Use Of GenAI

This homework is completed with the help of Windsurf VS code extension.`https://windsurf.com/`

What is used:

- Autofill feature to generate syntactically correct latex code (each tab key pressed filled no more than 100 characters, at most 20% of the predicted text is adapted) for the homework with human supervision.
- Use AI to debug the latex code and find unclosed parentheses or other syntax errors.
- Use AI to autofill the parts that follows the same structure as the previous parts (example: case by case proofs).
- Use AI to auto correct misspelled words or latex commands.

What is not used:

- Directly use AI to generate the solutions in latex document.
- Use AI to ask for hint or solution for the problems.
- Select part of the document and ask AI to fill the parts missing.

1. **Answer questions in Section 3** Due to the state space complexity of some visual input environments, we may represent Q-functions using a class of parameterized function approximators $\mathcal{Q} = \{Q_w \mid w \in \mathbb{R}^p\}$, where p is the number of parameters. Remember that in the *tabular setting* given a 4-tuple of sampled experience (s, a, r, s'), the vanilla Q-learning update is

$$Q(s,a) \coloneqq Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right), \tag{1}$$

where $\alpha \in \mathbb{R}$ is the learning rate. In the *function approximation setting*, the update is similar:

$$w \coloneqq w + \alpha \left(r + \gamma \max_{a' \in A} Q_w(s', a') - Q_w(s,a) \right) \nabla_w Q_w(s,a). \tag{2}$$

Q-learning can seem as a pseudo stochastic gradient descent step on

$$\ell(w) = \mathbb{E}_{s,a,r,s'} \left(r + \gamma \max_{a' \in A} Q_w(s', a') - Q_w(s,a) \right)^2. \tag{3}$$

where the dependency of $\max_{a' \in A} Q_w(s', a')$ on w is ignored, i.e., it is treated as a fixed target.

1. [**10pt**] Show that the update 1 and update 2 are the same when the functions in $\mathcal{Q}$ are of the form $Q_w(s,a) = w^T \phi(s,a)$, with $w \in \mathbb{R}^{|S||A|}$ and $\phi : S \times A \to \mathbb{R}^{|S||A|}$, where the feature function ϕ is of the form $\phi(s,a)_{s',a'} = \mathbb{I}[s' = s, a' = a]$, where $\mathbb{I}$ denotes the indicator function which evaluates to 1 if the condition evaluates to true and vice versa. Note that the coordinates in the vector space $\mathbb{R}^{|S||A|}$ can be seen as being indexed by pairs (s', a'), where $s' \in S$, $a' \in A$.

   Proof. When the functions in $\mathcal{Q}$ are of the form $Q_w(s,a) = w^T \phi(s,a)$, with $w \in \mathbb{R}^{|S||A|}$ and $\phi : S \times A \to \mathbb{R}^{|S||A|}$, note that $\sum_{s \in S} \sum_{a \in A} \phi(s,a)^T \phi(s,a) = \sum_{s \in S} \sum_{a \in A} \mathbb{I}[s' = s, a' = a] = 1$.

$$\begin{aligned}
Q(s,a) &= Q(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right) \\
w^T \phi(s,a) &= w^T \phi(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right) \phi(s,a)^T \phi(s,a) \\
w^T \phi(s,a) &= w^T \phi(s,a) + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right) \nabla_w (w^T \phi(s,a))^T \phi(s,a) \\
w^T \phi(s,a) &= \left(w^T + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right) \nabla_w Q_w(s,a) \right)^T \phi(s,a) \\
w &= w + \alpha \left(r + \gamma \max_{a' \in A} Q(s', a') - Q(s,a) \right) \nabla_w Q_w(s,a)
\end{aligned}$$

   □

2. [**10pt**] What is the deadly triad in the reinforcement learning? What are the main challenges of using deep learning for function approximation with Q-learning? How does Deep Q-Learning method overcome these challenges?

   The deadly triad in the reinforcement learning are

   i. Bootstraping

   ii. Function approximation

   iii. Off-policy

   The Deep Q-Learning method overcome the instability caused by the deadly triad interact with statistical estimation issues induced by the bootstrap method used by boostrapping on a separate network and by reducing the overestimation bias. (Use double Q-learning to reduce the overestimation bias.)

3. [**10pt**] Explain how double Q-learning helps with the maximization bias in Q-learning.

   The double Q-learning decouple the action selection and evaluation of action to separate networks.

2. **The auto-generated results figure** along with a brief description about what has the figures shown.

    1. **DQN**

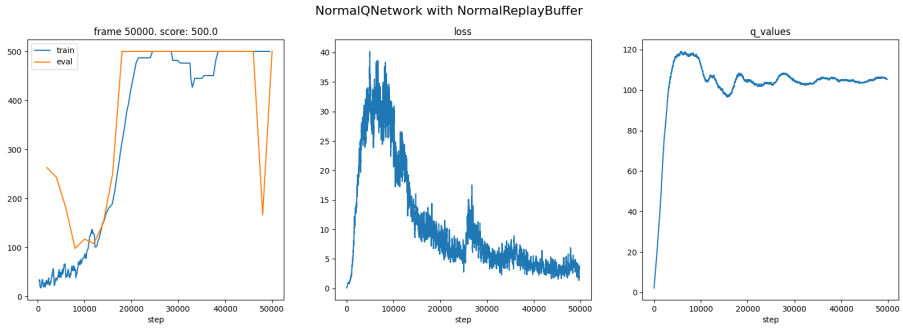

Figure 1: DQN. Nothing to say but what expected from training.

    2. **Double DQN**

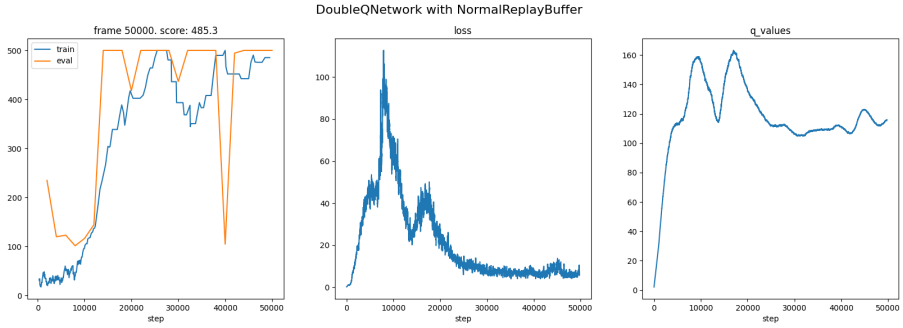

Figure 2: Double DQN. I found there is interesting camel like bump for q-value when training with Double DQN. It is less stable than the vanilla DQN.

    3. **Dueling DQN**

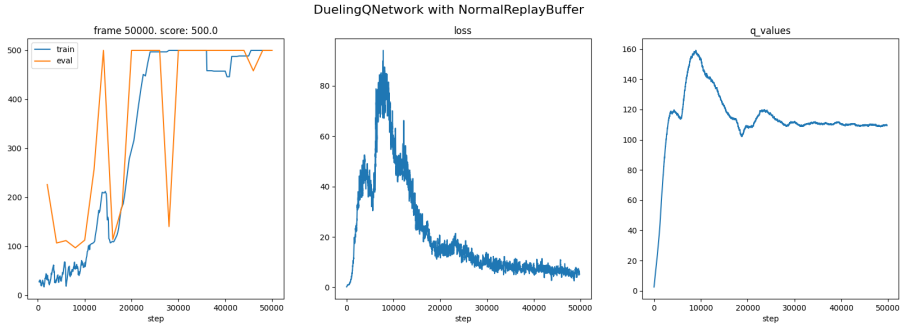

Figure 3: Dueling DQN. Using Advantage network creates comparable results as the DQN.

4. **Prioritized Experience Replay**

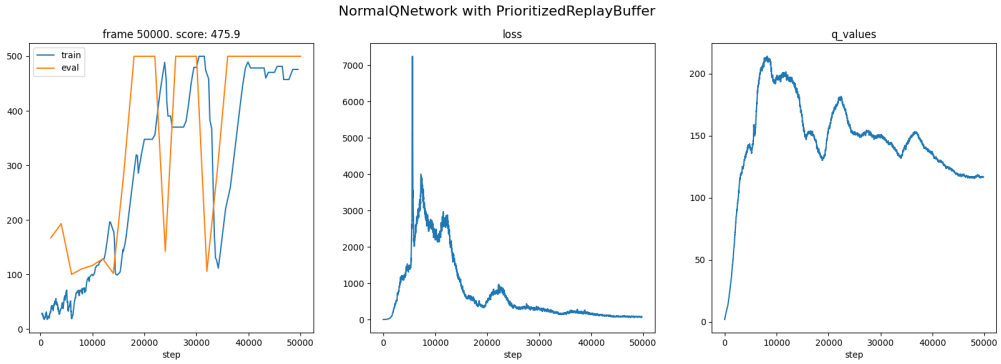

Figure 4: Prioritized Experience Replay. Using this alone makes the training process less stable and loss is significantly higher than the previous methods.

5. **N-Step Experience Replay**

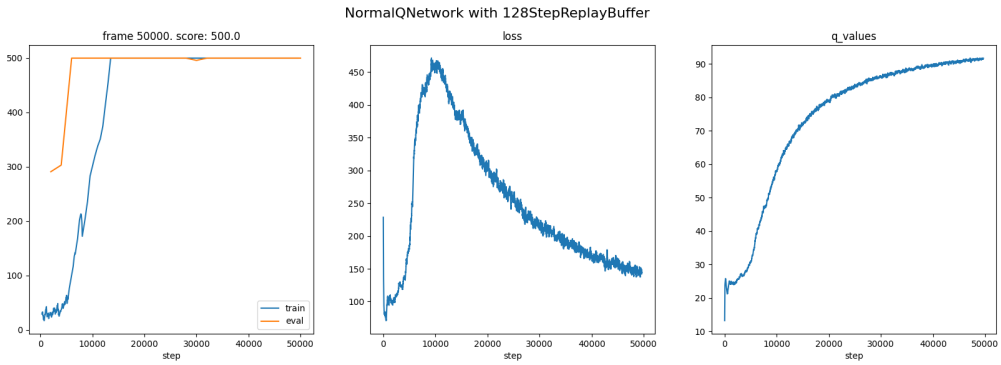

Figure 5: N-Step Experience Replay. So far the most stable method of training, especially when the replay buffer size is large. However, when the replay buffer size is too small, typically ≤ 70, the training process may not converge to optimal performance.

6. **N-Step + PER**

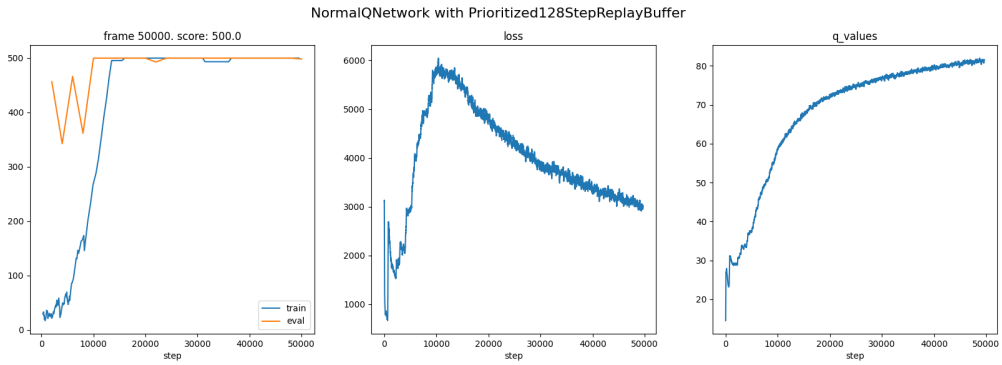

Figure 6: NStep + PER. Combining the two methods counter the unstable loss function for training in PER.

7. **Noisy DQN**

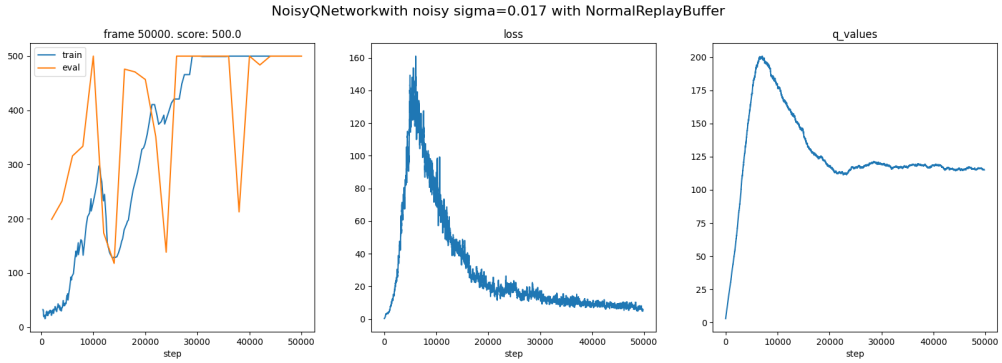

Figure 7: Noisy DQN. Experiment for sigma = 0.017 gets comparable result with normal DQN. Stability issue persist when sigma is too large.

3. **Any other findings**

   I implemented Extra credit Noisy DQN. Helpful commands to run in ./commands/4.8.sh Found that when sigma is too large, for example $\sigma = 0.5$. The model may not converge to optimal performance. Intuitively, the Noisy linear layer shall improve the robustness of the model. But the effect is not obvious as expected.