Reinforcement Learning Assignment 2-Value Based

Wei Chen [1] **Yijie Lu** [1] **Jialiang Wang** [1]

1. Introduction

1.1. Explanation of the techniques: DQN

The essence of the Q table is to store the state-action pair and the disadvantage is obvious, it can only deal with the problem of finite dimensions, when the dimension becomes larger, the space which occupies will become sharply larger. For example, in the case of Atari Games, the state space is a picture, considering the length and width, RGB three channels, and the numerical range of each channel, the dimension can be very large, and in fact this Q table contains very redundant information, because similar images may correspond to similar values. It is not good in terms of learning efficiency or space occupation. We can use deep neural network as function approximation, replacing the traditional Q table.Learning with function approximation allows 1) for a compact representation of the solution (usually in the form of a policy or value function, which can be stored in memory in approximate form) and 2) generalization, where similar states will share information automatically. Moreover, function approximation is also a practical solution for tasks with continuous action spaces, like frequently encountered in robotics.

As an optimization of the Q-learning method, DQN is also an off-policy method, that is, the interaction strategy and the learning strategy are separated. There are also two key technologies in the specific implementation of DQN, Replay Memory and Target Q networks, which greatly improve the performance of the network.

1.2. Replay Memory

Since neural networks have a more complex structure, they need to use a large amount of data to do small-step updates. Therefore using only single-step trials for optimization wastes both data and cannot achieve sufficient updates. In addition, since the reinforcement learning problem is a sequential decision, there is a strong correlation between the data at each step, which is also not conducive to the training of the generalizability of the neural network. To solve these problems, DQN proposes an approach: Replay Memory. Specifically, the experience fragment obtained from each experiment is recorded, we only need (s, a, R, s') to update the network,, so each such sequence is cut down as an experience, put into the experience pool, and when the network is trained, a random fetch is taken from it. It not only ensures the amount and diversity of training data, but also allows the experimental data to be reused.

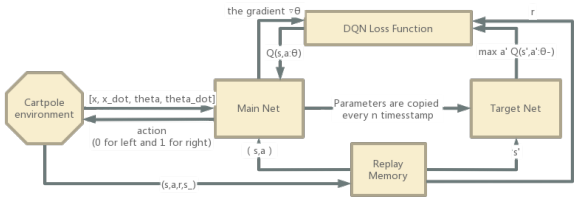

Figure 1. The network architecture of this assignment

1.3. Target Q networks

In the Q table, the Q values corresponding to different (s, a) are independent of each other and therefore can be updated separately. However, in a neural network, the change of parameters may affect all the Q values corresponding to (s, a).

$$\begin{aligned} \text{Loss} &= \text{target}\, Q - \text{ currentQ} \\ &= R + \gamma Q\left(s', a'\right) - Q(s, a) \end{aligned} \tag{1}$$

$$a' = \underset{a}{\text{argmax}} Q\left(s', a\right) \tag{2}$$

As can be seen from the loss function, while updating the network parameters, TargetQ is changing. This will lead to unstable update targets. To solve this problem, DQN proposes a way to fix Q targets,using a new network to generate the a'. For example, the main network is denoted as Q with argument θ, we introduce a copy of another network Q^- as Q, whose parameters are initialized to $\theta^- = \theta$. The Q network still interacts with the environment as mentioned earlier, gets feedback, and uses the feedback to update the network parameters, but the difference is that when updating, Q^- is used to generate the target value:

$$\begin{aligned} \text{Loss} &= \text{target}\, Q - \text{ current } Q \\ &= R + \gamma Q^{-}\left(s', a'\right) - Q(s, a) \end{aligned} \tag{3}$$

[1]Leiden University.. Correspondence to: Jialiang Wang <s2829746@vuw.leidenuniv.nl >.

$$a' = \underset{a}{\operatorname{argmax}} Q^{-}(s', a) \tag{4}$$

This guarantees target Q is a fixed value for a period of time. After that , the parameters of the note Q are updated to θ_1, and then $\theta^{-} = \theta_1$ can complete the update Q^{-}. Q^{-} here is not an old version of Q, due to the design of Replay Memory, each time the data is fetched randomly, the update direction has a certain diversity, don't worry about Q^{-} will make Q regress, they are more likely to learn for different (s, a) updates.

1.4. The loss function and network architecture

During training, the weights in the neural network are updated to make the model perform better on the training data. Within a certain number of episodes, improvements on the training set are positively correlated with improvements on the test set. However, sometimes the training data will begin to overfit, and further "improvements" will result in reduced generalization performance. We design functions which terminate the training before overfitting occurs, and monitor tracks the quantity used to decide whether the training should be terminated. Also, we present the loss function in the following text.

$$L_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim D}(\underbrace{r + \gamma \max_{a'} Q\left(s', a'; \theta_i^{-}\right)}_{\text{target}} - Q(s, a; \theta_i)) \tag{5}$$

2. Implementation and experiment design

For implementation, Tensorflow is leveraged to build a network which has three layers of fully connected networks: (1) 4·16 (2) 16·32 (3) 32·2, where the parameter initialization is *he_uniform* (It draws samples from a uniform distribution) The loss function is *MSE*, and the optimizer is *Adam*. The basic environment of the game is: A pole which is attached by an un-actuated joint to a cart. The goal is to prevent the pole from falling over by applying a force (+1/-1) to the cart. In our implementation, the game is considered over if the pole falls (-1 bonus), or if it is successfully pushed 500 times (+10 bonus). Within 500 pushes, the bonus is +1.

There are primarily four experiments we made:

- Distinct learning rate (0.01, 0.001 and 0.0001)
- Distinct gamma γ (0.95 and 0.85)
- Distinct policy (ϵ-greedy, Boltzmann exploration and annealing ϵ-greedy)
- Comparison between pure DQN, DQN with replay buffer, DQN with target model, and DQN with both target model and replay buffer.

Selection of (hyper)parameters of baseline						
Policy	Lr	γ	Buffer size	Batch size	ϵ	Freq
ϵ-greedy	0.0001	0.95	5000	128	0.1	50

Table 1. Selection of (hyper)parameters of baseline, where Lr represents the learning rate, buffer size is the size of the experience replay buffer and Freq indicates that every 50 times the original (or evaluation) model is learned, the parameters of the target model are updated once. ϵ and γ are the (hyper)parameters for policy ϵ-greedy, and for calculating the target Q value.

where the (hyper)parameters of the baseline for our experiments is as Table 1 shown. The performance of the baseline is as Fig. 2 shown.

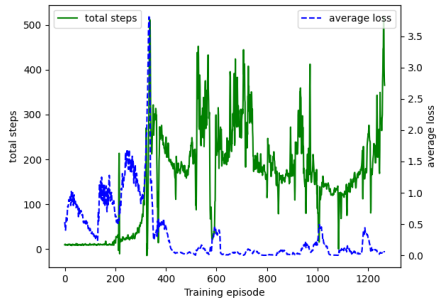

Figure 2. Baseline, where learning rate is set to 0.0001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

3. Experiments and results

We consider the criterion for a good model (or a good result) is that the average number of steps of the last 100 episodes is more than 200. In this way (early-stopping), it avoids over-fitting and catastrophic forgetting to some extent. In some degree, it also saves time, as we find that running 1200 episodes on our device takes more than 200 minutes. Also, there are chiefly four experiments made and the results and corresponding analysis are as follow:

(1) Learning rate:
The performance on different learning rate are presented in Fig.2 (baseline), Fig.3 (learning rate is 0.01) and Fig.4 (learning rate is 0.001), respectively.
We compare the model effect at different learning rates: 0.01, 0.001, and 0.0001. When the learning rate is 0.0001, the model training process is more stable, considering total steps and average-loss. When the learning rate is set as 0.01, at the beginning the total steps once reached 500, but the

learning rate, in other words, weight update, is too large, and model performance has not returned to optimal later.

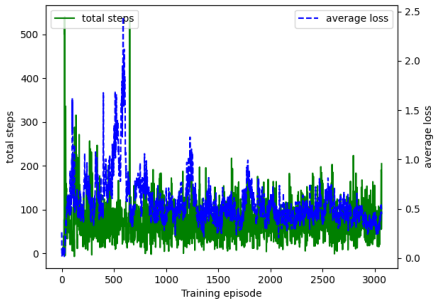

Figure 3. Experiment on learning rate, where learning rate is set to 0.01, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

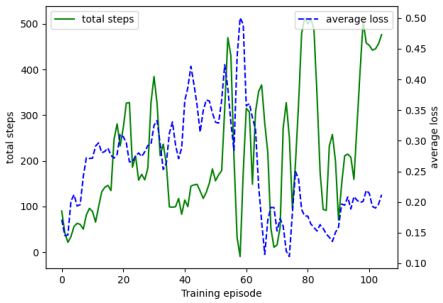

Figure 4. Experiment on learning rate, where learning rate is set to 0.001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

(2) Gamma:

When we test the performance under different value of gamma, we can see the results presented in Fig.2 (baseline) and Fig.5 (gamma is 0.85). Under conditions where the other hyperparameters are same as the baseline, the value of gamma equals to 0.85 and 0.95 were tested, 0.95 performs better because the higher the gamma value, the more we want the agent to focus more on the future.

(3) Exploration strategy:

The results of experiments on different policies are shown in Fig. 6 (not baseline), Fig.7 (annealing ϵ-egreedy) and Fig.8 (Boltzmann), where the selection of the (hyper)parameters for ϵ-egreedy is not the same (but similar) as that of the baseline. Exploration plays key role in reinforcement learning. We implement ϵ-greedy, Boltzmann exploration, annealing

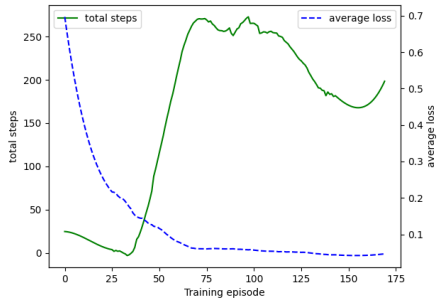

Figure 5. Experiment on gamma, where learning rate is 0.001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, buffer_size is 5000, gamma is 0.85. Every 50 times the original model is learned, the parameters of the target model are updated once.

ϵ-greedy for training the agent. ϵ-greedy suits this task better.

In contrast to total-steps, the total-steps of -greedy increase with episodes, there is a shock around 1000 times but can regain the correct action, and steps can reach about 200-250, as for Boltzmann exploration, the temperature parameter is set as 1, after 750 episodes, steps fluctuate around 150, however, with the increase of episodes, total-steps has a tendency to decrease. Annealing ϵ-greedy doesn't learn a good strategy in the early stages until 1000 episodes.
In contrast to average-loss, -greedy converges to around 0.1 after 600 episodes, and the fluctuation range seems smaller than others, Boltzmann exploration converges to about 0.5. Annealing -greedy performs the worst, it still does not converge after 1200 episodes.

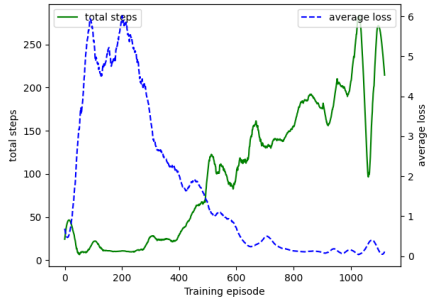

Figure 6. Experiment on policy, where learning rate is 0.0001, policy is ϵ-greedy, ϵ is 0.1, batch size is 32, size of buffer is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

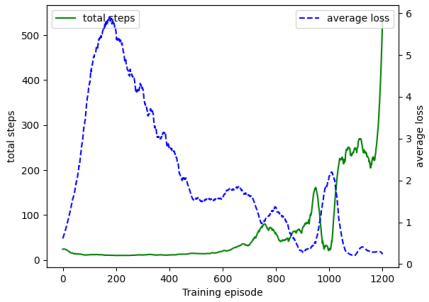

Figure 7. Experiment on policy, where learning rate is 0.0001, policy is annealing_ϵ-greedy, ϵ is 1.0 (where ϵ_min=0.01, ϵ_decay=0.9995), batch size is 32, buffer size is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

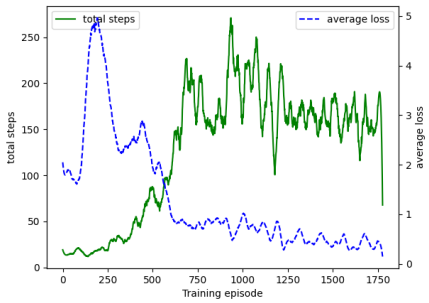

Figure 8. Experiment on policy, where learning rate is 0.0001, policy is Boltzmann, temp is 1, batch size is 32, buffer size is 5000, gamma is 0.95. Every 50 times the original model is learned, the parameters of the target model are updated once.

(4) Ablation study:

The performance are as Fig.2 (DQN with both target model and experience replay buffer), Fig.9 (DQN with only experience replay buffer), Fig.10 (DQN with only target model) and Fig.11 (pure DQN without both target model and experience replay buffer) shown.

Compare DQN with DQN-ER:

To alleviate the problems of correlated data and non-stationary distribution, we need replay memory, the size of replay memory is called replay buffer, If the replay buffer is too small, it will bring instability to the training. Because when the replay memory is full, it overwrites the oldest experience with the latest experience, that is, the storage mechanism is first-in, first-out. This will cause the agent to lose the experience of the initial training, lead to instability of the later training. After few attempts,we set the replay buffer as 5000. With experience replay removed, model struggles to learn the right strategy, and after 1200 episodes, total steps still fluctuate between 9 and 11.

Compare DQN with DQN-TN:

Although both models have oscillations, overall, DQN performs a little better, with the total step falling more in range of 300-400. In the early stages of training, the average-loss of DQNTN has always been higher, and the DQN has better stability.

Compare DQN with DQN-EP-TN:

As you can see in the results, DQN has higher learning speeds and better performance.

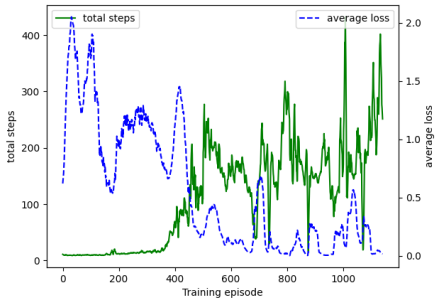

Figure 9. Experiment on comparison between pure DQN, DQN with replay buffer, DQN with target model, and DQN with both target model and replay buffer, where learning rate is set to 0.0001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95, and the target model is removed.

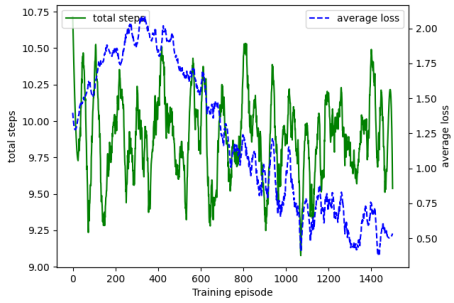

Figure 10. Experiment on comparison between pure DQN, DQN with replay buffer, DQN with target model, and DQN with both target model and replay buffer, where learning rate is set to 0.0001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95, and the experience replay buffer is removed.

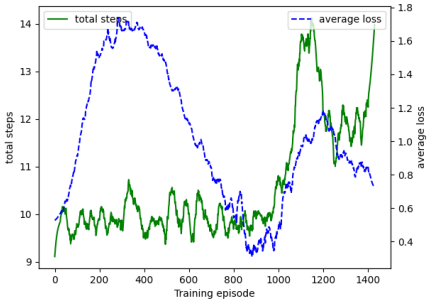

Figure 11. Experiment on comparison between pure DQN, DQN with replay buffer, DQN with target model, and DQN with both target model and replay buffer, where learning rate is set to 0.0001, policy is ϵ-greedy, ϵ is 0.1, batch size is 128, size of buffer is 5000, gamma is 0.95, and both the experience replay buffer and the target model are removed.

4. Conclusions

In conclusion, we implement a deep reinforcement learning model DQN, and demonstrate its ability to master difficult control policies for Cartpole, using only a four-dimensional vector as input.

Through experiments we come to the following conclusions (1) learning rate = 0.0001 performs better than 0.01 and 0.001. When the learning rate is set too large, the optimal value will be skipped, which will lead to instability and poor performance.(2) gamma = 0.95 performs better than 0.85 because the higher the gamma value, the more the agent focus on the future. (3) Among 3 exploration strategies : ϵ-greedy, Boltzmann exploration, annealing ϵ-greedy, ϵ-greedy suits this task better.(4) Our implement combines stochastic minibatch up-dates with experience replay memory to ease the training of deep networks and target networks to stable update targets, and demonstrates the effectiveness of these two components based on ablation experiments. Compared among pure DQN, DQN with replay buffer, DQN with target model, and DQN with both target model and replay buffer, DQN with both target model and replay buffer can accomplish more total steps, and the training process is more stable.

To avoid catastrophic forgetting and over-fitting, a larger experience replay buffer and lower frequency of updating the target model are tried. We see a more stable performance but slower progress (even hard to converge within 2000 episodes) of reaching the goal that the average number of steps of the last 100 episodes is more than 200.