3x4 Gridworld Problem with Deep Q-Learning

Kaito Namatame
Arizona State University
Email: knamatam@asu.edu

Code available at: https://colab.research.google.com/drive/1pOFe29VGXSXg6-mNYUjZcDYGNm28OVfw?authuser=2
The learning curves can be seen on the https://wandb.ai/gridworld_qlearning/gridworld_dqn_sweep_8

***Abstract*—This report presents a Deep Q-Learning (DQN) approach to solve a 3x4 gridworld task, where an agent learns to reach a goal while avoiding a lose state under stochastic transitions. A neural network approximates the Q-function using experience replay, a target network, and ϵ-greedy exploration. I describe the training workflow and analyze learning curves and hyperparameters, achieving an solid average evaluation reward.**

***Index Terms*—Deep Q-Learning, Gridworld**

I. Introduction

In the previous assignment, we implemented tabular Q-learning. In this work, we extend it to Deep Q-Learning (DQN) [1], which uses a neural network to approximate the Q-value function. The 3×4 gridworld is used as a benchmark where the agent learns to reach the goal while avoiding the lose and wall state. We analyze training performance and the effect of hyperparameters using learning curves.

II. Methodology

A. Environment Setup

The 3x4 gridworld has 3 rows and 4 columns, with a goal state at (3,2) (reward +1), a lose state at (3,1) (reward -1), a wall at (1,1), and a start state at (0,0). Non-terminal states cost a reward of -0.04. Actions (up, down, left, right) with 80% probability for desired directions, with 10% probability each for perpendicular directions. The agent stays in place if it hits a wall or boundary.

B. Data Generation

Training data is generated via experience replay. The agent collects transitions (state, action, reward, next state, done) under an ϵ-greedy policy and stores them in a buffer of 1,000. Randomly sampling minibatches of 64 ensures stable learning by breaking temporal correlations.

C. Neural Network Design

The Q-function is approximated by a fully connected neural network with:

- **Input Layer**: 12 neurons (one-hot encoded state for 3x4 grid).
- **Hidden Layers**: Two layers of 64 neurons each, with ReLU activation.
- **Output Layer**: 4 neurons (Q-values for each action).

D. Target Network

A separate target network, identical in architecture to the policy network, stabilizes training by providing consistent Q-value targets. The target network's weights are updated every 10 steps by copying the policy network's weights.

E. Training Algorithm

The DQN algorithm proceeds as follows:

Algorithm 1 DQN with Experience Replay

```
Initialize replay buffer $\mathcal{D}$ with capacity 1,000
Initialize policy network $Q(s,a;\theta)$ and target network $Q(s,a;\theta^-)$
Initialize Adam optimizer with learning rate $1 \times 10^{-3}$
for episode = 1 to 1500 do
    Reset environment to start state (0,0)
    Initialize episode reward $R = 0$, steps $t = 0$
    while not terminal do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select a with maximum Q-value: $a_t = \arg\max_a Q(s_t, a; \theta)$
        Execute $a$, observe reward $r$, next state $s'$, done $d$
        Store transition $(s, a, r, s', d)$ in $\mathcal{D}$
        Sample minibatch of $N$ transitions from $\mathcal{D}$
        Compute target: $y = r + (1-d) \cdot \gamma \cdot \max_{a'} Q(s', a'; \theta^-)$
        Update $Q(s,a;\theta)$ by minimizing Huber loss:
            $$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} \frac{1}{2}(y_i - Q(s_i, a_i; \theta))^2 & \text{if } |y_i - Q(s_i, a_i; \theta)| \le 1 \\ |y_i - Q(s_i, a_i; \theta)| - \frac{1}{2} & \text{otherwise} \end{cases}$$
        Increment $t$, $R \leftarrow R + r$
        if $t \mod 10 = 0$ then
            Update target network: $\theta^- \leftarrow \theta$
        end if
        $s \leftarrow s'$
    end while
    Decay $\epsilon \leftarrow \max(0.01, \epsilon \cdot 0.999)$
end for
```

F. Evaluation

The agent is evaluated over 100 episodes with $\epsilon = 0$ (greedy policy). Average rewards are computed and logged to W&B.

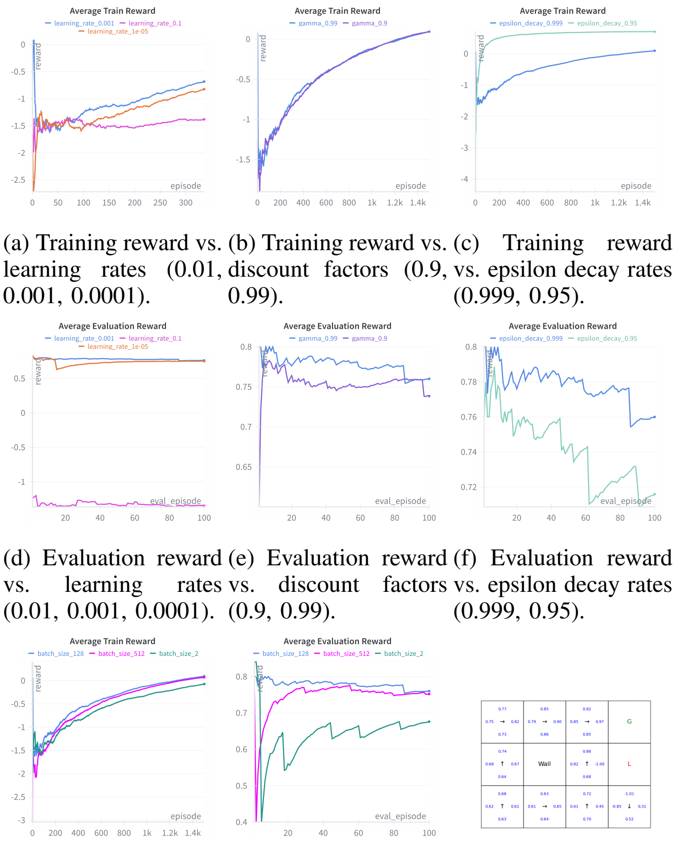
G. Hyperparameters

36 combinations of hyperparameters are tested:

- **Learning Rate**: {0.0001, 0.001, 0.1}
- **Discount Factor**: {0.9, 0.99}
- **ϵ Decay**: {0.95, 0.999}
- **Initial ϵ**: {1.0} (fixed)
- **Batch Size**: N = {2, 128, 512}

The selected hyperparameters are **α = 0.001**, **γ=0.99**, **ϵ decay=0.999**, batch size **N=128**, based on the highest average evaluation reward and policy stability, determined via analysis of learning curves.

III. Results and Discussion



(a) Training reward vs. learning rates (0.01, 0.001, 0.0001).

(b) Training reward vs. discount factors (0.9, 0.99).

(c) Training reward vs. epsilon decay rates (0.999, 0.95).

(d) Evaluation reward vs. learning rates (0.01, 0.001, 0.0001).

(e) Evaluation reward vs. discount factors (0.9, 0.99).

(f) Evaluation reward vs. epsilon decay rates (0.999, 0.95).

(g) Training reward vs. batch sizes (2, 128, 512).

(h) Evaluation reward vs. batch sizes (2, 128, 512).

(i) Learned policy with optimal hyperparameters.

Fig. 1: Comparison of DQN learning curves for different hyperparameters. Top row (a,b) shows the effect of learning rate on training and evaluation reward. Middle row (c,d) shows the effect of discount factor (γ). Bottom row (e,f) shows the effect of epsilon decay rate on training and evaluation performance.

- **Learning Rate**: A high learning rate (0.1) caused instability in training, as the Q-values fluctuated significantly due to large gradient updates. This led to oscillating rewards and a failure to converge as loss is getting bigger. In contrast, a low learning rate (1e-5) resulted in slow learning. The learning rate of 1e-3 achieved the best trade-off between stability and learning speed, yielding the highest evaluation reward.
- **Discount Factor**: From Fig. 1 (c), a lower discount factor (γ=0.9) led to faster but suboptimal learning, while a higher one (γ=0.99) caused some initial instability in training due to overemphasis on future rewards. However, the higher discount factor ultimately achieved higher average rewards in the evaluation phase and greater stability compared to the lower discount factor.
- **ϵ Decay**: With ϵ-decay = 0.999, the agent continued exploration for a longer period, resulting in slower improvement in the initial training phase, but achieved a higher average evaluation reward. In contrast, with ϵ-decay = 0.95, exploration decreased rapidly, leading to faster initial learning but premature convergence to a suboptimal policy and lower evaluation rewards. These results suggest that maintaining sufficient exploration, even in a small environment, contributes to more stable and effective policy learning.
- **Batch Size**: A small batch size (N=2) increases noise in gradient estimates, leading to unstable updates of the Q-value and lower reward, resulting in lower average evaluation reward, shown in Fig 1 (g). In contrast, a larger batch size provides a better average evaluation reward.
- **Hyperparameter Importance**: The learning rate is the most critical hyperparameter, as it directly affects Q-value convergence and stability. A high learning rate (0.1) caused instability, with large gradient updates leading to oscillating loss and poor evaluation rewards. In the small state space, excessive updates amplified errors, making it essential to carefully tune the learning rate for effective policy learning.

IV. Conclusion

This work successfully implemented Deep Q-Learning to solve the 3x4 gridworld problem, demonstrating that neural network function approximation can effectively generalize over a small but stochastic environment. Through systematic hyperparameter tuning and analysis, I identified the learning rate as the most influential factor in achieving stable convergence and optimal performance.

Future directions include exploring adaptive learning rate schedules, double DQN to mitigate overestimation bias, and applying similar techniques to larger and continuous state spaces. These extensions would further validate the scalability and robustness of the DQN framework beyond discrete grid-based environments. networks for improved learning.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.