EE-782 : Advanced Topics in Machine Learning Deep Recurrent Q-Learning for Partially Observable Markov Decision Processes

Vansh Kapoor
Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, India
200100164@iitb.ac.in

Sankalp Bhamare
Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, India
200110096@iitb.ac.in

Rohan Rajesh Kalbag
Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, India
20d170033@iitb.ac.in

***Abstract*—This project presents our unique implementation of Deep Recurrent Q-Learning (DRQL) that incorporates Transfer Learning for feature extraction, a customized LSTM for temporal recurrence, and a domain-informed reward function. This tailored approach aims to expedite convergence compared to the vanilla implementation outlined in the original paper. The performance evaluation focuses on two adaptive Atari 2600 games: Assault-v5 and Bowling, where game difficulty scales with player proficiency. Comparative analysis between the convergence of our optimized reward function and the vanilla version is conducted using StepLR and CosineAnnealingLR learning rate schedulers, complemented by theoretical explainations. Additionally, an efficient windowed episodic memory implementation employing bootstrapped sequential updates is proposed to optimize GPU memory utilization.**

***Index Terms*—Bootstrapping, Deep Recurrent Q-Learning, Flickering Atari Games, LSTM, OpenAI Gymnasium, Partially Observable Markov Decision Processes, Transfer Learning**

I. Introduction

Deep Q-networks: Undebated Queen for Deep RL

Deep Q-networks (DQNs) represent powerful frameworks for dynamically learning optimal policies in sequential decision processes characterized by a large number of states and actions. They achieve this by employing neural network-based feature extractors that leverage the property of state generalization, meaning that similar states share similar optimal value functions. DQNs utilize this property to generalize across states, allowing for effective learning and decision-making in complex environments with a multitude of possible states and actions.

DQNs have demonstrated their capability to acquire control policies at a level comparable to human performance across various Atari 2600 games. By learning policies that map directly from raw screen pixels to actions, these networks have consistently achieved state-of-the-art performance across a wide range of Atari 2600 games.

Why Deep Recurrent Q-networks?

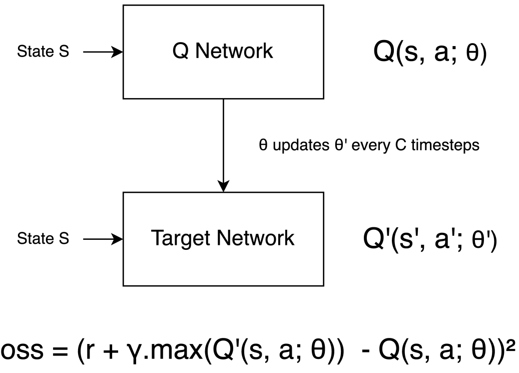
State S
Q Network
Q(s, a; θ)
θ updates θ' every C timesteps
State S
Target Network
Q'(s', a'; θ')
Loss = (r + γ.max(Q'(s, a; θ)) - Q(s, a; θ))²

Fig. 1. Double Q-Networks

However, Deep Q-Networks are limited in the sense that they learn a mapping from a **limited number of past states**, or game screens in the case of Atari 2600. In practice, DQN is trained using an input consisting of the last four states the agent has encountered. The game that requires a memory of more than four frames don't just depend on more than just DQN's current input, the process is **non-Markovian**. Instead of a Markov Decision Process (MDP), the game becomes a Partially-Observable Markov Decision Process (POMDP). There is however no notion of **temporal recurrence** in DQNs, and they do not work very efficiently in non-Markovian behaviour.

Real-world tasks often involve incomplete and noisy state information due to partial observability, this is seen in Atari 2600 games, given only a single screen, manifest as POMDPs. The efficacy of reinforcement learning can be improved by intoducing temporal recurrence in the form Recurrent-Neural-Networks. In the paper [1], the authors introduce Deep Recurrent Q-Network, a fusion of LSTM and a Deep Q-Network

II. Innovative Modifications Implemented

1) The proposal of a modified reward function incorporating a combination of Q-learning loss and episode duration, frequency of firing the weapon

2) The application of Cosine Annealing for learning rate scheduling aids the agent in **adapting to the escalating difficulty** across different levels in the gameplay.

3) The introduction of a windowed implementation for Episodic Memory, serving as input to the LSTM part of the double Q-learning network, for optimized memory utilization in the GPU.

4) Implemented **Transfer-Learning** by using **ResNet-18** pretrained on ImageNet **Transfer-Learning** for image feature extractors which speeds up convergence

III. Problem Formulation

The **Atari** arcade games are iconic forerunners in the realm of video gaming, that serve as enduring benchmarks for evaluating advanced reinforcement-learning algorithms. We implemented Recurrent Q-Learning on two such games: **Assault-v5** & **Bowling**. We employ LSTMs to capture state information, such as the velocity of the cannon in Assault V-5, by leveraging the **temporal information** embedded in the hidden state within the LSTM.

A. Assault V-5

- This game starts with an alien mother ship, that cannot be destroyed continually generating alien minion ships.
- There is a ground cannon which fires but with limited capacity, i.e., it cannot fire continuously due to overheating
- The cannon has a discrete action space of size seven and obtains a reward for destroying minion ships. It also has three lives which can be lost either by getting shot by alien ships or by overheating.

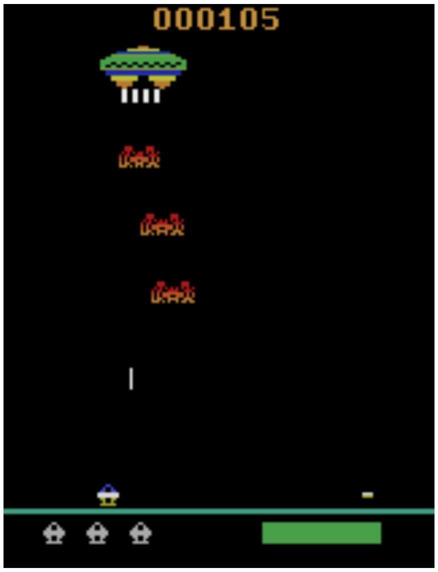

Fig. 2. A frame from the famous Atari Assault v-5 game

B. Bowling

- A Atari 2D bowling simulator game, with the goal to achieve the highest possible score by knocking down ten ”pins.” You have two attempts to knock down the pins during each turn, and there are a total of ten turns in each episode.
- The available actions for the agent are depicted in the Figure 4, where arrows indicate the direction for throwing the ball.

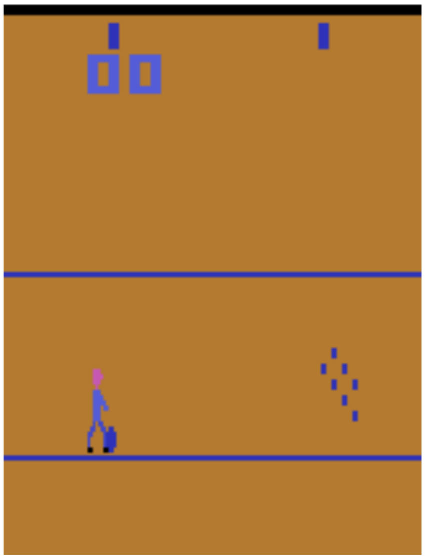

Fig. 3. A frame from the famous Atari Bowling game

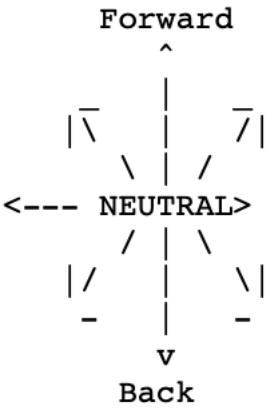

Fig. 4. Bowling Agent Actions

IV. Approach

Here we use **Transfer-Learning** by utilizing a ResNet-18 model as a feature extractor to get the state embedding. We utilize this embedding and pass it on to a Q-network. Q-Learning is a model-free off-policy algorithm for estimating the long-term expected return of executing an action from a given state. These estimated returns are known as Q-values. We utilize an ϵ greedy strategy to generate episodes and

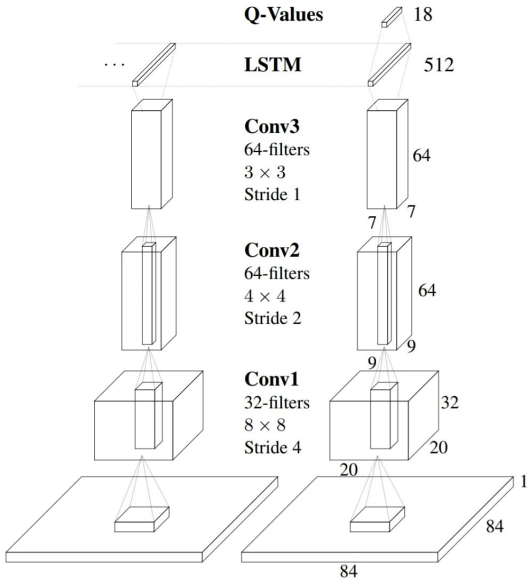

Fig. 5. LSTM-based Recurrent Deep Q-Learning

perform a **Double Deep Q-Learning update** based on the rewards given to us by the game-play environment.

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{a_0} Q(s_0, a_0) - Q(s,a) \right)$$

We initially perform a **square root decay** for the exploration factor ϵ to perform sufficient exploration during the initial iterations. We later perform a geometric decay to speed-up convergence to a purely greedy strategy.

A. Domain Knowledge for Reward Function Implementation

We utilize our domain knowledge, i.e., strategies based on the general human-play to allocate rewards to our agent. Here we utilize Vanilla Q-Learning loss, frequency of shooting and game-play time (episode length). This helps agent to survive in the game for longer time by preventing overheating due to continuous firing and also helps the agent select the ”shooting” action only when necessary. Hence by training on domain knowledge-based strategies, the agent tries to outperform human game play.

B. Rolling Window Optimization for Memory

The approach adopts the Sliding Window Optimization technique. Exploiting the principles of Long Short-Term Memory (LSTM) operations, the results from prior timestamp computations are applied to subsequent time steps. Specifically, computations for all frames, excluding the current one ($WINDOW\_SIZE - 1$ frames), are reused, resulting in an overall performance gain of x $WINDOW\_SIZE$. This method achieves significant memory savings, as deep memory instances now preserve a sequential role of frames rather than the entire window for each instance. Computed dynamically for each inference, this strategy not only optimizes computations but also conserves memory resources effectively.

V. Results

A. Assault V-5

Vanilla Rewards

Because the agent receives rewards solely based on its score, it explores a range of strategies in each episode, refining its approach through trial and error during policy improvement steps. As the agent lacks any incentives beyond the final score, the convergence towards the optimal policy occurs gradually. This **slow convergence** might require thousands of iterations to effectively beat strategies resembling those employed by humans.

Our Approach: Crafted Rewards

Figure 7 clearly demonstrates that the performance attained by the agent, utilizing rewards designed by us, significantly surpasses the performance achieved using vanilla rewards across both types of learning rate scheduling. In our approach, we craft rewards to encourage **optimal firing frequency** (preventing overheating), **prolonging gameplay**, and initiating horizontal motion only when necessary. This introduces **human-based gameplay rewards**, incentivizing the agent to emulate human gameplay and enhance it through Q-Learning.

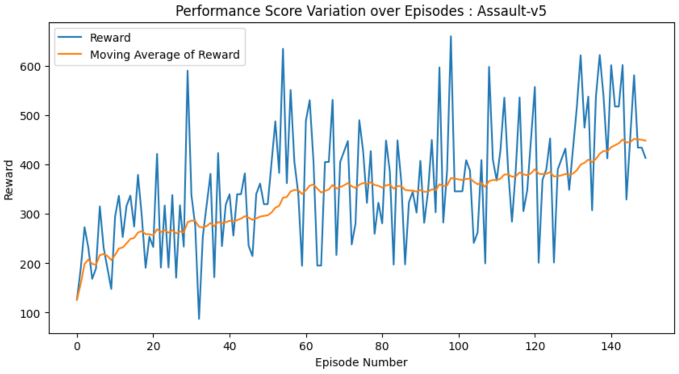

Fig. 7. Performance of RL Agent with iterations for Assault-v5

From Figure 6 we verify that our strategy indeed facilitates faster convergence: initially, the agent rapidly learns human-style gameplay, then refines this approach through Q-Learning updates to attain the optimal policy. Additionally, our observations indicate that cosine-annealing leads to superior long-term performance. As the gameplay difficulty escalates with the agent’s improved performance, **reverting to a previously higher learning rate** aids the agent in achieving optimal performance under heightened difficulty levels

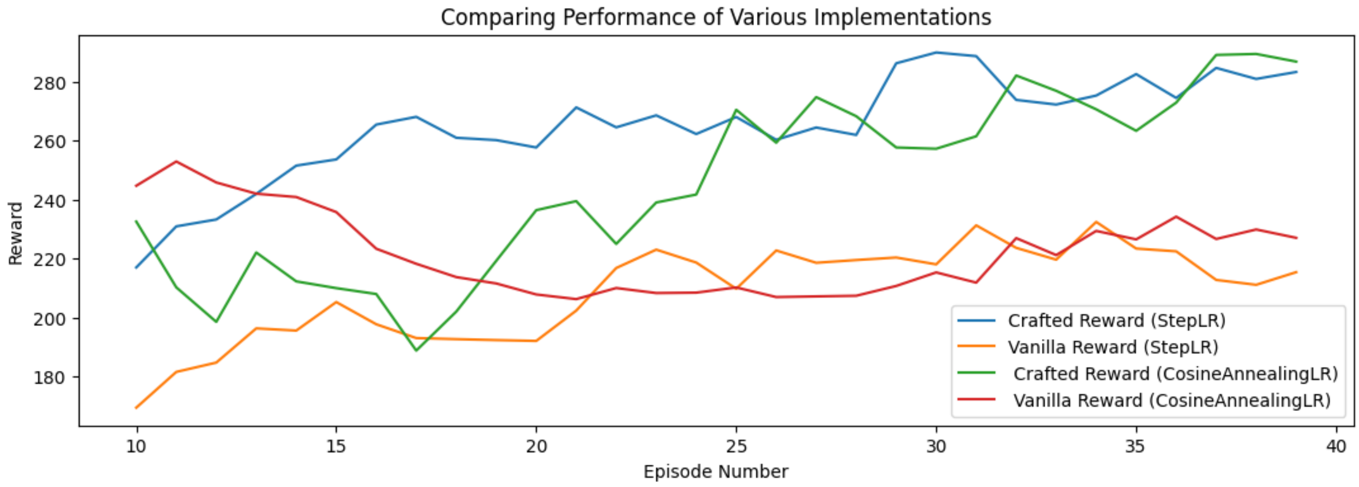

Fig. 6. Performance Evaluation of our Proposed Approach

B. Bowling V-5

Observing Figure 8 reveals that employing Vanilla rewards alongside the CosineAnnealing LR enhances the agent's performance but results in slower convergence for the same reasons outlined for Assault V-5.

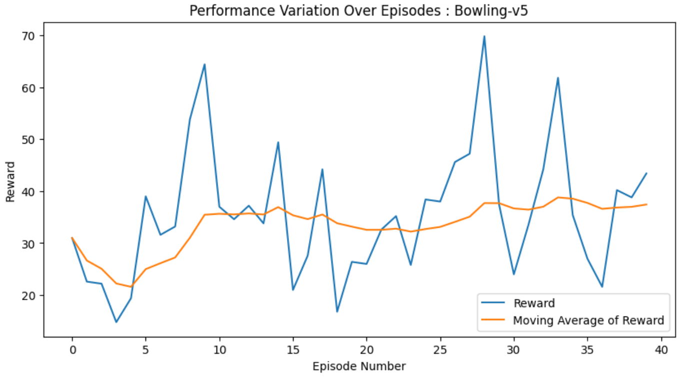

Fig. 8. Performance of RL Agent with iterations for Bowling

References

[1] M. J. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," *CoRR*, vol. abs/1507.06527, 2015. arXiv: 1507.06527. [Online]. Available: http://arxiv.org/abs/1507.06527.

[2] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. arXiv: 1509.06461. [Online]. Available: http://arxiv.org/abs/1509.06461.

[3] B. Bakker, "Reinforcement learning with long short-term memory," in *Advances in Neural Information Processing Systems*, T. Dietterich, S. Becker, and Z. Ghahramani, Eds., vol. 14, MIT Press, 2001. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2001/file/a38b16173474ba8b1a95bcbc30d3b8a5-Paper.pdf.

[4] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27, Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/8bb88f80d334b1869781beb89f7b73be-Paper.pdf.

[5] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan, "Optimality and approximation with policy gradient methods in markov decision processes," *CoRR*, vol. abs/1908.00261, 2019. arXiv: 1908.00261. [Online]. Available: http://arxiv.org/abs/1908.00261.