Implementation of Deep-Q-Network(DQN) on a CartPole Environment

Deepak Somesh Kumar Jyothi [1]

Abstract

This study is about the implementation of the DQN on the CartPole Environment using target network (TN) and experience replay (ER), to learn to balance the pole. We also explore the role of loss function in Q-Learning, deep learning techniques, and conduct experimental analysis through an ablation study and comparative evaluations.

1. Theoretical Background

In the context of deep Q-Learning, The loss function is derived based on the Bellman equation and Q-Learning update rule. The key idea is to minimize the differences between expected current value and the target Q-value from the Bellman equation. Bellam equation for optimal Q-values: The optimal state-action value function $Q(s, a)$ satisfies the bellman optimality.

1.1. Q-Learning Loss Function:

- **Bellam equation for optimal Q-values:** The optimal state-action value function $Q(s, a)$ satisfies the bellman optimality. (Mnih, 2013)

$$Q(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q(s', a') \mid s, a\right]$$

Here s is state, a is action, r is reward, γ is discount factor, s' is next state, a' is next action.

In practice, we are not aware of the optimal Q-values, so Q-Learning updates the estimate $Q(s, a)$ based on the observed transitions (s, a, r, s'). (Sutton & Barto, 2018)

- **The Q-Learning update rule as follows:** The rule is derived from the bellman optimality equation. For transition the $Q(s, a)$ is updated as -

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Here α is learning rate (between 0 and 1), which controls the step size update. $r + \gamma \max_{a'} Q(s', a') - Q(s, a)$, represents the temporal difference (TD) error. TD measures the difference between the current estimate of $Q(s, a)$ and the target value $r + \gamma \max_{a'} Q(s', a')$. (Sutton & Barto, 2018)

We use a neural network as a function approximator to represent $Q(s, a)$ with a parameter θ (e.g, weights in a neural network). We define a loss function that minimizes the squared TD error. The function approximation $Q(s, a, \theta)$ is defined in TD target as:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta)$$

Then the TD error is $TD_{error} = y - Q(s, a; \theta)$

Now the loss function would be :-

$$L(\theta) = \mathbb{E}[(y - Q(s, a; \theta))^2]$$

$$i.e, L(\theta) = \mathbb{E}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]$$

This is the mean squared error (MSE) loss function, that updates θ by minimizing the difference between predicted and the target Q-Values.

The loss function in Q-learning is introduced from the Bellman equation, leading to the squared TD error. It helps in training neural network (especially in DQN) to approximate the Q-value $Q(s, a)$. (Sutton & Barto, 2018)

The motive of training process is to find the network parameters θ that minimize this loss function over a batch of transitions that are sampled from the experience replay using gradient decent.

$$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$$

1.2. Why Tabular Update Rules Cannot Be Used?

In the high-dimensional state space, it is infeasible to store Q-values for every state-action pair in a table. To implement efficient learning in the complex environments, we prefer function approximation using neural networks to generalize across the states. In this study, we are diving into the implementation of the DQN on CarPole Environement,

Algorithm 1 DQN with TN and ER

Initialize: Q-Network
Initialize: Target-Network T
Initialize: Replay-Buffer D
repeat
  Initialize state s.
  Select action a using ϵ-greedy policy from $Q(s, a; \theta)$
  Execute action a, observe reward r, next state s'
  Store transition (s, a, r, s') in D
  Sample mini-batch of transitions from D
  Target: $y = r + \gamma * \max(Q'(s', a'; \theta^-))$
  Compute Loss: $L(\theta) = (y - Q(s, a; \theta))^2$
  Update θ using gradient decent
  Every S steps: $\theta' \leftarrow \theta$ (Update Target network)
until *step* is reached $maxsteps$

while experimenting on different approaches(Naive Learning, Learning with only Target network(TN), with only Experience Replay(ER), with both TN and ER).

2. Experiments and Results

we implement DQN on the CartPole environment and observe the return (cumulative reward). The plot shows the mean reward of 10^6 steps as a function of environment steps. The agent initially explores randomly but gradually learns to balance pole, reaching near-optimal performance.

2.1. Implementation of Q-Learning

Initially we start training the agent with naive q-learning(no TN or ER) and observe the results with fixed parameters as below.
$Config : learningrate = 0.001, gamma = 0.99,$
$epsilon = 1.0, epsilondecay = 0.995,$
$epsilonmin = 0.01, numenvs = 5,$
$maxsteps = 10 * *6, stepinterval = 10000,$
$targetupdate = 1000, batchsize = 512,$
$replaybuffersize = 500000$

Optimal Performance: The agent achieves the optimal performance 1, but the learning curve shows some instability due to the lack of a target network(TN) and experience replay(ER).

Before we jump on implementing the TN and ER. we want to find best values for model to acheive better performance. we are going to conduct ablation study over the naive Q-learning algorithm wit hyperparameters.

2.2. Ablation Study for Hyperparameters

The key hyperparameter we are going to target are learning rate, network size, update-to-data ratio, exploration factor.

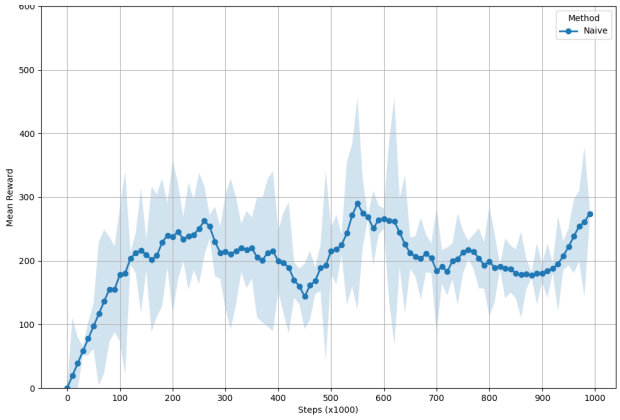

Figure 1. Naive Q-Learning with fixed Hyperparameters

Learning Rate: After considering the different values of learning rates(0.001, 0.0005, 0.01) with remaining fixed hyperparameters as shown in 2.1. we train the agent for upto 5 repetitions and observe the mean rewards.

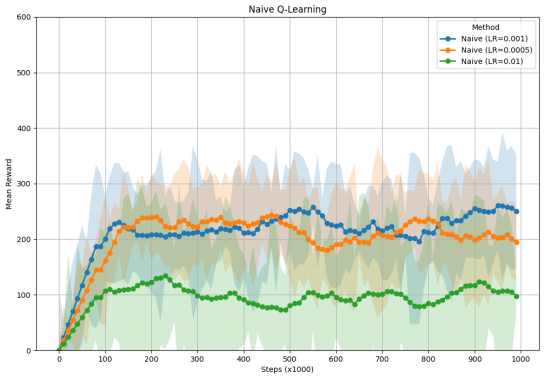

Figure 2. Naive Q-Learning, ablation study on Hyperparameter(learning rates)

We observe that the learning rate of 0.001 acheives the optimal and better learning curve.2

Exploration Factor(Epsilon): For exploration we are considering to perform ablation study on $EpsilonDecay$ (0.999, 0.995, 0.99) with learning rate of 0.001 (Observation from previous study).

As shown in Figure 3, We observe that with Epsilon Decay of 0.995 achieves the balanced results between exploration and exploitation. For 0.999, we observe prolonged exploration and slower convergence, For 0.99 it shows rapid exploitation.

Update-to-Data Ratio: The UTD ratio controls how many

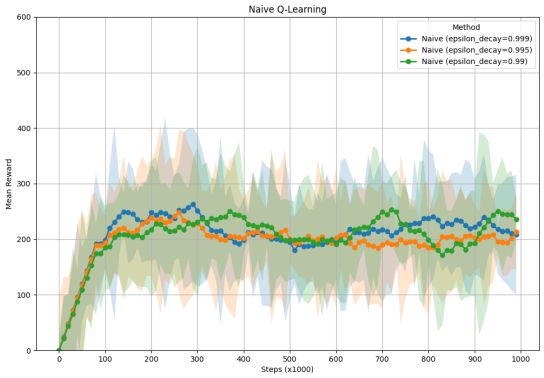

Figure 3. Naive Q-Learning, ablation study on Hyperparameter (epsilon decay)

times the model updates its parameters per environment steps, which affects the learning efficiency and stability.

we are considering to train our agent with ratios(1:1. 1:4, 1:10).

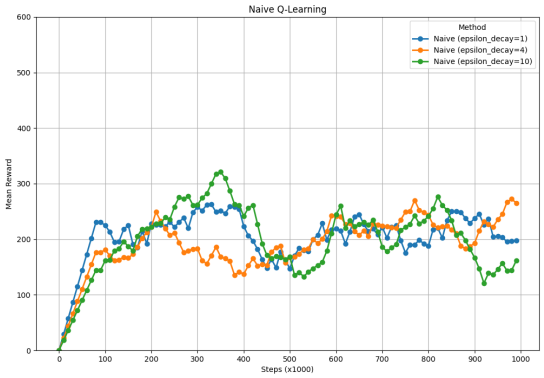

Figure 4. Naive Q-Learning, ablation study on update-to-data ratio.

The higher the ratio, the faster learning, the downside is the risk of overfitting. For Lower ratio it is more stable, but slower learning. From our study, for each step (1:1) is lot slower but stable, Update for every 4 steps (1:4) is optimal and stable, Update for every 10 steps (1:10) is quick, but unstable learning. 4

Network Size: Considering the optimal hyperparameters we have (Learning rate, Exploration factor, Update-to-data ratio), its time for us to find the optimal network size with optimal capacity and balanced performance.

Consider the different sizes (small(128 units), medium(256

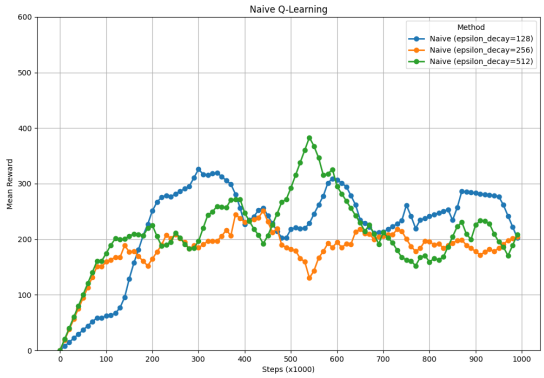

Figure 5. Naive Q-Learning, ablation study on network size

Table 1. Optimal Hyperparameter values

HYPERPARAMETER	VALUE
LEARNING_RATE	0.001
EPSILON_DECAY	0.995
UTD RATIO	1:4
NETWORK_SIZE	256

units), large(512 units)) of the network. we find the size with medium have the good balance between capacity and performance, whereas in small, has limited capacity and does not achieve optimal performance. The large size leads to overfitting, which concludes that medium(256 units) is the better choice.

3. Adding Target Network and Experience Replay

We have observed that in naive approach of Q-learning, the training of the agent is unstable thus, lead to poor performance. The **Target Network** is a copy of the main Q-network, but its parameters are updated less frequently (e.g, every few thousand steps), this reduces the moving target problem observed in naive and stabilizes the training. (Mnih et al., 2015) (Sutton & Barto, 2018) The **Experince Replay** maintains the replay buffer to store transitions (s, a, r, s', done), which then randomly samples experiences from the buffer, breaking the correlation between consecutive updates and leading to stable training. (Mnih et al., 2015) (Sutton & Barto, 2018)
we are going to consider the optimal hyperparameters 1 achieved from the ablation study to observe different variants of DQN.

In the variant with TN and ER, we achieved the best results so far, compared to the other variants(Naive, Only TN, Only

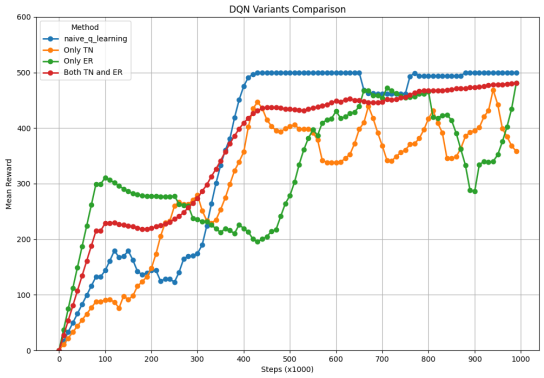

Figure 6. DQN Implementation on all the methods (Naive, TN, ER, TN and ER)

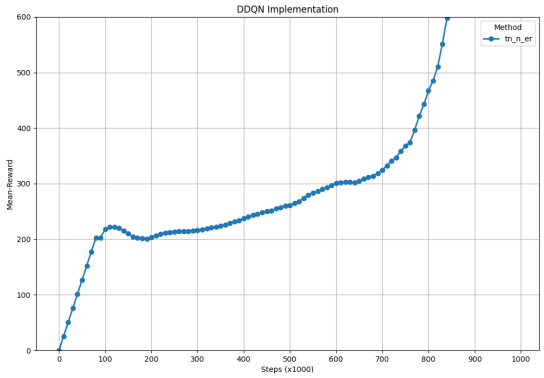

Figure 7. Implementation of Double DQN

ER). we can observe stable learning in DQN method with TN and ER.

- **Naive Q-Learning:** Unstable, Slow Convergence
- **Only TN:** Improved stability, Slow Convergence
- **Only ER:** Some instability, Faster Convergence
- **Both TN and ER:** Best performance, Stable, Faster Convergence

4. Double DQN Implementation(DDQN)(BONUS)

The DDQN is introduced to reduce the over-estimation bias. Instead of using the target network to select the action and evaluate the Q-value, we use local network to do the same. (Van Hasselt et al., 2016)

- In original DQN approach the target Q-value is:

$$target = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

- In DDQN(Double), the target Q-value is:

$$target = r + \gamma Q(s', \max_{a'} Q(s', a'; \theta^-); \theta^-)$$

  Here, θ is the local network, and θ^- is the target network.

The agent achieves higher and more consistent rewards compared to standard DQN 7

Here, we implement the DDQN into the experiment. We considered the final variant (with TN and ER) for the implementation.

5. Conclusion

In this study, we have implemented and evaluated the DQn on the CartPole Environment. We demonstrated the key importance of the target network and experience replay in stabilizing training and improving the performance. We have highlighted the sensitivity of DQN to Hyperparameters through ablation study and the need for careful tuning. Overall, DQN with both TN and ER achieved the best results.

References

Mnih, V. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.