University of British Columbia

Advanced Machine Learning Tools for Engineers
EECE 568

Transformer Based Reinforcement Learning

Author: Ethan McKeen

Author: Christy Zhang

November 27, 2024

1 Goal

The goal of this project is to integrate transformers into reinforcement learning to enhance the agent's ability to model long-term dependencies and improve decision-making in complex environments. By leveraging the self-attention mechanism, this approach aims to address the limitations of traditional RL architectures in handling sequential tasks. The project will evaluate whether transformers can outperform existing models, particularly in scenarios requiring long-term planning, and provide insights into their potential for advancing reinforcement learning across diverse simulated environments.

2 Related Work

The application of transformers in reinforcement learning (RL) has opened new possibilities for modeling sequences and capturing long-term dependencies, addressing challenges that traditional RL approaches often face. Transformers are particularly effective in partially observable environments or tasks requiring the ability to reason over long time horizons.

2.1 Decision Transformer

One framework integrating transformers into RL is the Decision Transformer, introduced by Chen et al. (2021). This method reframes RL as a sequence modeling problem rather than a standard optimization problem of maximizing cumulative rewards. It treats states, actions, and rewards as sequential data, using a transformer to predict the next action based on the past trajectory and a desired return. By aligning RL with supervised learning principles, Decision Transformers enable the use of pre-trained models and fine-tuning techniques, significantly reducing training complexity [1].

2.2 Trajectory Transformer

Another approach is the trajectory Transformer introduced by Janner et al. (2021). Instead of focusing on individual actions, they looked at modeling entire trajectories of states, actions, and rewards. They tokenize these trajectories and use a transformer to predict the sequences while maintaining causal constraints, the model ensures that only past information influences predictions. This method allows for trajectory-level optimization, making it particularly effective for long-term planning and decision-making [2].

These two approaches illustrate complementary strengths in integrating transformers with RL. The Decision Transformer focuses on immediate action prediction using sequential data, excelling in reactive decision-making. In contrast, the Trajectory Transformer prioritizes planning and long-term optimization, proving its utility in complex tasks with extended time horizons. Together, they showcase how transformers can address key RL challenges and expand the possibilities for future research in this area.

3 Strategy

3.1 Transformers

Transformers were the chosen architecture to be adapted into an RL environment and achieve our goal because they show promise in multiple areas including:

Scalability: Transformers handle large-scale data efficiently, making them suitable for complex RL environments with extensive state-action spaces.

Memory: Their self-attention mechanism allows them to capture long-range dependencies and retain relevant past information, which is crucial for RL tasks requiring memory of past states and actions. This is the case in many RL environments considering that most RL environments require sequential decision making.

Generalization: Transformers excel at generalizing from previous experiences, helping RL agents to adapt more swiftly and effectively to new or unseen scenarios.

3.2 Evaluation

Our strategy for evaluating and comparing RL model performance is by measuring:

Learning Curve: Plotting the rewards versus training episodes to observe the learning progress. A steep learning curve indicates that the agent is learning quickly. This provides an understanding of how well the agent performs on average for each episode and can give insight into the consistency of the model. This also provides information about sample efficiency such as how many interactions with the environment are required to achieve a certain level of performance.

Convergence Time: The time or number of iterations it takes for the agent to reach a stable, optimal policy. Faster convergence is generally better. This provides insight into the computational overhead of the model and the general complexity of the model.

3.3 Environment

To test the effectiveness of a transformer based reinforcement learner we chose environments that would play to the transformers strengths. The environments Pendulum and Lunar Lander were chosen because they involve complex action state interactions, the environments rely on sequential decisions and episode lengths can be vary in length and have important temporal dependencies [3][6]. This makes them ideal for evaluating the performance and generalization capabilities of transformer-based reinforcement learners.

4 Transformer Based Actor Critic (Methods That Worked)

The model succeeds is a Transformer Based Actor Critic (TBAC) agent. This model uses the key ideas of an Actor Critic reinforcement learner but leverages a transformer architecture for a more robust model, and is found particular success in the Pendulum environment. Some ideas from Deep Deterministic Policy Gradient (DDPG) were used to improve this model [8] [1]. from This section introduces the Pendulum environment, architecture of the TBAC agent, and demonstrates the training progress.

4.1 Pendulum Environment

The TBAC agent found success in the Pendulum environment [3]. The Pendulum environment in OpenAI Gym is a classic control problem where the objective is to swing a pendulum to keep it upright. The pendulum starts in a random position and the goal is to apply torque to keep it balanced vertically. The state space consists of the sine and cosine of the pendulum's angle and its angular velocity, while the action space is a continuous value representing the torque applied.

Num	Observation	Min	Max
0	$x = cos(\theta)$	-1.0	1.0
1	$y = sin(\theta)$	-1.0	1.0
2	Angular Velocity $\frac{\delta\theta}{\delta t}$	-8.0	8.0

Table 1: Pendulum State Space

The reward is structured as:

$$r = -(\theta^2 + 0.1 * (\frac{\delta\theta}{\delta t})^2 + 0.001 * \tau^2)$$

It encourages the agent to keep the pendulum as vertical as possible, while minimizing the amount of torque used. This environment serves as a standard benchmark for testing and evaluating reinforcement learning algorithms in a relatively simple yet challenging continuous control task. Figure 1 depicts the pendulum envrionment.

[1] DDPG combines the strengths of Deep Q Learning and actor-critic methods to enable continuous action spaces by using a deterministic policy gradient approach.

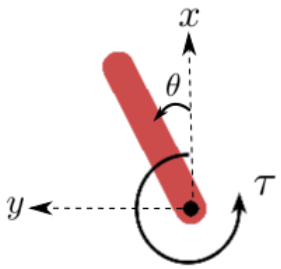

Figure 1: Pendulum Environment

4.2 Transformer Based Actor Critic Architecture

This model builds upon the core principles of an Actor-Critic reinforcement learner while incorporating a transformer architecture for enhanced robustness. Its implementation is inspired by a successful reinforcement learning model that utilizes DDPG by Kanishk Navale, against which its performance is also compared [4].

4.2.1 Actor Critic Architecture

The agent functions within a continuous action space and utilizes the Actor-Critic architecture, which consists of two main components: the Actor and the Critic. The Actor is responsible for mapping observations from the environment to the optimal actions, essentially guiding the agent's behavior to maximize its performance. On the other hand, the Critic evaluates the quality of the actions taken by the Actor by predicting the expected return, or value, associated with those actions. A significant innovation in this architecture is the incorporation of Transformer blocks within the Critic. This enhancement allows the Critic to more effectively capture temporal dependencies and patterns in the input data. By leveraging the self-attention mechanism of Transformers, the Critic can analyze and understand the sequential nature of the observations and actions. This architecture is outlined in figure 2 below.

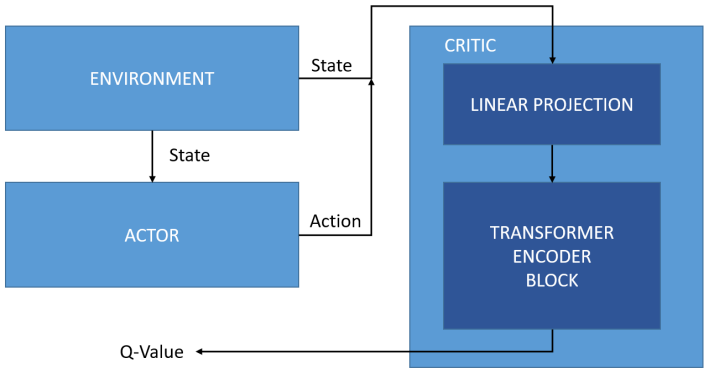

Figure 2: TBAC Architecture Block Diagram

4.2.2 The Critic

The model architecture leverages the Actor-Critic framework integrated with a Transformer-based Critic and a dense neural network Actor. The Critic's architecture centers on a TransformerBlock, which uses multi-head self-attention to process sequences of state-action pairs, capturing dependencies and complex patterns in the data. The Critic begins with an input projection layer that reduces the dimensionality of the concatenated state-action vector into a lower-dimensional embedding space. This is followed by the TransformerBlock, which includes a multi-head attention mechanism, feed-forward layers, layer normalization, and dropout to enhance generalization. Finally, the Critic

outputs a single scalar value representing the estimated Q-value for the given state-action pair. This transformer consists of an encoder only. This is because the decoder in the original Transformer architecture is used to sequentially generate outputs conditioned on encoder outputs and previously generated outputs. The Actor-Critic framework does not require sequential output generation from the Critic. Instead, it predicts a scalar value (the Q-value) for a given state-action pair.

Algorithm 1 Transformer Block Forward Pass

procedure TRANSFORMERBLOCK(x)
    Compute attention output: *attn_output*, _ $\leftarrow$ *MultiHeadAttention*(x, x, x)
    Add and normalize: $x \leftarrow$ *LayerNorm*$(x +$ *Dropout*$($*attn_output*$))$
    Compute feedforward output:

$$\textit{ff\_output} \leftarrow \textit{ReLU}(\textit{Linear}(x)) \rightarrow \textit{Linear}(x)$$

    Add and normalize: $x \leftarrow$ *LayerNorm*$(x +$ *Dropout*$($*ff_output*$))$
    return x
end procedure

Algorithm 2 Critic Forward Pass

procedure CRITIC(state, action)
    Concatenate state and action: `input` $\leftarrow$ [state, action]
    Apply linear projection: `embedded_input` $\leftarrow$ Linear(input)
    Pass through Transformer Block:

$$\texttt{transformed\_input} \leftarrow \textit{TransformerBlock(embedded\_input)}$$

    Compute Q-value: `Q` $\leftarrow$ Linear(transformed_input)
    return Q
end procedure

4.2.3 The Actor

The Actor, on the other hand, is a standard feed-forward neural network composed of four fully connected layers with ReLU activation and dropout for regularization. It maps the environment's state to a continuous action vector scaled using a hyperbolic tangent function (tanh). This design enables the Critic to model complex temporal dependencies, while the Actor ensures computational efficiency and effective policy learning. Both networks utilize the Adam optimizer for gradient updates and are trained in tandem, with the Critic guiding the Actor by estimating the value of its policy. This architecture effectively combines the sequential modeling power of Transformers with the stable convergence properties of Actor-Critic methods.

Algorithm 3 Actor Forward Pass

procedure ACTOR(state)
    Pass *state* through hidden layer H_1: $x \leftarrow$ *ReLU*$(H_1(state))$
    Pass x through hidden layer H_2: $x \leftarrow$ *ReLU*$(H_2(x))$
    Apply dropout: $x \leftarrow$ *Dropout*(x)
    Pass x through hidden layer H_3: $x \leftarrow$ *ReLU*$(H_3(x))$
    Pass x through hidden layer H_4: $x \leftarrow$ *ReLU*$(H_4(x))$
    Compute final action: $action \leftarrow$ *Tanh*$(H_5(x))$
    return $action$
end procedure

4.2.4 The Agent

The agent operates by collecting state transitions through interactions with the environment, incorporating a small amount of exploration noise to encourage policy diversity. Each transition is stored

in a replay buffer with an associated temporal difference (TD) error calculated by the difference between predicted and actual rewards over a time step. As training progresses, transitions with higher TD errors are sampled more frequently. During optimization, the Critic update involves computing target Q-values using target networks and reducing TD errors with a weighted mean squared error (MSE) loss. The Actor update adjusts the policy to maximize the Critic's estimate of the Q-value and updates the prioritized experience replay (PER) priorities based on new TD errors. Soft target network updates gradually align the target networks with their corresponding main networks using a parameter τ, ensuring stability in training. The code for the agent is inspired by the DDPG implementation by Kanishk Navale and the pseudo code can be found in appendix A [4].

4.3 Model Choices

The choice of the Transformer-based Actor-Critic method for reinforcement learning was motivated by its ability to address the challenges of sequential decision-making in dynamic environments, leveraging both the robustness of actor-critic frameworks and the advanced representational power of transformers.

4.3.1 Why Choose the Pendulum Gymnasium

Open AI's Pendulum Gym environment was chosen because it presents continuous state-action spaces, making it difficult to model with tabular methods or discrete action techniques. The tasks involve sequential decision-making, where understanding temporal dependencies and the cumulative effects of actions is crucial. Transformers excel with these tasks and seemed like the perfect solution. A similar model was also tested in the LunarLander environment but the model performed significantly worse than other simpler models. More details about this can be found in section 5, but overall the LunarLander environment did not compliment the strengths of a transformer like the pendulum environment did.

4.3.2 Why Choose Actor-Critic

The Actor Critic base architecture was chosen due to high training stability. Unlike purely policy-based methods, actor-critic methods benefit from using a critic to estimate value functions, reducing variance in policy gradient updates. The dual-network approach helps stabilize learning by separating the value estimation (critic) from policy optimization (actor). Actor-critic methods also naturally extend to continuous action spaces using deterministic or stochastic policy gradients. The use of Temporal Difference (TD) learning in the critic makes actor-critic methods more sample-efficient than purely Monte Carlo approaches [2], which require complete episodes for updates. TD learning is also more computationally efficient than Monte Carlo methods.

4.3.3 Why Choose Transformers

Capturing Temporal Dynamics: Transformers, equipped with self-attention mechanisms, excel at modeling relationships between elements in the input sequence, regardless of their position. This capability is particularly beneficial in environments with long-term dependencies, where actions taken early in an episode can significantly influence rewards much later. By attending to relevant parts of the sequence, transformers can effectively capture these long-range dependencies, making them well-suited for RL tasks with complex temporal dynamics.

Generalization: Transformers, with their self-attention mechanism, allow the model to focus on the most relevant parts of the input sequence, regardless of its position. This adaptability enables the model to capture patterns that are invariant to input ordering or length, which is essential to generalizing in tasks with diverse state-action distributions. This generalization strength is particularly important in reinforcement learning, where the agent must perform well not just on previously encountered states but also in unseen or slightly modified scenarios.

Parallel Processing: One of the key advantages of transformers is their ability to process sequences in parallel. Other models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory

[2] A Monte Carlo simulation is a computational technique that uses random sampling to estimate the probability of different outcomes in complex systems.

(LSTM) networks were considered since they excel in modelling sequential data [9][10]. However, transformers can handle entire sequences at once, significantly improving computational efficiency. This parallel processing capability, despite the transformers' complexity, leads to faster training and inference times.

4.3.4 Model Evolution

Throughout the development of the TBAC model there were many aspects improved. Initial implementations of the agent lacked target networks, leading to potential instability in value updates. Target actor and target critic networks with soft updates via a parameter τ, were added improving stability by decoupling target updates from the rapidly changing main network. Initial models relied solely on policy randomness for exploration, which can be suboptimal. Adding controlled noise to actions during training, improved exploration in continuous spaces. As seen in section 5, initial models did not seperate actor and critic networks. Separate networks, enables independent optimization for policy and value estimation. This modularity allows for different loss functions to compliment each network. Mean squared error (MSE) loss for the critic, provides stable Q-value updates and a deterministic policy gradient loss for the actor, helps maximize the critic's evaluation of the actor's policy.

4.4 Results Evaluation

The Transformer Based Actor Critic (TBAC) model was compared against the highest actor critic model (AC) found on the Pendulum Gymnasium leaderboard [4]. Its results were comparable but it did fall short in convergence speed. Figures 2 and 3 demonstrate the first experiment where the two models were trained over only 500 episodes. 500 of episodes is chosen here to align with the evaluation matrix used in the leaderboard. It can be seen that the transformer based model takes longer to reach maximal rewards and is unable to converge, unlike the other model. However, it is noteworthy that TBAC begins to receive minimum rewards less often than AC. This shows that the transformer, as expected, generalized better than the AC because even after 500 episodes there were still states in which the AC received the minimum amount of reward. The TBAC model's ability to do this suggests it has learned more robust and adaptable policies compared to the AC model.

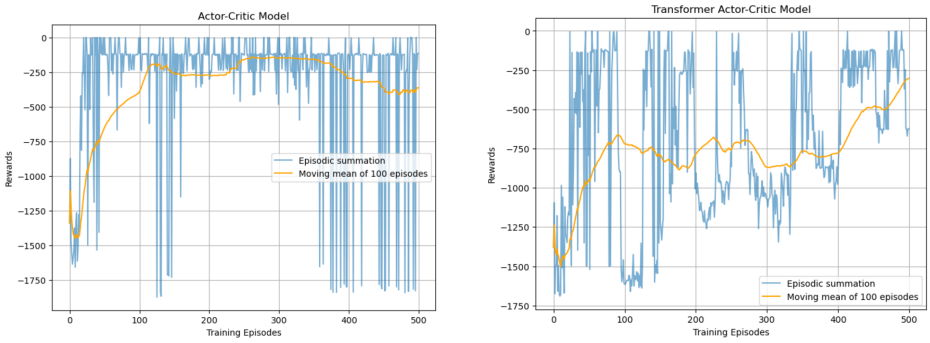

Figure 3: Training Rewards of AC Model on Pendulum Environment in 500 Episodes

Figure 4: Training Rewards of TBAC Model on Pendulum Environment in 500 Episodes

Figures 4 and 5 demonstrate both models' performance after 1000 episodes. It can be seen that both models converge but the AC converges faster. This is due to the models' simplicity. This simplicity allows them to learn and update their parameters more quickly because there are fewer layers and parameters to optimize. Transformers, on the other hand, have multiple layers of self-attention and feedforward networks, which require more computational resources and time to train. TBAC requires roughly 300 more episodes to converge as well but, as previously discussed, TBAC converges with a much more robust model. Unlike AC, the converged model never receives minimum rewards.[3]

[3]Inconsistency in the plots between the 500 and 1000 episode tests is due to the random initialization of the Pendulum environment.

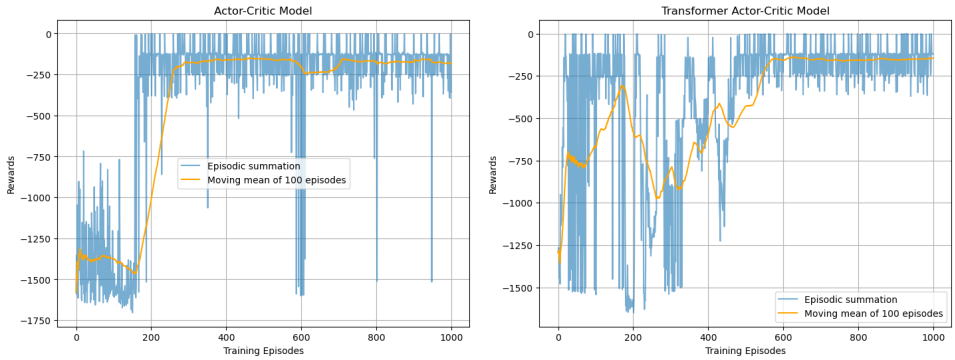

Figure 5: Training Rewards of AC Model on Pendulum Environment in 1000 Episodes

Figure 6: Training Rewards of TBAC Model on Pendulum Environment in 1000 Episodes

5 What Did Not Work

5.1 LunarLander Environment

The LunarLander environment in OpenAI Gym is a classic control problem where the goal is to land a spacecraft safely on a designated landing pad [6]. The lander starts at the top center with a random force applied on it, and the agent must control its descent by firing thrusters to manage its position and orientation. The state space consists of variables such as the lander's coordinates, linear velocities, angle, and angular velocity, along with discrete indicators for whether each leg is in contact with the ground. The action space includes discrete commands to fire the main engine or either of the side engines. precise and fuel-efficient landings with bonuses for safe touchdowns and penalties for crashes, inefficiency, and going out of bounds. This environment serves as a benchmark for testing reinforcement learning algorithms in a physics-based simulation requiring precision, control, and planning.

5.2 Architecture

The initial implementation of Transformer-based reinforcement learning integrates the transformer architecture into the Actor-Critic algorithm. It processes the input state with temporal embeddings and passes it through a Transformer to capture long-term dependencies. The model then computes both the value of the state and the probabilities of possible actions, enabling it to sample an action and store relevant information for training. The procedure is as follows:

Algorithm 4 Transformer Actor-Critic Forward Pass

procedure ACTORCRITICTRANSFORMER(*state*)
    Convert *state* to a tensor: *state* ← *torch.from_numpy(state).float()*
    Add temporal embeddings: *state* ← *state* + *temporal_embedding*
    Project input to higher dimensions: *state* ← *Linear(state)*
    Pass through Transformer Encoder: *transformer_out* ← *TransformerEncoder(state)*
    Extract the last token: *state* ← *transformer_out[-1]*
    Compute state value: *state_value* ← *Linear(state)*
    Compute action probabilities: *action_probs* ← *Softmax(Linear(state))*
    Create action distribution: *action_dist* ← *Categorical(action_probs)*
    Sample an action: *action* ← *action_dist.sample()*
    Store log probability: *logprobs.append(action_dist.log_prob(action))*
    Store state value: *state_values.append(state_value)*
    return *action.item()*
end procedure

5.3 Results

This approach integrates the transformer architecture into the Actor-Critic algorithm. However, a comparison of its training performance with the standard Actor-Critic algorithm revealed that the transformer neither improved performance nor reduced the number of episodes required for convergence.

The baseline model for comparison is an Actor-Critic algorithm that successfully solves the Lunar Lander problem within 1000 episodes. Figure 8 illustrates the training process and the average reward measured over a 20-episode frame. The baseline achieves the problem-solving threshold of >200 rewards within 1000 episodes, with an average reward of 262.40 during 5 consecutive tests.

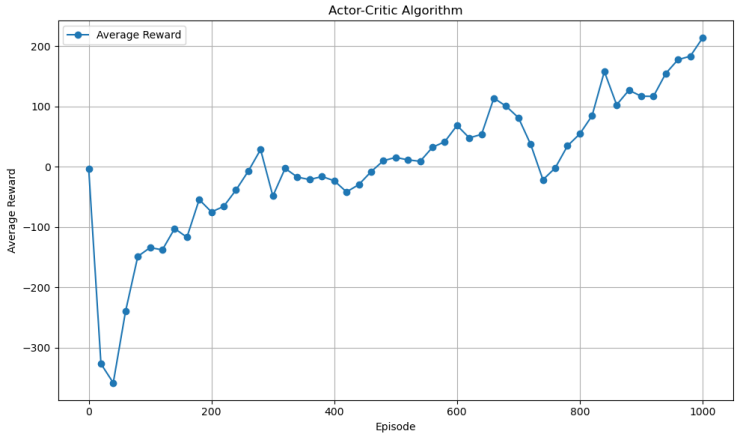

Figure 7: Training Rewards for Actor-Critic Model on Lunar Lander Environment

The transformer-based model demonstrates significantly poorer performance compared to the baseline Actor-Critic model. It fails to exhibit any indication of convergence even after 2000 episodes, which is twice the number of episodes required for the baseline model to converge. As shown in Figure 9, the training progress of the transformer-based model remains inconsistent. Furthermore, the training process for each episode is considerably slower due to the computational complexity of the transformer architecture. Therefore, it can be concluded that the transformer-based approach is less efficient and achieves lower cumulative rewards compared to the standard Actor-Critic algorithm.

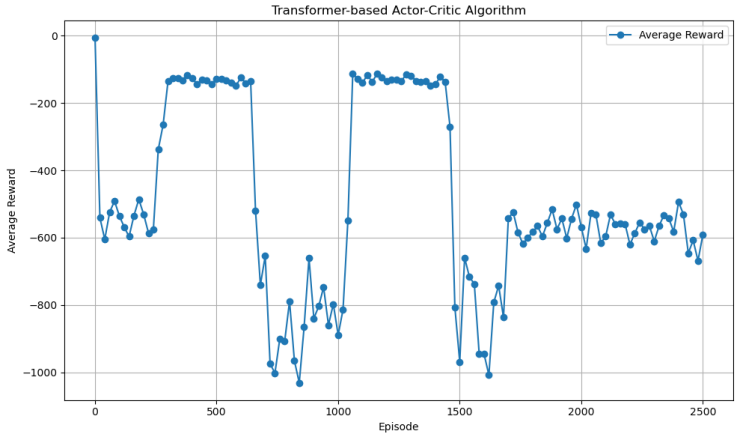

Figure 8: Training Rewards for Transformer-based Actor-Critic Model on Lunar Lander Environment

5.4 Insights on Model Performance

The poorer performance of the transformer-based model can likely be attributed to several factors. First, the sparse rewards in the Lunar Lander environment made training more challenging for the transformer structure, which often requires dense feedback to learn effectively. Additionally, the simplicity of the environment coupled with the complexity of the transformer model resulted in prolonged training times, as the model was unnecessarily sophisticated for the problem. The small observation space further reduced the relevance of the transformer's attention mechanism, as it had limited utility in processing the straightforward input data. While the environment initially appeared to require sequential decision-making, this proved less critical in practice; the agent primarily needed to perform actions to keep the lander upright, with navigating toward the flag being a secondary consideration. These factors collectively contributed to the suboptimal performance of the transformer-based approach in this context. As outlined in the previous section, this limitation can be addressed by applying the approach to more complex environments where sequential decision-making plays a more significant role. These settings involving continuous control and sequential decision-making would better align with the strengths of the transformer architecture, such as its ability to process and prioritize information from a broader temporal context.

6 Conclusions and Future Work

This study demonstrates that transformers can be effectively integrated into reinforcement learning (RL) to address challenges in environments requiring sequential decision-making and long-term dependency modeling. The results highlight the potential of transformers in handling complex RL tasks, as seen in the Pendulum environment. Although the model had an increased convergence time, the Transformer-Based Actor-Critic (TBAC) model offering more robust generalization capabilities compared to traditional actor-critic methods.

However, the findings also underline limitations. In simpler environments like Lunar Lander, the computational overhead of transformers proved unnecessary, as the sparse reward structure and limited sequential dependencies failed to align with the architectural advantages of transformers. This emphasizes the importance of selecting RL tasks that match the strengths of transformer models to justify their complexity and computational cost.

Future work could focus on conducting experiments across a wider variety of environments beyond the Open AI's Gymnasium, such as more complex control tasks and real-world scenarios, to better understand the generalization capabilities of the TBAC model. Advanced optimization techniques can be investigated to improve the convergence speed of TBAC models. Also performing extensive hyperparameter tuning to find the optimal configurations for TBAC models, potentially uncovering settings that enhance performance and reduce training times.

References

[1] L. Chen, K. Lu, A. Rajeswaran, K. Lee, and A. Grover, "Decision Transformer: Reinforcement Learning via Sequence Modeling," arXiv preprint arXiv:2106.01345, 2021. [Online]. Available: https://arxiv.org/abs/2106.01345.

[2] M. Janner, J. Fu, M. Zhang, and S. Levine, "Trajectory Transformer," arXiv preprint arXiv:2106.02039, 2021. [Online]. Available: https://arxiv.org/abs/2106.02039.

[3] Gymnasium Developers, "Pendulum-v1 — Gymnasium documentation," Gymnasium: Classic Control Environments, [Online]. Available: https://www.gymlibrary.dev/environments/classic_control/pendulum/. [Accessed: Nov. 27, 2024].

[4] K. Navale, "Naive MultiAgent Reinforcement Learning," GitHub repository, [Online]. Available: https://github.com/KanishkNavale/Naive-MultiAgent-ReinforcementLearning. [Accessed: Nov. 27, 2024].

[5] Gymnasium Developers, "HalfCheetah-v4 — Gymnasium documentation," Gymnasium: MuJoCo Environments, [Online]. Available: https://www.gymlibrary.dev/environments/mujoco/half_cheetah/. [Accessed: Nov. 27, 2024].

[6] Gymnasium Developers, "LunarLander-v2 — Gymnasium documentation," Gymnasium: Box2D Environments, [Online]. Available: https://www.gymlibrary.dev/environments/box2d/lunar_lander/. [Accessed: Nov. 27, 2024].

[7] N. Barhate, "Actor-Critic-PyTorch," GitHub repository, [Online]. Available: https://github.com/nikhilbarhate99/Actor-Critic-PyTorch/tree/master. [Accessed: Nov. 27, 2024].

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint* arXiv:1509.02971, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971.

[9] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. [Online]. Available: https://doi.org/10.1016/0364-0213(90)90002-E.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

A Pseudocode for Agent Class

Algorithm 5 Agent Class

Require: Environment env, number of games n_games, training mode $training$, learning rates α, β, discount factor γ, soft update factor τ, batch size $batch\_size$, noise type $noise$, prioritized experience replay parameters per_alpha, per_beta

Initialize replay buffer $memory$ with maximum size
Initialize actor and critic networks
Initialize target actor and critic networks
Synchronize target networks with actor and critic
Initialize noise scheduler and beta scheduler
procedure CHOOSEACTION($observation$)
    Convert $observation$ to tensor $state$
    $action \leftarrow actor.forward(state)$
    if $is\_training$ **then**
        Add exploration noise to $action$
    end if
    Scale $action$ to environment limits
    return $action$
end procedure
procedure OPTIMIZE
    if Buffer size $< batch\_size$ **then**
        return
    end if
    Sample $state, action, reward, new\_state, done, weights, indices$ from memory
    Compute target Q values using target networks
    Compute temporal difference errors TD_errors
    Compute critic loss as mean squared error of weighted TD_errors
    Update critic network
    Compute actor loss as negative Q-values of critic for actor's actions
    Update actor network
    Update replay buffer priorities with TD_errors
    Soft update target networks using τ
    Increment optimization step counter
end procedure
procedure SAVEMODELS
    Save weights of actor, target actor, critic, and target critic
end procedure
procedure LOADMODELS
    Load weights of actor, target actor, critic, and target critic
end procedure
procedure ADDEXPLORATIONNOISE($action$)
    if $noise$ is 'normal' **then**
        Sample noise using noise parameter
        Add noise to $action$
    end if
    return $action$
end procedure
procedure ACTIONSCALING($action$)
    Scale $action$ from neural network output range to environment action range
    return Scaled $action$
end procedure