# Implementation of a Deep Q-Network for a $3 \times 4$ Stochastic Grid World

Aman Chandak
achan160@asu.edu

***Abstract*—This report presents the implementation and evaluation of a Deep Q-Network (DQN) for solving a stochastic grid world navigation problem. The agent learns to navigate a $3 \times 4$ grid environment with a blocked cell, a penalty state, and a goal state. The environment features stochastic transitions, requiring the agent to learn a robust policy. DQN successfully learns the optimal policy to maximise the cummulative reward.**

## I. Workflow Summary

This section summarizes the complete workflow, from environment setup to agent evaluation.

### A. Environment and Data Generation

The environment is a $3 \times 4$ grid with 12 discrete states. The state $(r, c)$ is one-hot encoded into a 12-dimensional vector for the network. Data is generated interactively as the agent explores. At each step, the agent selects an action $a$, and the stochastic environment returns a next state $s'$, a reward $r$, and a terminal flag $done$. This transition tuple $(s, a, r, s', done)$ is stored in an Experience Replay buffer.

### B. Neural Network Design

The Q-function is approximated by a fully-connected neural network. I have tested two architectures, with the main experiment using 64 neurons per hidden layer.

- **Input Layer:** 12 neurons (for the one-hot state vector).
- **Hidden Layer 1:** 64 neurons (or 32, for comparison).
- **Activation 1:** ReLU (Rectified Linear Unit), which acts as a non-linear thresholding function.
- **Hidden Layer 2:** 64 neurons (or 32, for comparison).
- **Activation 2:** ReLU.
- **Output Layer:** 4 neurons (one for the Q-value of each action: N, E, S, W).

The network weights are initialized using the PyTorch default (Kaiming uniform initialization for linear layers).

### C. Training Algorithm

The training process (Algorithm 1) implements a standard DQN with two key stabilization features:

**1. Target Network:** A separate target network, $Q_{\text{target}}(\theta^-)$, is used to generate the target Q-values. Its weights are a frozen copy of the main policy network $Q_{\text{policy}}(\theta)$ and are updated (synchronized) only every 250 episodes. This provides a stable target for the loss calculation.

**2. Minibatch Training:** Once the replay buffer contains at least 500 transitions, training begins. In each training step, a random minibatch (size 16, 32, or 64) is sampled from the buffer. This breaks the temporal correlation between consecutive experiences.

The policy network's weights $\theta$ are updated by minimizing the Mean Squared Error (MSE) loss:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')} \left[ (Y - Q_{\text{policy}}(s, a; \theta))^2 \right] \quad (1)$$

where the target $Y$ is defined as:

$$Y = r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) \quad (2)$$

### D. Stopping Criteria

The training process has two main stopping criteria:

- **Per Episode:** An episode terminates if the agent reaches a terminal state (goal or penalty) or if it reaches the maximum step limit of 50 steps.
- **Total Training:** The entire training run stops after completing a total of 2500 episodes (or 1000 for hyperparameter tests).

**Algorithm 1:** DQN Training (from `rl_3.py`)

```
Initialize policy $Q(s, a; \theta)$ and target $Q(s, a; \theta^-)$;
$\theta^- \leftarrow \theta$;
Initialize replay memory $\mathcal{D}$ with capacity 20,000;
for episode = 1 to 2500 do
    Reset environment, $s \leftarrow (2, 0)$;
    for step = 1 to 50 do
        Select action $a$ via $\epsilon$-greedy strategy;
        Execute $a$, observe $r, s'$, terminal flag;
        Store $(s, a, r, s', terminal)$ in $\mathcal{D}$;
        $s \leftarrow s'$;
        if $|\mathcal{D}| \geq 500$ (start_learn) then
            Sample minibatch (size 32) from $\mathcal{D}$;
            Compute target $Y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$;
            Update $\theta$ by minimizing $(Y - Q(s, a; \theta))^2$;
        end
        if terminal then
            break;
        end
    end
    Update $\epsilon = \max(0.05, \epsilon \times 0.997)$;
    if episode % 250 == 0 then
        $\theta^- \leftarrow \theta$;
        (Update target network)
    end
end
```

## II. Hyperparameters

The hyperparameters used in the main experiment are summarized in Table I.

TABLE I
DQN HYPERPARAMETERS (MAIN EXPERIMENT)

| Parameter | Value |
|---|---|
| Learning rate ($\alpha$) | $1 \times 10^{-3}$ |
| Discount factor ($\gamma$) | 0.99 |
| Replay memory size | 20,000 |
| Mini-batch size | 32 |
| Target network sync frequency | 250 episodes |
| Training episodes | 2500 |
| Max steps per episode | 50 |
| Start learning threshold | 500 transitions |
| $\epsilon_{start}$ | 1.0 |
| $\epsilon_{end}$ | 0.05 |
| $\epsilon$ decay factor | 0.997 (multiplicative) |
| Hidden layers | 2 (64 neurons each) |
| Optimizer | Adam |
| Gradient clip norm | 1.0 |

### *A. Hyperparameter Determination*

The chosen hyperparameters are a combination of common starting points and empirical tuning, verified by the comparison tests.

- **Optimizer, $\alpha$, $\gamma$:** Adam with $\alpha = 1 \times 10^{-3}$ is a robust default. $\gamma = 0.99$ is standard for tasks with moderately long horizons.
- **Replay/Batch Size:** A large buffer (20,000) ensures diverse experiences. A batch size of 32 was tested against 16 and 64 and found to provide a good balance of stability and performance.
- **Epsilon Decay:** An exponential decay from 1.0 to 0.05 over $\approx 2300$ episodes ensures a long exploration phase.
- **Network Size/Target Update:** A 2-layer network with 64 neurons was compared against 32 neurons. While the comparison run was short, the main run's 64-neuron network achieved near-perfect results. The target update frequency (250 episodes) provides a stable learning target.

## III. RESULTS AND EVALUATION

Evaluation is performed by analyzing the learning curves and Q-function visualizations saved by the Python script.

### *A. Training and Evaluation Curves*

The primary learning curves from the 2500-episode training run are shown in Figure 1.

- **Episodic Return (Moving Average 50):** The agent begins with highly negative rewards (avg. -1.0 to -1.5), indicative of random exploration leading to the step limit or penalty. A clear positive trend begins around episode 250. The MA(50) reward climbs steeply and stabilizes around episode 1000 at a high positive value of approximately 0.7, indicating the agent has learned an efficient path to the goal (Goal reward 1.0, minus step costs).
- **Training Loss (MSE):** The loss (smoothed with MA 200) shows an initial drop, followed by a large spike between 5000 and 15000 training steps. This spike corresponds to when the agent discovers the high-reward goal state, which creates a large (but correct) Bellman error. The loss then trends down as these new high values are propagated through the network, stabilizing around 0.0125.
- **Success Rate (MA 50):** This curve provides the clearest metric. The success rate starts at 0, begins a sharp climb around episode 250, and reaches $\approx 90\%$ by episode 500. It quickly achieves a near-perfect success rate (98-100%) and remains there for the rest of the 2500-episode run, demonstrating a robust and optimal learned policy.

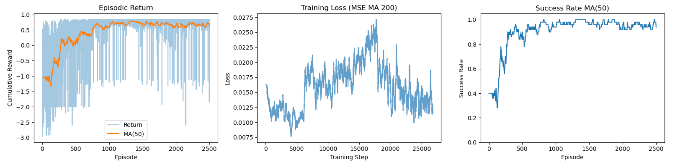

Fig. 1. Main training and evaluation curves (Episodic Return, Training Loss, and Success Rate) from the 2500-episode run.

### *B. Learned Q-Function*

Figure 2 visualizes the learned Q-values for each of the four actions in every state. This analysis confirms the agent learned the correct optimal policy.

- **Optimal Path:** The agent's learned policy is clearly visible. For instance, in the 'Action: E' (East) heatmap, the Q-value at state $(0, 2)$ is 0.95, correctly identifying the final move to the goal. In the 'Action: N' (North) heatmap, the Q-value at $(1, 2)$ is 0.91, guiding the agent up and around the penalty.
- **Penalty Avoidance:** The most critical learned behavior is at state $(1, 2)$, which is adjacent to the penalty at $(1, 3)$. The 'Action: E' heatmap shows a large negative Q-value ($-0.63$) for this state, correctly identifying this as a catastrophic move. The agent has learned to strongly prefer moving North (Q-value 0.91) from this state to avoid the penalty.
- **Obstacle Avoidance:** The agent correctly assigns values to the states around the 'WALL' at $(1, 1)$, such as the high value (0.75) for moving North from state $(2, 1)$.

### *C. Hyperparameter Sensitivity*

The comparison plots in Figure 3 (run for 1000 episodes) justify the hyperparameters chosen in Table I.

- **Learning Rate:** The 'lr=0.01' curve is erratic and converges to a much lower reward, indicating instability. 'lr=0.0001' learns very slowly but steadily. The chosen 'lr=0.001' (orange) shows a large initial dip but recovers with the steepest learning curve, converging quickly to a high reward.
- **Discount Factor:** 'gamma=0.9' (blue) learns fastest but plateaus at the lowest reward, as it is too "short-sighted." 'gamma=0.95' and 'gamma=0.99' (the chosen value)

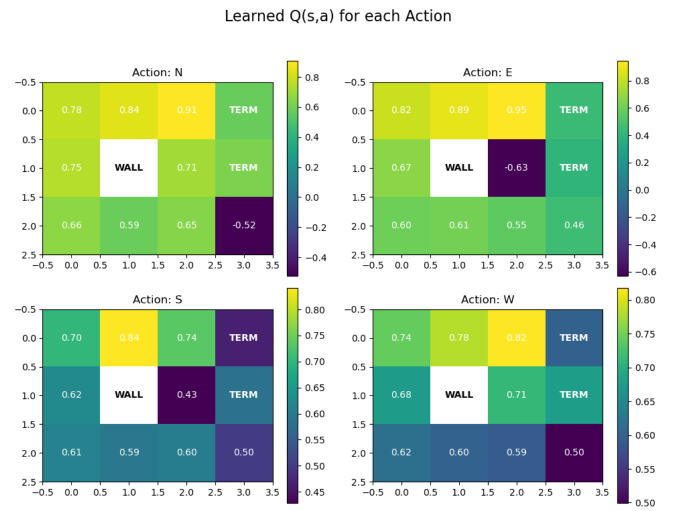

Fig. 2. Heatmap of the learned Q-values, $Q(s, a)$, for each of the four actions.

converge to a higher reward, as they correctly value the long-term goal.

- **Batch Size:** ‘batch_size=64‘ (green) learns noticeably slower. ‘batch_size=16‘ (blue) and ‘batch_size=32‘ (orange) perform similarly, with ‘32‘ being slightly more stable, making it a good default.
- **Network Size:** In this shorter run, ‘hidden_size=32‘ (blue) converges faster and to a higher reward than ‘hidden_size=64‘. This suggests that for this simple 12-state environment, a smaller network is sufficient. However, our main run with 64 neurons achieved a perfect success rate, proving it is also a highly effective choice.

## IV. Code Verification

The complete, runnable Python code used for this report is provided in the Code Github. The python file, rl_hw2.py. inside RL_assignment_2 folder, includes all environment logic, the DQN algorithm, and the functions necessary to reproduce the results and save all plots as PNG files for verification.

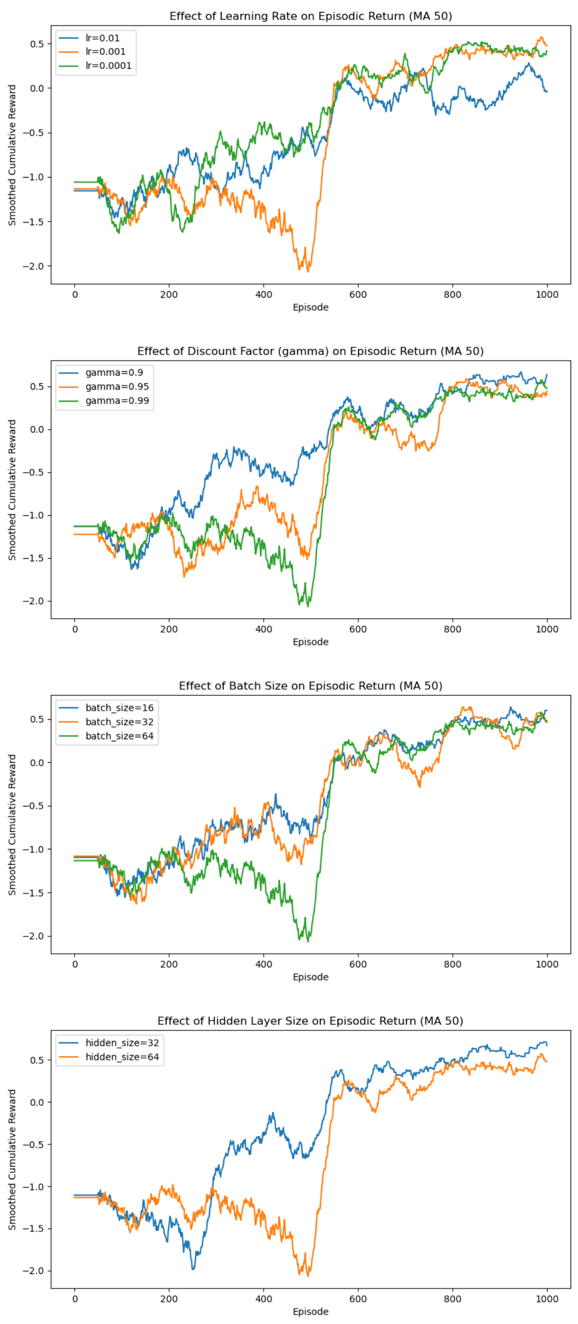

Fig. 3. Hyperparameter sensitivity plots. From top to bottom: Learning Rate, Discount Factor, Batch Size, and Hidden Layer Size.