Creating Hierarchical Dispositions of Needs in an Agent

1st Tofara Moyo
Bulawayo , Zimbabwe
tofaramoyo@gmail.com, Mazusa AI

***Abstract*—We present a novel method for learning hierarchical abstractions that prioritize competing objectives, leading to improved global expected rewards. Our approach employs a secondary rewarding agent with multiple scalar outputs, each associated with a distinct level of abstraction. The traditional agent then learns to maximize these outputs in a hierarchical manner, conditioning each level on the maximization of the preceding level. We derive an equation that orders these scalar values and the global reward by priority, inducing a hierarchy of needs that informs goal formation. Experimental results on the Pendulum v1 environment demonstrate superior performance compared to a baseline implementation.We achieved state of the art results.**

I. Introduction

The advent of Artificial Intelligence (AI) has witnessed significant advancements in Deep Reinforcement Learning (RL). A crucial aspect of deep RL involves defining the output layer of a neural network to accommodate the dimensionality of the action space. When the action space is discrete and relatively small, selecting an action from the output distribution is straightforward. However, as the dimensionality of the action space increases or transitions to a continuous domain, identifying the optimal action becomes a computationally expensive optimization problem.

Many real-world applications, such as robotics and autonomous vehicles, inherently require continuous action spaces. For instance, controlling the movement of a robotic arm or regulating the steering and acceleration of a self-driving car necessitate precise and continuous actions. The complexity of these tasks underscores the need for efficient and effective deep RL methodologies capable of handling continuous action spaces.

Instead of modeling the state-action Q-value function, model-free continuous control via reinforcement learning directly optimizes a policy function which maps states to probability distributions over continuous action .

The complexity of a policy learned by reinforcement learning (RL) algorithms is inherently bounded by the complexity of the reward function. Consequently, significant efforts have been devoted to crafting intricate reward functions that can guide RL agents towards sophisticated behaviors.In contrast, humans and other animals appear to develop complex behaviors through a hierarchical process, wherein an initially simple reward function focused on fundamental drives such as pain avoidance and pleasure seeking serves as the foundation for a layered structure of dispositions.

Each level in this hierarchy is oriented towards satisfying the preceding levels, ultimately referencing the base reward function.

The mechanisms underlying this process remain unclear. However, if we could induce artificial agents to learn hierarchical reward functions, it would enable the specification of simple base reward functions, allowing the algorithm to autonomously develop complex goals. A hierarchical reward function would confer upon the agent the capacity to pursue intricate objectives. The hierarchical structure of human needs has been extensively studied, yielding frameworks such as Maslow's Hierarchy of Needs. This hierarchy progresses from fundamental, essential needs to more abstract and complex requirements, including self-actualization.

This paper presents a novel approach to inducing hierarchical reward structures in artificial agents. Our method involves introducing a secondary rewarding agent that parallels the traditional agent, receiving identical state inputs. The rewarding agent features a continuous action output layer, wherein the outputs serve as signals rather than control inputs.

We propose an equation that integrates these signals, yielding a reward signal that is used to reinforce the traditional agent. This framework is designed to elicit a hierarchical organization of needs within the traditional agent, promoting more effective and efficient learning.

Our approach offers two primary benefits: enhanced stability throughout the training process and improved accuracy in the learned policy.

II. Background

A. Markov decision process

We formulate our model of continuous control reinforcement within the framework of a finite Markov Decision Process (MDP). An MDP is defined by the tuple: $M = \langle S, A, s_0, r \rangle$ where S denotes the state space A denotes the action space $s_0 \in S$ denotes the initial state $r(s,a) : S \times A$ denotes the reward function, which assigns a scalar value to each state-action pair.At each time step t, the agent selects an action a_{t+1} according to a policy$\pi : S \longrightarrow A$, which can be either stochastic or deterministic.A stochastic policy is defined as a probability distribution over actions given a state $\pi(a \mid s) : S \longrightarrow P(A)$ where $P(A)$ denotes the set of probability distributions over A.A deterministic policy can be obtained by taking the expected value of the stochastic policy $\pi(s) = \mathbf{E} a \pi(a \tilde{\mid} s)[a]$.The objective of the agent is to maximize its future expected reward: $\max \pi \mathbf{E}[\sum_{t=0}^{\inf} r(s_t, a_t)]$.

B. Policy Gradient Methods

Policy gradient methods are a type of reinforcement learning algorithm that learns to optimize the policy directly, rather than learning the value function. The policy gradient theorem provides the foundation for policy gradient methods. It states that the gradient of the expected cumulative reward with respect to the policy parameters can be computed as:

$$\bigtriangledown_\theta J(\pi_\theta) = \mathbf{E}_\tau \tilde{\pi_\theta}[\textstyle\sum_{t=0}^{T} \bigtriangledown_\theta \log \pi\theta(a_t \mid s_t) Q^{\pi_{theta}}(s_t, a_t)]$$

where $J(\pi_\theta)$ is the expected cumulative reward π_θ is the policy parameterized by $\theta\tau$ is a trajectory sampled from the policy s_t and a_t are the state and action at time t $Q\pi\theta(s_t, a_t)$ is the action-value function $\bigtriangledown\theta \log \pi\theta(a_t \mid s_t)$ is the gradient of the log-probability of the action.

1) Actor-Critic Methods: Actor-critic methods are a type of policy gradient method that uses an actor to represent the policy and a critic to estimate the value function. The actor and critic are updated simultaneously using the policy gradient theorem. In this work we used a PPO implementation based of an actor critic network. The actor update rule is :

$$\theta \longleftrightarrow \theta + \alpha \mathbf{E}_{\tau \tilde{\pi\theta}}[\sigma_{t=0}^{T} \bigtriangledown \theta \log \pi\theta(a_t \mid s_t)(Q_{\pi\theta}(s_t, a_t) - V_{\pi_\theta}(s_t))]$$

where $V_{\pi_\theta}(s_t)$ is the state-value function.

The critic update rule being:

$$\omega \longleftarrow \omega + \beta \mathbf{E}_{\tau \tilde{\pi_\theta}}[\textstyle\sum_{t=0}^{T}(Q_{\pi\theta}(s_t, a_t) - V_{\pi_\theta}(s_t)) \bigtriangledown \omega V_{\pi_\theta}(s_t)]$$

where ωs is the critic's parameter and $\beta$is the learning rate.

2) Proximal Policy Optimization: We implement a policy gradient method using a truncated version of the generalized advantage estimator (GAE). The GAE is computed as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \qquad (1)$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t), \qquad (2)$$

The policy is run for T timesteps, with T less than the episode size. We use the standard notation for the discount factor γ and GAE parameter λ. To perform a policy update, each of N parallel actors collects T timesteps of data. We then construct the surrogate loss on these NT timesteps of data and optimize it using the ADAM algorithm with a learning rate α. We use mini-batches of size $m \leq NT$ for K epochs.

We use a combined loss function that includes the policy surrogate, value function error term, and entropy term:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right],$$

where S denotes the entropy bonus, $L_t VF$ is the value function squared-error loss, and c_1 and c_2 are coefficients for the value function loss and entropy term, respectively.

C. Hierarchical Reward Functions

We design a hierarchical reward function that induces a hierarchy of dispositions. The reward function is defined as:

$$r = R * r_1 + R$$

where R is the global reward at time step t and r is a single scalar value derived from the sole output of the rewarding agent. This equation sets up a two-stage hierarchy, where the actions associated with optimizing r_1 are followed in such a way that they simultaneously optimize R and hence r.

III. Experiments

This section presents the results of our experimental evaluation of the proposed hierarchical reward function on a continuous control problem from the OpenAI Gym suite: the Pendulum-v1 environment with a low-dimensional state space.

The architecture of our experimental setup for the Pendulum-v1 environment consisted of a neural network with a final layer outputting a 1-dimensional real-valued vector. Our implementation of the Proximal Policy Optimization (PPO) algorithm was based on a publicly available GitHub repository.

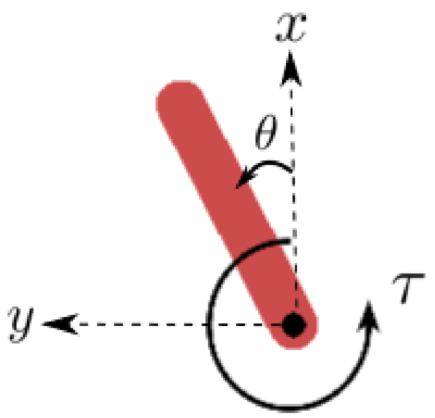
For each environment, we trained five models using different random seeds for a fixed total number of time steps. Following completion of training, each model was evaluated over 100 consecutive episodes to assess its performance.

The performance of each model was evaluated using the cumulative reward obtained over the 100 evaluation episodes. This metric provides a comprehensive assessment of the model's ability to maximize the reward function while adapting to the environment's dynamics.

A. *Pendulum-v1*

The Pendulum-v0 environment is a well-established continuous control task from the OpenAI Gym suite. The primary objective of this task is to stabilize a pendulum by applying a torque, effectively balancing the pole in an upright position.

The Pendulum-v0 environment is characterized by: An unbounded, 3-dimensional observation space A 1-dimensional action space, where actions represent the torque applied to the pendulum Bounded actions within the interval $[-2, 2]$.



The agent follows an actor-critic framework. The actor $\pi_\theta(a|s)$ consists of a neural network made of 3 fully-connected layers of 64 units each, with $\tanh$ activation functions. The output layer has 1 linear neuron. The critic $V_{\theta_v}(s)$ does not share layers with the actor, but has an equivalent architecture of 3 hidden layers, and one output neuron representing the value function.

B. *Reward agent*

The reward agent follows an actor-critic framework. The actor $\pi_\theta(a|s)$ consists of a neural network made of 5 fully-connected layers of 64 units each, with $\tanh$ activation functions. The output layer has 6 linear neuron. This is because we wanted to set up a 6 level heirarchy.The equation we used was of the form of the equation we presented earlier but evolved to include a hierarchy of 6 steps. It took the following form.

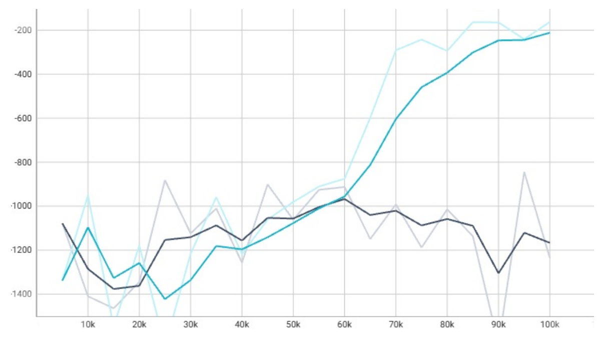
$$r = R * (r1 * (r2 * (r3 * (r4 * (r5 * r6 + r5) + r4) + r3) + r2) + r1) + R$$

As you can see the equation is a rewrite of the earlier equation, with a replacement of terms.

IV. Results and Discussion

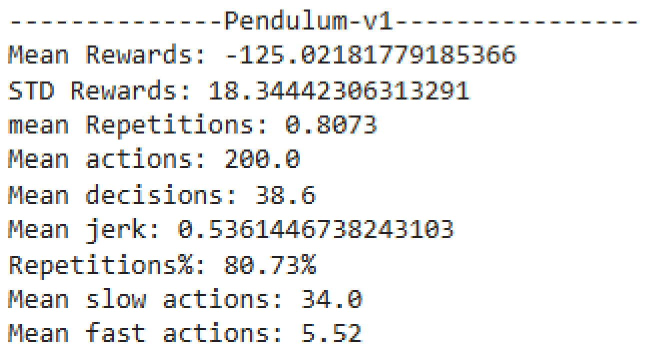
A. *Pendulum-v1*

For the Pendulum-v1 environment, we observe that our method learnt faster, with greater stability and higher rewards than the PPO method without our adjustments.See FIG 2.

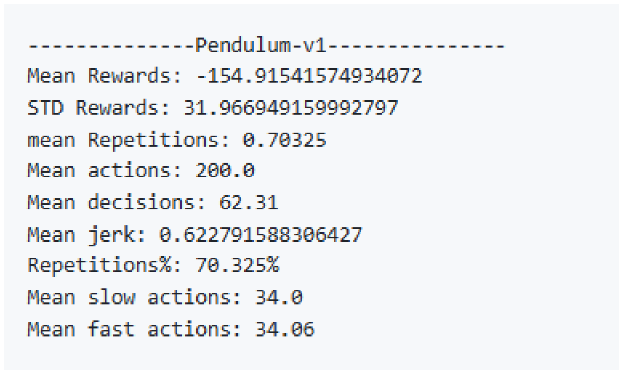


In blue is our algorithm vs black, the established method -Average rewards over a 10-episode window for the Pendulum task.

Additionally ,in another run, we were able to beat the state of the art with the TLA model after adapting the github code to include our reward function.The previous state of the art was held by the vanilla version of the TLA algorithm.The results obtained by it were -154 reward points while ours achieved a higher score of -125.Below is a comparison of the two methods results.



```
--------------Pendulum-v1---------------
Mean Rewards: -125.02181779185366
STD Rewards: 18.34442306313291
mean Repetitions: 0.8073
Mean actions: 200.0
Mean decisions: 38.6
Mean jerk: 0.5361446738243103
Repetitions%: 80.73%
Mean slow actions: 34.0
Mean fast actions: 5.52
```



```
--------------Pendulum-v1---------------
Mean Rewards: -154.91541574934072
STD Rewards: 31.966949159992797
mean Repetitions: 0.70325
Mean actions: 200.0
Mean decisions: 62.31
Mean jerk: 0.622791588306427
Repetitions%: 70.325%
Mean slow actions: 34.0
Mean fast actions: 34.06
```

V. Future Work

In our initial approach, we assumed a linear reward function, where scalar values are prioritized and each level is multiplied by the level above, with an additional term. However, this simplistic model may not accurately capture the complexities of real-world systems.

A more comprehensive approach would involve using a graph to model the reward dynamics of the system. In this framework nodes at the same depth would be summed, rather than multiplied, to capture the cumulative effects of different factors and nodes above would be multiplied to represent the hierarchical relationships between different components.This graph-based approach would enable a more nuanced and accurate representation of the reward function.

To implement this graph-based reward modeling, we propose a reward critic architecture that takes the state of the agent as input and outputs a graph representing the reward dynamics. We would then trace each leaf node up to the root, collecting values in an array to form a line-based partial reward function for the traditional agent and sum the rewards over all leaves to obtain the final reward.

To further enhance the reward critic, we can modify it to take into account the actions of the traditional agent, in addition to the state. This would enable the reward critic to evaluate both states and actions, providing a more comprehensive assessment of the agent's behavior.By exploring these graph-based reward modeling and reward critic architectures, we can develop more sophisticated and accurate reward functions that capture the complexities of real-world systems.

VI. Conclusions

In this study, we conducted a comprehensive evaluation of the effectiveness of hierarchical reward functions in reinforcement learning. Our results demonstrate that agents trained with hierarchical reward functions exhibit faster convergence, improved stability, and higher final rewards compared to agents implementing standard Proximal Policy Optimization (PPO) algorithms.

A comparative analysis of the performance of agents trained with hierarchical reward functions and standard PPO algorithms reveals significant advantages of the former approach. Specifically: Agents trained with hierarchical reward functions exhibit faster convergence rates, achieving optimal performance in fewer iterations .The stability of agents trained with hierarchical reward functions is improved, with reduced variance in performance across different trials and the final rewards obtained by agents trained with hierarchical reward functions are consistently higher than those achieved by agents implementing standard PPO algorithms.

Our results suggest that the proposed method of implementing hierarchical reward functions is effective for simple cases. To further establish the scalability and generalization of this approach, we plan to extend our experiments to more complex environments with intricate dynamics.

We propose to develop more complex graph-based hierarchical reward functions to capture nuanced relationships between different components. This will enable the creation of more sophisticated reward functions that can effectively guide the learning process in complex environments.

A key advantage of the proposed hierarchical reward function approach is the potential for component reuse and transfer learning. By fostering the reuse of components learned early on in the development process, we can accelerate the learning process and improve the overall performance of the agent.

The proposed hierarchical reward function approach has significant implications for complex goal formation and navigation in real-world environments. By emulating the hierarchy of needs exhibited by humans, we can create agents that are capable of navigating complex environments and achieving sophisticated goals.

Future work will focus on extending the proposed hierarchical reward function approach to more complex environments and developing more sophisticated graph-based reward functions.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of go without human knowledge," nature, vol. 550, no. 7676, pp. 354–359, 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, pp. 436–444, 2015.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in International conference on machine learning. PMLR, 2015, pp. 1889–1897.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International conference on machine learning. PMLR, 2016, pp. 1928–1937.

[7] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," arXiv preprint arXiv:1611.01224, 2016.

[8] P.-W. Chou, D. Maturana, and S. Scherer, "Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution," in International conference on machine learning. PMLR, 2017, pp. 834–843.

[9] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for Reinforcement learning with function approxima-Tion," in Advances in neural information processing systems, 2000, pp. 1057–1063.

[10] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "Highdimensional continuous control using generalized advantage estimation," arXiv preprint arXiv:1506.02438, 2015.

[11] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[13] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[14] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," 2018.

[15] P. Rodrigues and S. Vieira, "Optimizing agent training with deep learning on a self-driving reinforcement learning environment," in 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2020, pp. 745–752.

[16] S. Risi and K. O. Stanley, "Deep neuroevolution of recurrent and discrete world models," in Proceedings of the Genetic and Evolutionary Computation Conference, 2019, pp. 456–462.

[17] A. Gaier and D. Ha, "Weight agnostic neural networks," arXiv preprint arXiv:1906.04358, 2019.

[18] R. Jena, C. Liu, and K. Sycara, "Augmenting gail with bc for sample efficient imitation learning," arXiv preprint arXiv:2001.07798, 2020

[19] Optimizing Attention and Cognitive Control Costs Using Temporally-Layered Architectures- Devdhar Patel, Terrence Sejnowski, Hava Siegelmann