12.10.25

Atari Pong Reinforcement Learning

Group 6

Vicente Aguayo, Ella Bourassa, Shantanu Deshpande, Palina Karzhenka, Grace Lim, Paul Ling, Lingfeng Ren, Heqi Zhao, Sam Yin

Introduction

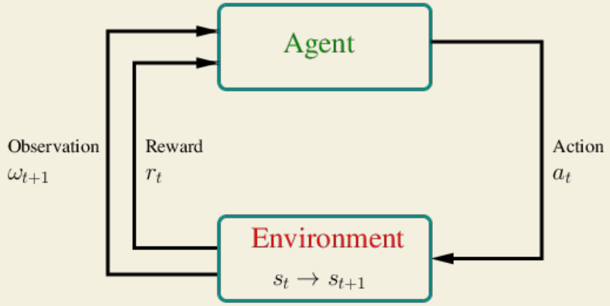
Project Objective:

Perform a comparative analysis on the performance of three different Pong agents to find the best approach

Our Goals:

1. Evaluate and compare the performance of different RL algorithms.
2. Analyze any trade-offs between final performance and learning stability

Background



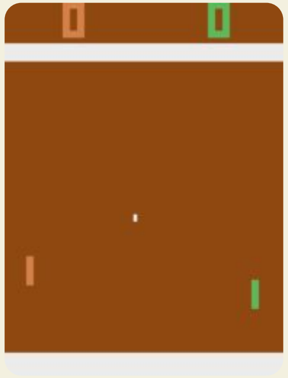
RL Paradigm Diagram[1]

What is **Reinforcement Learning**?

- A machine learning paradigm where an **agent learns by interacting with an environment**.
- Goal: **learn a policy**, which is a state to action mapping, that **maximizes cumulative reward**.

Why **Atari Pong?**

- Games can provide a **well-defined, complex environment** with clear rules.
- Pong has a **simple, discrete action space** and a **clear reward structure**.


ALE Pong[2]

[1] RL Paradigm Diagram: https://deeplearningmath.org/images/MDP_model.png
[2] ALE Atari Pong image: https://ale.farama.org/_images/pong1.gif

Methodology

SARSA Agent

Main idea: the SARSA agent learns by taking steps and **updating its knowledge base with the actual sequence of events**: state, action, reward, state, action (on-policy).

Key features for stable learning:

- **On-policy TD updates**: the Q-target uses the next action selected by the same ε-greedy policy.
- **Online learning without replay**: parameters update every step.
- **ε-greedy exploration schedule**: to balance exploration and exploitation.
- **Stacked input frames** (default = 4) allow the agent to infer ball velocity and direction.

SARSA Agent

Implementation Tools and Training Setup:

- **Gymnasium/ALE:** Atari Pong environment.
- **Custom PyTorch SARSA implementation**.
- Input observations were **84×84 preprocessed grayscale frames**, stacked according to the configured frame-stack size.
- **Rewards:** +1 for scoring, -1 for conceding, and 0 for all other transitions (with optional reward clipping enabled via config).
- **2500 timesteps across 2 hours, run on Mac M2 chip**

Gradient-Free Agent

Main idea: the gradient-free agent learns to play by using a **simple linear model**. Instead of using gradients, the agent improves by **trying linear softmax policies for each step**, **keeping the best ones**, and updating toward them.

Key features for stable learning:

- **CEM parameter search**: Every generation, sample a batch of policies, evaluate them in Pong, keep the top 20%, and update the mean/std of weights toward these top policies
- **Linear softmax policy:** The flattened features (2688 values) go into a single weight matrix, which outputs action probabilities with softmax
- **Feature preprocessing:** Each Pong frame is downsampled to 43x32 grayscale and combined with a frames difference image so the agent can detect motion.

Gradient-Free Agent

Implementation Tools and Training Setup:

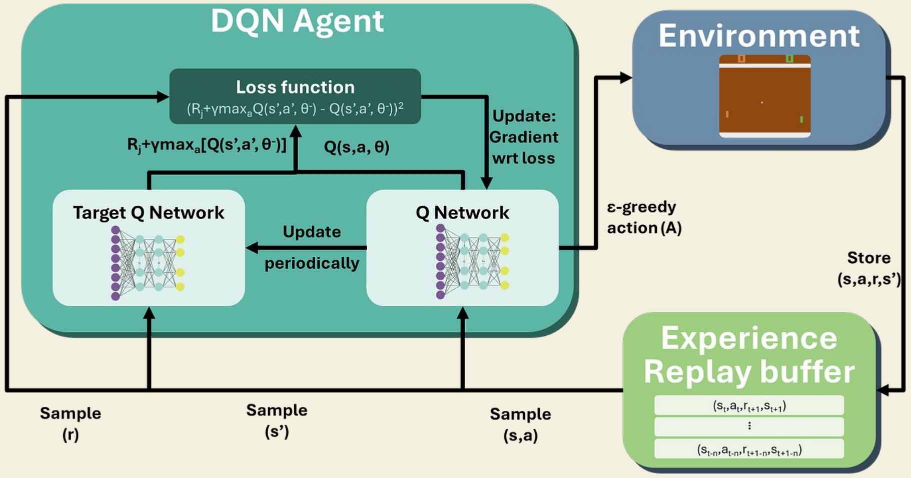
- **Gymnasium/ALE** provided the Atari Pong environment.
- **Custom CEM implementation in NumPy.**
- Input features were **42x32** downsampled grayscale pixel values plus frame differences concatenated with a bias term.
- **Rewards:** +1 for scoring, −1 for losing, 0 otherwise.
- **2500** training CEM generations, **87 million timesteps.**

DQN Agent

Main idea: the DQN agent learns to play by **estimating the value of each action**.

Key features for stable learning:

- **Experience replay** to prevent catastrophic forgetting.
- **Separate target network** to set stable learning targets.
- **4 consecutive stacked frames** as one input in order to understand ball's motion and speed.



DQN Agent
Loss function
$(R_j+\gamma \max_a Q(s',a',\theta^-) - Q(s',a',\theta^-))^2$
$R_j+\gamma \max_a[Q(s',a',\theta^-)]$
$Q(s,a,\theta)$
Update: Gradient wrt loss
Target Q Network
Update periodically
Q Network
Environment
ε-greedy action (A)
Store (s,a,r,s')
Experience Replay buffer
$(s_t,a_t,r_{t+1},s_{t+1})$
⋮
$(s_{t-n},a_{t-n},r_{t+1-n},s_{t+1-n})$
Sample (r)
Sample (s')
Sample (s,a)

Deep Q-network[3]

[3]https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQO2JiGEIAuitm-u0ONYO5w7WUQodDkO_9cLw&s

DQN Agent

Implementation Tools and Training Setup:

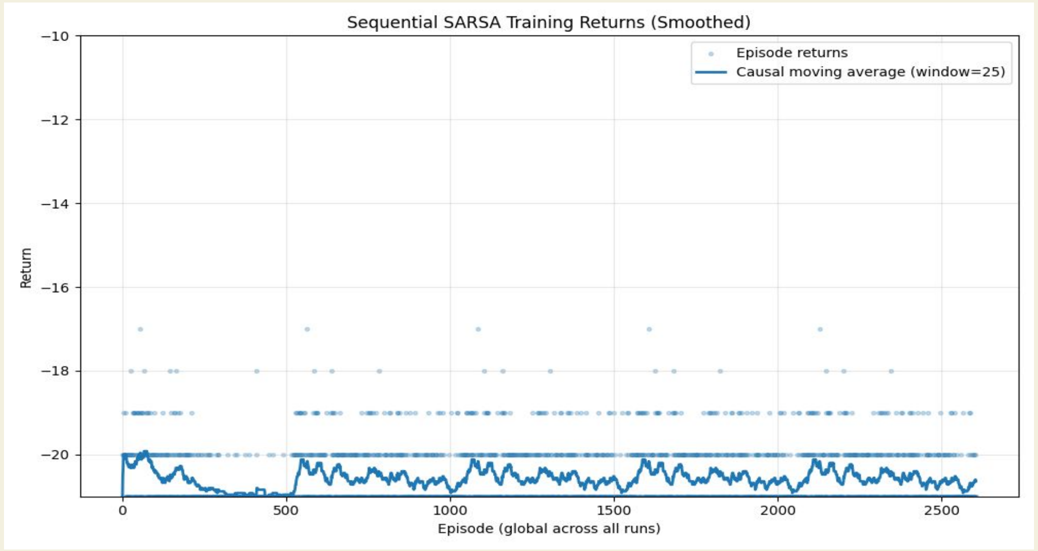
- **Gymnasium/ALE** provided the Atari Pong environment.
- **Stable-Baselines3** supplied the core DQN algorithm and configured the convolutional neural network architecture.
- Input was a series of **84x84 grayscale images**.
- **Rewards** are +1 for scoring, -1 for conceding, 0 otherwise.
- **Training data** was generated by the agent playing **2 million timesteps**.
    - This training was accomplished with **8 parallel environments** and GPU acceleration.

Results

SARSA Agent

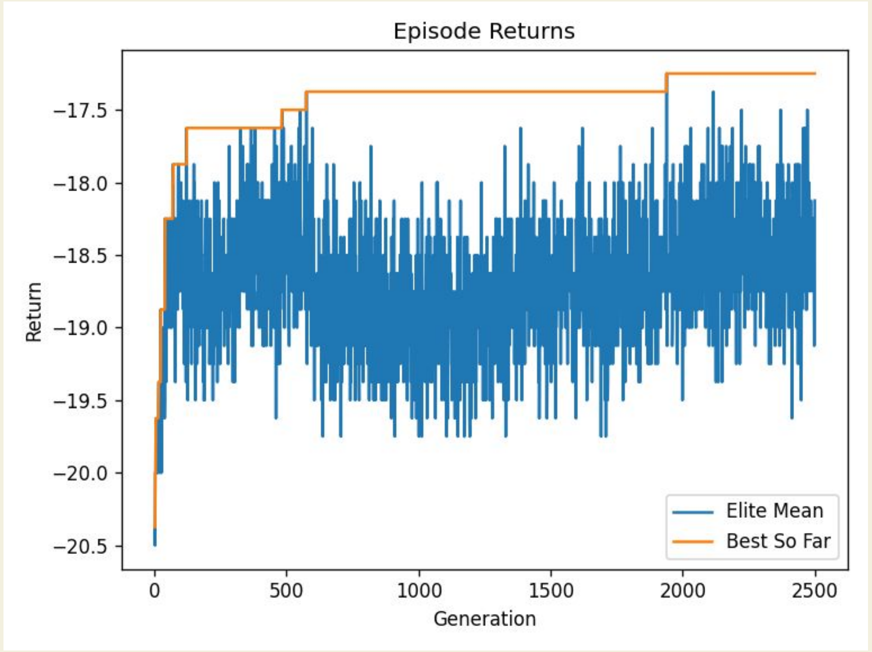
- Weakest performance among 3 models.
- Trained for 2,500 timesteps.
- Returns stayed negative throughout training.
- **No convergence** towards successful pong play.

- On policy means it's slow
- No experience replay
- High dimensional state space



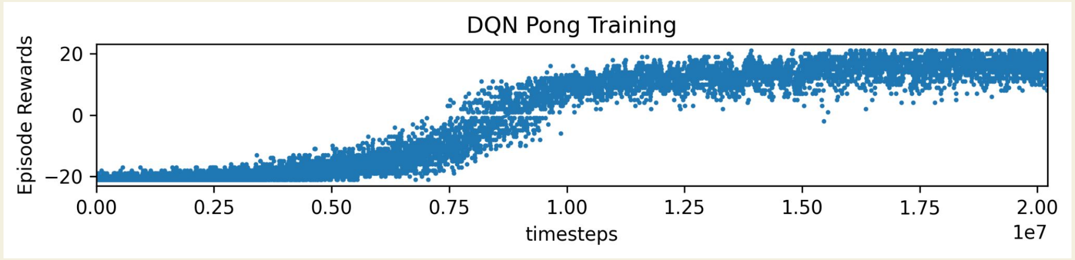
Gradient-Free Agent

- Slow improvement: Over 2500 generations, returns increased from roughly -20.5 to around -17.5, showing the agent found policies slightly better than random.
- Never reached positive performance: Unlike DQN, the agent never learned a winning strategy.
- Very long training time: Takes many hours for a full run since CEM evaluates many full Pong Games per generation.



DQN Agent

- Rapid Convergence: First ~500k timesteps exploring (low reward), followed by a learning curve where it mastered the game mechanics.
- Final performance: mean reward 21/21 (perfect play).
- Performance is relatively stable after optimal policy is learned.



DQN Agent Demo

Discussion & Conclusion

Discussion

Limitations

- Different training durations and processing power across agents could serve as a confound in our study and limits our ability to make direct comparisons.

Future Directions

- Standardize all applicable methods across agents, including relevant hyperparameters, training durations, and processing power.
- Have Pong agents play against each other, as another metric to determine winning agent type.

Conclusion

- Implemented **three reinforcement learning agents** to play Atari Pong.
- Project served as a **hands-on introduction** to RL concepts and training pipelines.
- Comparative experiments revealed **practical limitations** in some methods (e.g., slow or unstable learning).
- Results highlight that **algorithm choice isn’t enough**: architecture and design decisions heavily influence performance in complex, high-dimensional environments.

Thank You!

Q&A