π gradient updates. Assuming that δ outputs some logits η that are converted into μ through the softmax function, we can calculate $\hat{\mu}$ as follows:

$$\mu' = \text{softmax}(\Delta\eta), \quad \Delta \in [0, 1]$$

We then create a modified policy π' using this pseudo-attention:

$$\pi'(s_{c:c+n}) = [pi_1; pi_2; ...pi_k]\mu'$$

Then, we plug this into the REINFORCE algorithm to get the loss that is used to differentiate Π (note that δ is still differentiated over $\mathcal{L}_\nabla$ described above).

$$\mathcal{L}_\Delta = -\sum_{t=0}^{T}\log[\pi'(s_t)]A_t$$

3.4.3. Total Loss Function

To get the total loss function of the model, we combine the individual loss functions in a linear superposition, such that each network is recieving gradients from each relevant loss component, but the components do not interact. This also introduces 2 hyper parameters, λ_μ and λ_Δ, which weight how much each component contributes to the system.

$$\mathcal{L} = \mathcal{L}_\Delta + \lambda_\mu\mathcal{L}_\mu + \lambda_\Delta\mathcal{L}_\Delta$$

4. Experiments

The model was tested using the Procgen Benchmark (6), which consists of procedural generated reinforcement environments that require agents to learn generalizable skills. Specifically, the "coinrun" environment was used, in which the agent must control a platforming robot to collect coins for positive rewards.

In our experiments, episodes were truncated after 128 steps, receiving zero reward is the episode terminates without reaching a coin. The options "use_backgrounds=False", "restrict_themes=True", and "use_monochrome_assets=True" were applied to the environment to simplify the model inputs. Epochs were sampled as sets of 8 episodes, with a small memory buffer maintained to improve sampling efficiency. Finally, due to computational constraints, the environment was set to "easy" mode, which makes it easy to achieve higher rewards.

4.1. Vision Pre-training

The state of the environment is representing by a 64x64x3 rgb image. For our experiments, we concatenated 3 most recent frames to get a new 64x64x9 image, which was passed

into a convolutional neural network (CNN) to act as each model. To improve training efficiency, each model was initialized as a pre-trained encoder CNN. The pretraining task was encoding subsequent states into similar feature vectors, and non-subsequent states into different ones (analogous loss function to $\mathcal{L}_\mu$).

4.2. Benchmark

To compare the hierarchical agent with a traditional algorithm, the benchmark was defined as an AttenPi model identical to the others used in experimentation, except with $k = 1$. This makes the model functionally identical to a traditional policy model, since $pi = pi_1$ for all μ.

4.3. Results

Tuning revealed that the optimal hyper-parameters were $\lambda_\mu = 3$, $\lambda_\Delta = 3$, and $\Delta = 0.25$, so those were the values used throughout testing. A discount of $\gamma = 0.97$ was also used, with the reward of an episode defined as the discounted return from the first state. Finally, option lengths of $n = 8$ were determined as balance between long-term decision-making and short-term specialization.

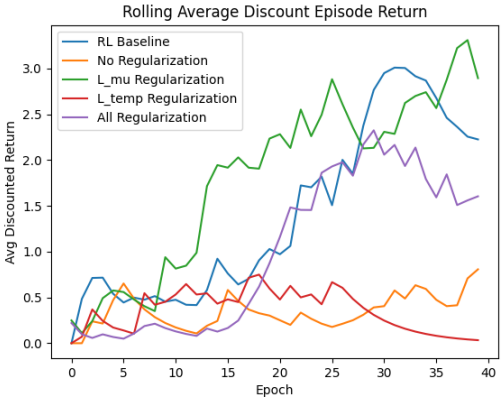

Figure 1. Comparison of the performance of different regularization methods against a non-hierarchcal baseline.

In term of reward performance, the model that only used $\mathcal{L}_m u$ regularization learned the most quickly. This is likely due to to the state encoder introducing more sampling diversity, and forcing different policies to learn more often. Surprisingly, the $\mathcal{L}_m u$ showed little performance gain over the non-regularized model. While $\mathcal{L}_m u$ regularization is the only regularization method that outperforms the baseline, the model with both regularization methods was not far behind.

We also see the correlation between choice variance and performance. We see that without regularization, the algorithm had essentially zero choice variance, and waa always

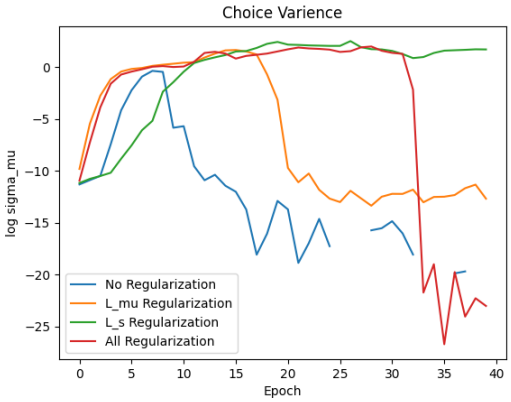

Figure 2. Comparison of the choice variance between different regularization methods.

outputting the same μ (the gaps in the plot are areas where variance equals exactly zero, so the log is undefined). We also see that $\mathcal{L}_t emp$ regularization is the only method that maintains high variance throughout, with the expense of poor performance. $\mathcal{L}_m u$ regularization had a positive impact on variance, though it was not able to be maintained for the entirety of training. The mixed regularization was able to maintain high variance for most of training, but plummeted towards the end - which could possibly be due to a code error.

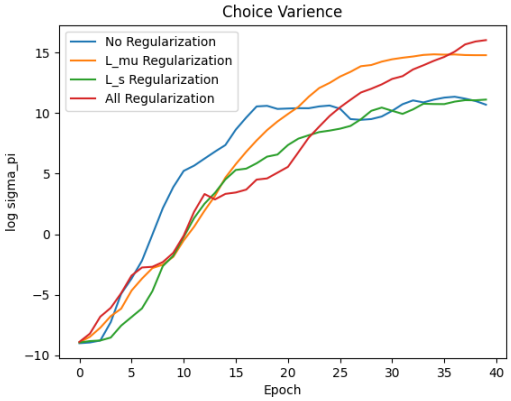

Figure 3. Comparison of the policy variance between different regularization methods.

Surprisingly, every form of regularization maintains a high, and nearly monotonically increasing, policy variance. This shows the lack of need for explicit policy variance control.

4.4. Analysis

Without regularization, AttenPi has both poor performance and poor hierarchical properties, showing the need for addi-

tional considerations. While $\mathcal{L}_\mu$ gives the best performance, it is problems maintaining useful hierarchical properties - this may fixable in different environments or with different parameters, but the current results show more potential for hierarchical pretraining than actual hierarchical inference. By itself, $\mathcal{L}_\Delta$ has few advantages over the vanilla model.

Optimistically, the model with mixed regularization was able to learn similarly to the baseline, while still maintaining high variance. This shows that with appropriate care, AttenPi has the potential to be a powerful HRL framework. While it was not able to outright beat the performance of the baseline, it was able to match it while still maintaining a hierarchical structure.

Potential benefits of AttenPi with mixed regularization compared to the baseline are two-fold. First, it has the potential to scale for more complex tasks, with each sub-policy specializing in a specific skill. Second, it adds a layer of explainability to the system. By monitoring μ, a human can interpret some level of the agent's planning, and ore easily diagnose why it performs the way that it does.

5. Conclusion

While more testing is likely required to claim definitive results, the AttenPi architecture shows promise as a general-purpose HRL architecture. With end-to-end differentiablility, it can be trained with little modification to traditional policy-gradient methods, only requiring the addition of regularization components.

There are extensive opportunities for future work involving this idea, including improved loss functions, testing in more complex environments, pre-trained sub-policies, variable-length options, and supervised skill instruction.

6. Code

The repo containing the code for this project can be found at: github.com/aklein4/HeiRL

References

[1] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018.

[2] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017.

[3] D. Precup Richard S.. Sutton a and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning, 1999.

[4] Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. Lisa: Learning interpretable skill abstractions from language, 2022.

[5] Alexander Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. In *International Conference on Learning Representations*, 2020.

[6] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.