Blackjack-GPT: An Actor-Critic Reinforcement Learning via LLM System

Bryan Ambriz, Sri Vinay Appari, Suresh Ravuri
Computer Engineering Department
San José State University (SJSU)
San José, CA, USA
Email: {bryan.ambriz, srivinay.appari, suresh.ravuri}@sjsu.edu

***Abstract*—The goal of this project is to extend traditional reinforcement learning agents to utilize deep reinforcement learning by enhancing the agent with a neural network model 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. Deep reinforcement learning strategies involving a neural network agent exist that can learn to predict the optimal state-value and action-value functions used within the Q preference learning algorithm. For this purpose we will use the reinforcement learning library Gymnasium, the neural network library Pytorch, and the deep reinforcement learning library agileRL in Python within the Google colaboratory computational environment. Thus, we aim to train a deep neural network to learn the optimal reinforcement learning strategy within Gymnasium's Blackjack or complex Atari game environments.**

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

I. 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 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.

II. 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.

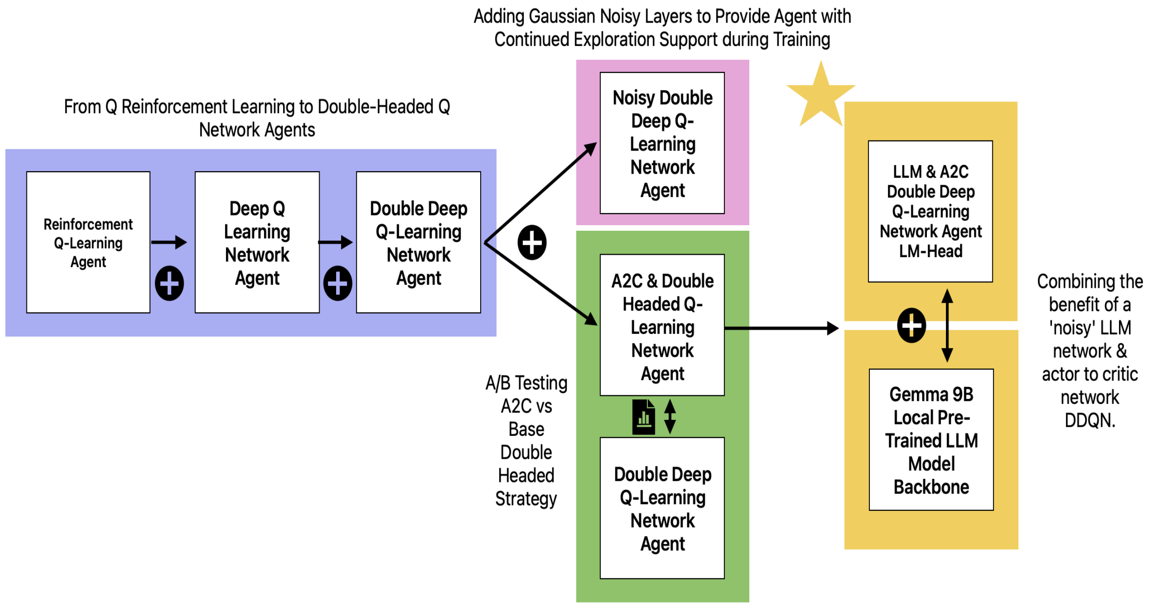
Adding Gaussian Noisy Layers to Provide Agent with Continued Exploration Support during Training
From Q Reinforcement Learning to Double-Headed Q Network Agents
Reinforcement Q-Learning Agent
Deep Q Learning Network Agent
Double Deep Q-Learning Network Agent
Noisy Double Deep Q-Learning Network Agent
A2C & Double Headed Q-Learning Network Agent
Double Deep Q-Learning Network Agent
A/B Testing A2C vs Base Double Headed Strategy
LLM & A2C Double Deep Q-Learning Network Agent LM-Head
Gemma 9B Local Pre-Trained LLM Model Backbone
Combining the benefit of a 'noisy' LLM network & actor to critic network DDQN.

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, and finally the combined LLM based A2C agent in yellow.

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.

./fig/fig2_rl_dqns_code.png

Fig. 2. Example of the code differences between the DDQN compared to A2C agent.

III. Method(s) / System Design

IV. Evaluation Methodology / Materials

V. Results

VI. Conclusion

References