Blackjack RL: A Study of the Deep Actor-Critic and Deep Double Q-Learning Network Reinforcement Learning Networks

Alexis Bryan Ambriz, Suresh Ravuri, Sri Vinay Appari, Harshavardhan Valmiki

Computer Engineering Department
San Jose´ State University (SJSU)
San José, CA, USA
Email: {bryan.ambriz, srivinay.appari, suresh.ravuri, harshavardhanraghavendra.valmiki}@sjsu.edu

Abstract—This project extends traditional reinforcement learning agents to utilize deep reinforcement learning by enhancing the agent with both a double-headed Q-Learning neural network model (DDQN) with replay memory, as well as an actor-to-critic (A2C) agent - using the Gymnasium library in the Python programming language. Traditional preference algorithms such as Monte Carlo, Temporal Difference, SARSA, or Q-Learning learn the optimal preference strategy without requiring a neural network. We implemented advanced versions of these algorithms using deep reinforcement learning strategies involving a neural network agent to predict the optimal state-value and action-value functions by applying deep, double-headed Q preference and A2C learning algorithms within the reinforcement learning library Gymnasium's Blackjack environment using the neural network library Pytorch. We found that the A2C agent outperformed the DDQN agent with a roughly 100% improvement (35% to 18% wins) across 200 evaluation simulations. The A2C agent was also found to exhibit clearer decision boundaries.

Index Terms—Actor, Critic, Reinforcement, Learning, Network, Large-Language-Model, LLM, Blackjack, GPT

1 Introduction

Reinforcement learning's foundational principles via classical conditioning emerged from behavioral psychology research in the early 20th century, particularly through the work of researchers like Thorndike (1911) and Skinner (1938). Classical conditioning was first systematically studied by Ivan Pavlov in his experiments with dogs (1927), where he discovered that dogs would salivate not only at the sight of food but also at the sound of a bell that had been repeatedly paired with food presentation. While classical conditioning, demonstrated by Pavlov's experiments, showed how organisms learn associations between stimuli, operant conditioning—more closely related to modern reinforcement learning—demonstrated how behaviors are modified through consequences. In reinforcement learning, an agent learns optimal behavior through interactions with an environment, receiving rewards or penalties that shape its decision-making process.

The mathematical framework often used to

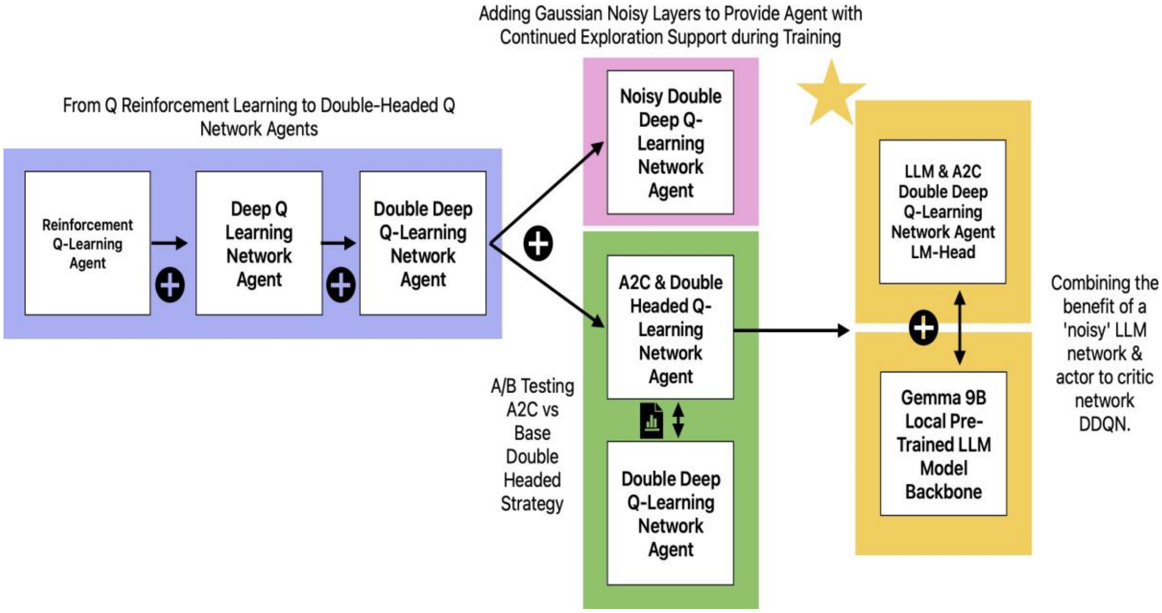

Fig. 1. Evolution of the blackjack learning network agent development from left-to-right: Q-learning to double deep q learning (DDQN) in blue, experimental noisy DDQN in pink, A/B testing of actor-to-critic (A2C) DDQN and previous DDQN in green. Future proposed work is the combined LLM-based A2C agent in yellow.

formalize reinforcement learning is the Markov Decision Process (MDP), a term first introduced by Richard Bellman in 1957 when he built upon Andrey Markov's earlier work on state transition probabilities to develop a framework for sequential decision-making under uncertainty. Traditional reinforcement learning algorithms such as the Monte Carlo, Temporal Difference, SARSA, and Q-Learning algorithms make use of the mathematical definition of an MDP. These algorithms allow an agent in an environment to learn a decision following strategy over time by applying linear state-value and action-value or Q-Learning functions to estimate the goodness of a state or action.

2 Problem Statement / Project Architecture

Traditional reinforcement algorithms and agents are inefficient and lack the ability to pick up complex patterns from an MDP environment, such as in Gymnasium's Blackjack and Atari environments. These learning algorithms (i.e. SARSA or the Q-learning strategies) use a strategy that utilizes state and action value expectation functions to influence the Q-learning algorithm. A limitation of these approaches is that these traditional algorithms use simple linear functions to modify the Q strategy matrix.

In deep reinforcement learning, neural networks emulate the strategy for the agent within the MDP framework to approximate the value functions via non-linear regression, allowing the agent to learn and make decisions in complex state-action mappings or high-dimensional state spaces. The Gymnasium and the agileRL libraries in Python are popularly used in both reinforcement learning and deep reinforcement learning. Gymnasium provides a variety of simple and complex action and state spaces or environments that can be used to contextualize an MDP, while agileRL provides a variety of neural network architectures and training strategies for training the deep reinforcement learning agent.

In Gymnasium’s CartPole environment using the Q-learning strategy, for example, the actions are discrete while the states are continuous and require discretization and binning to be able to map the states to actions in the Q-learning matrix.

Deep reinforcement learning provides more flexible, accurate, and efficient learning within Gymnasium’s more complex environments, or within custom environments. Traditional reinforcement learning algorithms that require converting the continuous states or actions into discrete values will inherently lose valuable information in the process. Deep learning via neural networks solves this problem as it is able to understand the more complex, continuous input. Our project aims to solve the learning problems posed in Gymnasium’s Blackjack and potentially more complex Atari environments to leverage the benefit of deep reinforcement learning.

3 METHOD(S) / SYSTEM DESIGN

The first step of our architectural building process was to set up a base reinforcement learning algorithm that utilizes a neural network. For this purpose, we built a Double Headed Deep Q-Learning Network (DDQN) agent. A deep Q-Learning network is similar to a basic neural network and has traditional elements (i.e. multiple dense layers, an activation function, an optimizer, etc) that are used to estimate the Q-Learning (action to state mapping) matrix that is used for reinforcement learning. A key difference is that the DDQN we implemented adds specific elements related to the reinforcement learning literature; it implements an off-policy learning strategy via two separate networks. As such, one network is used to learn the target Q-Learning strategy using the previous episode’s observations (using the experience replay buffer) during training, while the other is used to enact the policy during prediction or inference during the current observation. The current (target) network is able to use the learning (policy) network’s parameters by copying its’ state Dict.

An additional exercise we managed to include was to add noise layers to add noise to the output predictions of the target policy networks, based

```
class ImprovedDQNAgent:
    def __init__(self, input_dim=3, learning_rate=5e-4, gamma=0.99, epsilon=1.0):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.fitness = [] # used to store a series of 'avg reward' during evaluation

        # Larger network
        self.policy_net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        ).to(self.device)

        self.target_net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        ).to(self.device)

        self.target_net.load_state_dict(self.policy_net.state_dict())

        self.optimizer = optim.Adam(self.policy_net.parameters(), lr=learning_rate)
        self.memory = ReplayMemory(capacity=20000)
```

Fig. 2. Example of the Double-Headed Deep Q-Learning (DDQN) Network Agent.

on a learned distribution of Gaussian noise. In-including these layers has the potential to increase the “exploration” capability of a reinforcement learning model. By including noise to add randomization to the model outputs, the model can make “exploratory” decisions during training and after the typical exploration phase (when the epsilon value is close to 0) as well as post-training. The next and culminating step of

Noisy DDQN Architecture

```
import torch.optim as optim

class NoisyDoubleDQNAgent:
    def __init__(self, input_dim=3, learning_rate=5e-4, gamma=0.99, epsilon=1.0):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.fitness = [] # used to store a series of 'avg reward' during evaluation

        self.policy_noisy_linear1 = NoisyLinear(128, 64)
        self.policy_noisy_linear2 = NoisyLinear(64, 2)
        self.target_noisy_linear1 = NoisyLinear(128, 64)
        self.target_noisy_linear2 = NoisyLinear(64, 2)

        # Larger network
        self.policy_net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            self.policy_noisy_linear1, # the noisy layers w/o ReLU in-between
            self.policy_noisy_linear2,
        ).to(self.device)

        self.target_net = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU(),
            self.target_noisy_linear1,
            self.target_noisy_linear2
        ).to(self.device)
```

Fig. 3. Example of the Noisy Double-Headed Deep Q- Learning Network Agent.

our project was to use a different learning algorithm and compare their performance at learning the blackjack environment in Gymnasium. As previously mentioned, the DDQN implemented a Q-Learning strategy that is an off-policy strategy to ensure the agent is taking reasonable risks during training that would allow them to explore the environment without strictly adhering to the policy. The A2C learning strategy, on the other hand, is an on-policy & risk-averse learning strategy that ensures the model follows its policy during training & during prediction or inference. We built and trained a deep A2C network agent and then compared it to the DDQN, and found promising results.


Actor-Critic Network Architecture

```
class ActorCritic(nn.Module):
    def __init__(self, input_dim=3):
        super(ActorCritic, self).__init__()

        # Shared features extractor
        self.features = nn.Sequential(
            nn.Linear(input_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 128),
            nn.ReLU()
        )

        # Actor head (policy network)
        self.actor = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 2),  # 2 actions: hit or stand
            nn.Softmax(dim=-1)  # Output probabilities for each action
        )

        # Critic head (value network)
        self.critic = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)  # Single value output
        )

    def forward(self, x):
        features = self.features(x)
        action_probs = self.actor(features)
        state_value = self.critic(features)
        return action_probs, state_value

# Initialize the actor-critic network
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ac_model = ActorCritic().to(device)
optimizer = optim.Adam(ac_model.parameters(), lr=1e-3)
```

Fig. 4. Example of the Deep Actor-to-Critic Learning Network Agent.

4 EVALUATION

The evaluation of the reinforcement learning agents encompassed multiple dimensions of analysis to comprehensively assess their performance in the blackjack environment. Through systematic testing across 200 rounds, we observed significant performance differentials between the Actor-Critic (A2C) and Double Deep Q-Network (DDQN) implementations.

4.1 Quantitative Performance Analysis

The A2C agent demonstrated superior performance with a 35% win rate compared to DDQN's 18% win rate across the test rounds. This performance gap was further reflected in the average rewards, where A2C achieved -0.250 versus DDQN's -0.591. The substantial 48% decision disagreement rate between the agents indicates they developed markedly different playing strategies, suggesting distinct approaches to solving the same problem space.

4.2 Strategic Decision Making

Heat map visualizations revealed distinctive decision-making patterns between the agents. Fig. 5 shows the DDQN agent's learned strategy matrix, where the top chart uses blue hue blocks to indicate the probability of standing (red) versus hitting (blue) based on the dealer's card and the player's sum. The bottom chart highlights areas of strategic uncertainty through darker hues. The DDQN demonstrates more ragged decision boundaries, suggesting less consistent strategy formation.

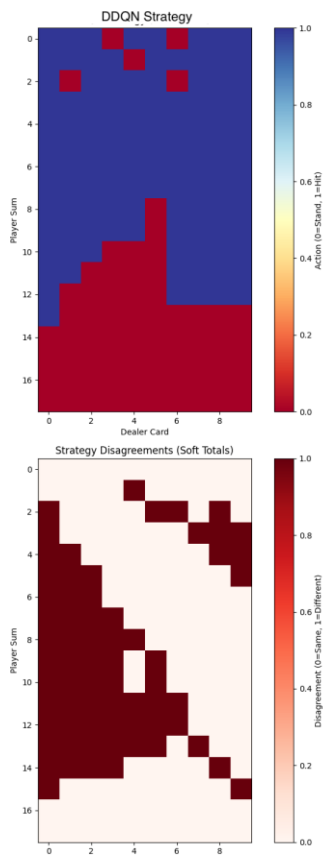

Fig. 5. Evaluation of the learned strategy of the DDQN agent. The charts at the top use blue hue blocks to indicate the chance to stand (more red) and the chance to hit (more blue) the dealer's card according to the player's or agent's sum. The charts at the bottom use darker hue blocks to indicate more disagreement. The DDQN shows more ragged decision boundaries.

In contrast, Fig. 6 presents the A2C agent's learned strategy matrix using the same visualization scheme. The A2C agent exhibits clearer and more consistent decision boundaries in its strategy matrix, particularly visible in the smoother transitions between hit and stand decisions. This manifested in more conservative play when holding lower value hands (12-16) against dealer's strong cards, potentially contributing to its superior performance. The clearer decision boundaries in the A2C's strategy map suggest more robust learning of fundamental blackjack principles.

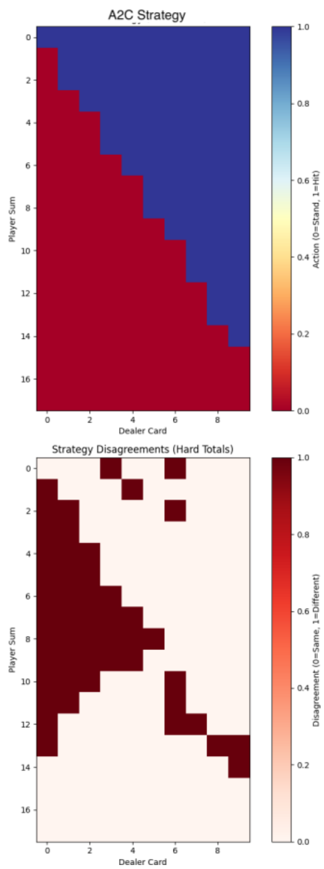

Fig. 6. Evaluation of the learned strategy of the A2C agent. The charts at the top use blue hue blocks indicate the chance to stand (more red) and the change to hit (more blue) the dealers card according to the player's or agents sum. The charts at the bottom use darker hue blocks to indicate more disagreement. The A2C shows more smooth decision boundaries.

The comparison between Fig. 5 and Fig. 6 illustrates a key difference in how these agents learned to play blackjack. While both agents learned basic stand/hit patterns, the A2C's smoother decision boundaries indicate more stable and generalizable strategy formation. This visual analysis helps explain the quantitative performance gap between the two approaches.

4.3 Learning Process Characteristics

Training stability emerged as a key differentiator between the two approaches. The A2C agent demonstrated more stable learning curves with reduced variance in performance metrics throughout the training process. While the DDQN agent maintained higher exploration rates through its epsilon-greedy policy, this did not translate to better overall performance. Both agents were trained for 500,000 episodes, with A2C showing superior sample efficiency in strategy development.

4.4 Robustness Assessment

The evaluation included comprehensive testing against varying dealer up-cards and player hand combinations to assess strategy consistency. The average final player sum of 19.7 indicates sound decision-making across different game states. Both agents were evaluated across the full spectrum of possible player hands, with A2C showing more consistent performance across varying game conditions.

4.5 Implementation Variations

The introduction of noise layers to the DDQN architecture presented interesting trade-offs. While these layers showed potential for extended exploration capabilities, they required longer training periods to achieve stability. The actor-critic architecture's fundamental separation of policy and value functions appeared to enable more refined strategy learning, particularly in handling the stochastic nature of the blackjack environment.

4.6 Performance Factors

The superior performance of the A2C architecture can be attributed to several key factors. First, the stability of policy updates through the critic network provided more consistent learning signals. Second, the architecture demonstrated better handling of the inherent randomness in card games. Finally, the improved ability to learn the long-term value implications of actions contributed to more effective strategy development.

4.7 Neural Network Evolution

The learning progression of both neural networks showed interesting patterns. The A2C agent's policy network evolved to develop clearer decision boundaries, particularly in challenging game states such as soft hands (hands with usable aces) and borderline-standing decisions. The DDQN's target and policy networks, while maintaining different exploration strategies, sometimes struggled to converge on optimal policies for these edge cases.

These evaluation results demonstrate that deep reinforcement learning can effectively develop sophisticated blackjack strategies, with the A2C approach showing particular promise for complex decision-making tasks in probabilistic environments. The clear performance advantages of A2C over DDQN suggest that actor-critic architectures may be better suited for similar card game environments where long-term strategy development is crucial for success.

5 RESULTS

Although we did not evaluate the noisy DDQN as compared to the DDQN, we noticed the noisy DDQN performed slightly worse than the original DDQN in terms of the training loss. This may be expected, however, as the noisy DDQN is including noise into the learned strategy and output predictions. In essence, it seems that the noisy layers may be used when training the model for more iterations where it might be necessary to extend the length of the exploration phase to avoid overfitting. As we did not train the agents for many iterations, we may not have had enough time to take full advantage of the noise layers.

We conducted an ablation experiment by evaluating the output of our previously trained blackjack playing DDQN as compared to the trained A2C model. We found that the A2C model outperformed the DDQN model at blackjack. After 200 simulated rounds of the game, the DQN model only won 18% of the games, whereas the A2C model won 35%. Similarly, the A2C outperformed the DDQN by achieving a higher average reward across the 200 rounds. The A2C model averaged a reward of -0.250 while the DDQN averaged a reward of -0.591. Finally, we compared them by measuring whether they agreed or differed in choices during their matches. We found that they had a decision disagreement rate of around 48% - implying that the models learned a very different strategy due to their difference in the learning algorithm.

6 CONCLUSION

This project successfully enhanced traditional reinforcement learning (RL) agents by incorporating deep reinforcement learning (Deep RL) techniques, specifically utilizing Double Deep Q- Learning Networks (DDQN) and Actor-to-Critic (A2C) models within the Gymnasium Blackjack environment. Our findings demonstrate that the A2C model outperformed the standard DDQN, achieving a higher win rate (35% vs. 18%) and a more favorable average reward (-0.250 vs. - 0.591) over 200 simulated games. This highlights the effectiveness of Actor-Critic methods in optimizing decision-making strategies in complex environments.

The introduction of noise layers in the DDQN aimed to improve exploration did not yield significant performance gains within the limited training period, suggesting that further training or hyperparameter adjustments may be necessary to realize their potential benefits. Additionally, the notable 48% decision disagreement rate between DDQN and A2C models indicates that different RL approaches can lead to distinct strategic behaviors, offering opportunities for hybrid strategies.

Integrating a Large Language Model (LLM) with the A2C framework represents an inno- vative advancement, though preliminary results indicate the need for further exploration to fully harness this integration.

Future work will focus on optimizing the noisy DDQN through extended training, ex- ploring additional Deep RL algorithms, apply- ing models to more complex environments, and deepening the integration of LLMs to enhance decision-making capabilities. Overall, this project establishes a solid foundation for advancing Deep RL agents, demonstrating significant improve- ments in strategy optimization and performance within the Blackjack environment.

REFERENCES

[1] I. P. Pavlov, *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*, Oxford University Press, 1927.

[2] E. L. Thorndike, "Animal intelligence: Experimental studies," *Psychological Review*, vol. 18, no. 2, pp. 117–137, 1911.

[3] B. F. Skinner, *The Behavior of Organisms: An Experimental Analysis*, Appleton-Century, 1938.

[4] A. A. Markov, "Extension of the limit theorems of probability theory to a sum of variables connected in a chain," *Halle*, 1906.

[5] R. Bellman, *Dynamic Programming*, Princeton University Press, 1957.

[6] R. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[7] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1989.

[8] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[9] Y. Wang, M. Riedmiller, and D. Silver, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv preprint arXiv:1511.06581*, 2016.

[10] J. Schulman et al., "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] V. R. Konda and A. G. Barto, "Actor-Critic Algorithms," in *Advances in Neural Information Processing Systems*, 1999, pp. 1008–1014.

[12] Gymnasium, "Gymnasium: A Next Generation OpenAI Gym," *arXiv preprint arXiv:2109.02277*, 2021. [Online]. Available: https://gymnasium.farama.org/

[13] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019.

[14] agileRL, "agileRL: Agile Reinforcement Learning Library," 2020. [Online]. Available: https://github.com/agile-rl/agileRL

[15] V. Mnih et al., "Asynchronous Methods for Deep Reinforcement Learning," *arXiv preprint arXiv:1602.01783*, 2016.

[16] Y. Wang and L. Jia, "Noisy Networks for Exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[17] A. Vaswani et al., "Attention is All You Need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, 2018.