CPEN 502 Assignment Part 3 –

Reinforcement Learning with Backpropagation

Student Name: Chao-Wu Chu
Student Number: 85406312

(4) The use of a neural network to replace the look-up table and approximate the Q-function has some disadvantages and advantages.

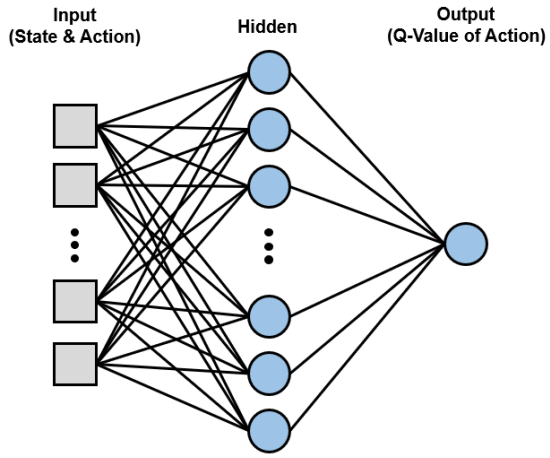
(a) There are 3 options for the architecture of your neural network. Describe and draw all three options and state which you selected and why. (3pts)

Answer:

The options for the three neural network architectures are as follows:

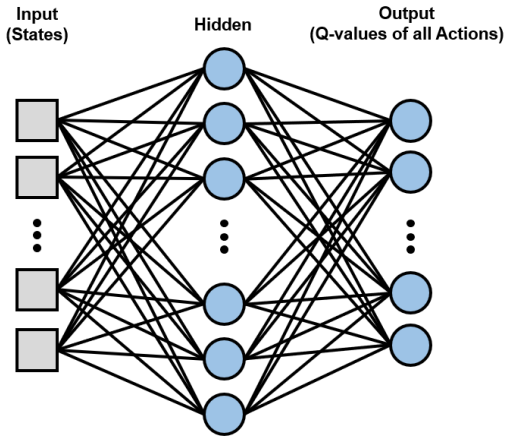
(1) Single State-Action Pair Architecture

This model processes a state-action pair as input and produces a single Q-value. It is direct and efficient for scenarios where evaluating the relationship between a particular action and its outcome is crucial. However, this model may become less efficient in situations where multiple actions for the same state need assessment, as it necessitates separate forward passes for each action.



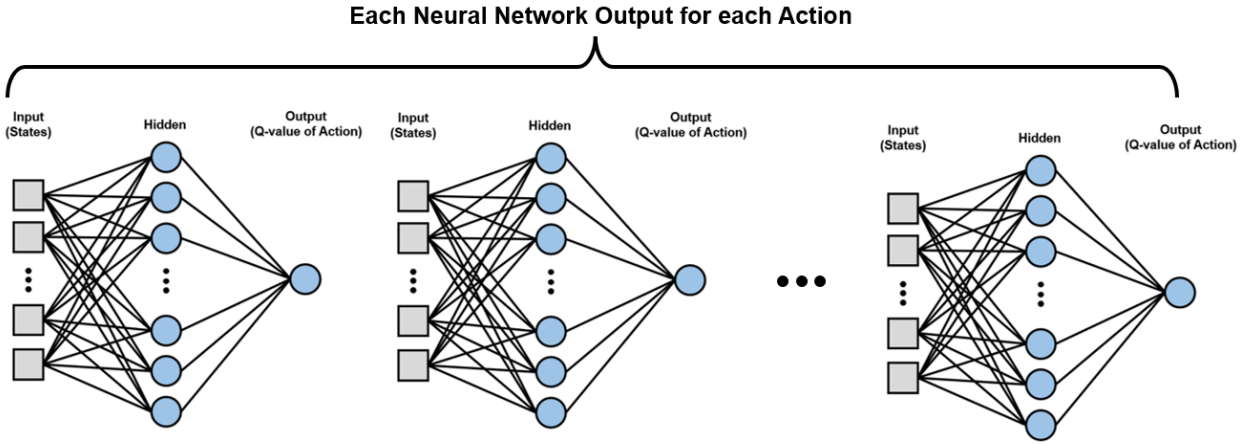
(2) State-Only Input Architecture

This design requires only the state as input and computes Q-values for all potential actions. It surpasses the first model in efficiency when multiple actions for the same state need evaluation, performing all assessments in a single forward pass. This feature is especially beneficial in dynamic environments where quick evaluation of multiple possible actions is required.



(3) Multiple Neural Networks Architecture

In this structure, each action is associated with its own neural network. This strategy can result in more specialized networks that are potentially more adept at assessing their respective actions. However, it also substantially increases the complexity and the demands on resources for multiple neural networks training.



For my project, I have chosen the third architecture. I believe that employing multiple neural networks to train for each specific action can minimize the interference in weight updates among different actions. This architecture enables my robot to select better actions for subsequent steps and learn more efficient strategies for victory.

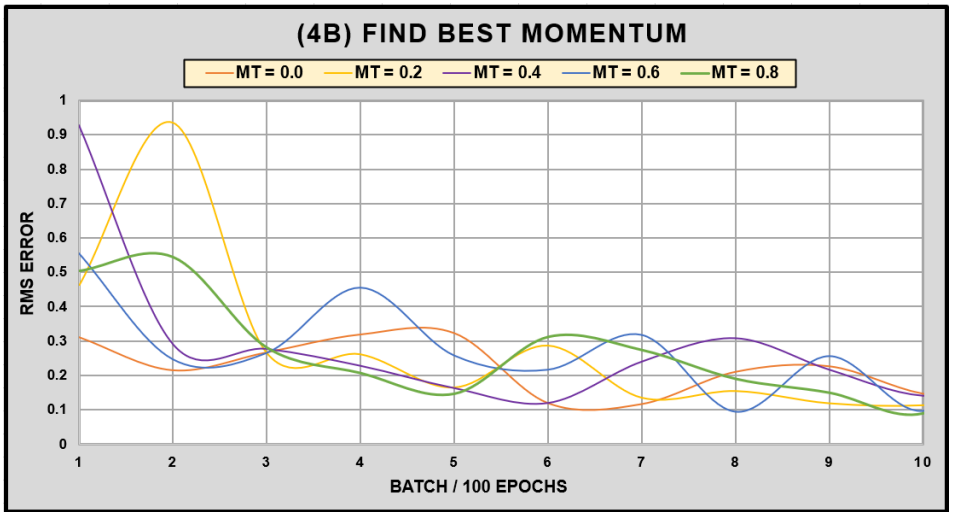
(b) Show (as a graph) the results of training your neural network using the contents of the LUT from Part 2. Your answer should describe how you found the hyper-parameters which worked best for you (i.e. momentum, learning rate, number of hidden neurons). Provide graphs to backup your selection process. Compute the RMS error for your best results. (5 pts)

Answer:

I utilized the Lookup Table (LUT) trained in Part 2, using the LUT states as inputs and the Q-values of actions as outputs. These were then fed into the neural network performing weight updates for its specific action.

After training the neural networks for 1000 epochs, I obtained the Root Mean Square (RMS) error for various experimental comparisons. From the RMS error data, I was able to identify the optimal hyper-parameters for the training process, including momentum, learning rate, and the number of hidden neurons. Subsequently, I determined the best hyper-parameters corresponding to the lowest RMS error.

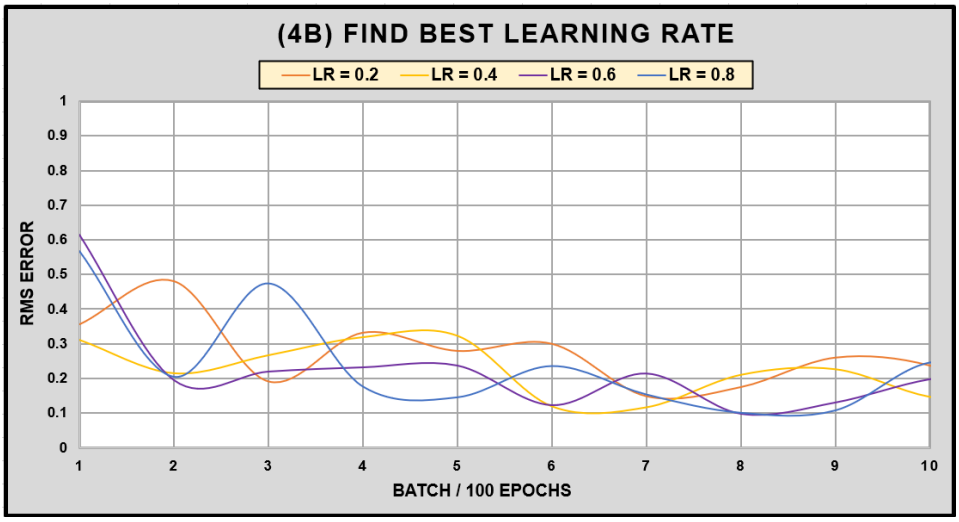
(1) Momentum:



Momentum	0.0	0.2	0.4	0.6	0.8
RMS Error	0.146	0.112	0.142	0.096	0.089

The analysis of RMS error data indicated that higher momentum values effectively accelerate the training speed of the neural networks. Generally, the RMS error tends to decrease as the Momentum value increases. Therefore, I adopted a Momentum value of 0.8.

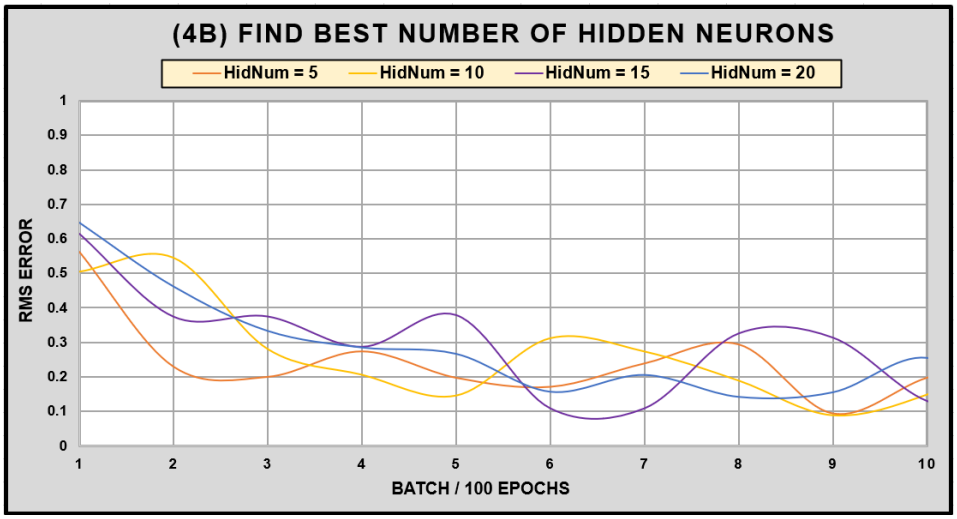
(2) Learning Rate:



Learning Rate	0.2	0.4	0.6	0.8
RMS Error	0.237	0.146	0.197	0.246

The data analysis on RMS error revealed that the learning rate has a significant impact on the learning speed. A too-low learning rate results in slow learning, whereas a too-high learning rate causes the learning to be overly rapid, leading to oscillations and difficulty in reaching the optimal minimum value, thereby reducing overall learning efficiency. I found that a learning rate of 0.4 yielded the lowest RMS error, optimizing the training process.

(3) Number of Hidden Neurons:



Number of Hidden Neurons	5	10	15	20
RMS Error	0.198	0.149	0.130	0.256

Having too few hidden neurons might lead to inefficiencies due to insufficient model complexity, while too many hidden neurons can cause the neural network to become overly complex, leading to an overfitting effect and adversely affecting training. The RMS error data analysis showed that when the Number of Hidden Neurons is set to 15, it results in the smallest RMS error. This finding aligns with theoretical expectations, suggesting an optimal number of neurons in the hidden layer.

(c) Comment on why theoretically a neural network (or any other approach to Q-function approximation) would not necessarily need the same level of state space reduction as a look up table. (2 pts)

Answer:

Neural networks are adept at generalizing from known to unknown states, unlike look-up tables which require an explicit entry for each state-action pair. This generalization capability allows them to operate effectively in large or continuous state spaces, where look-up tables would be impractical due to their exponential growth in size.

Additionally, neural networks can efficiently process high-dimensional input spaces by

focusing on the most relevant features, thereby internally reducing the problem's dimensionality. This is in contrast to look-up tables, which need manual or algorithmic feature selection to manage complexity.

Furthermore, the scalability of neural networks makes them more suitable for complex environments. While the size of a look-up table grows exponentially with the state space, making it impractical for large spaces, neural networks handle such growth more efficiently, as their complexity doesn't increase proportionally with the state space size.

Therefore, neural networks require less state space reduction for Q-function approximation compared to look-up tables, underlining why they don't need the same level of state space reduction as a look-up table.

(5) Hopefully you were able to train your robot to find at least one movement pattern that results in defeat of your chosen enemy tank, most of the time.

(a) Identify two metrics and use them to measure the performance of your robot with online training. I.e. during battle. Describe how the results were obtained, particularly with regard to exploration? Your answer should provide graphs to support your results. (5 pts)

Answer:

To assess my robot's performance during online training in battle scenarios, I selected "Win Rate" and "RMS Error" as the key metrics.

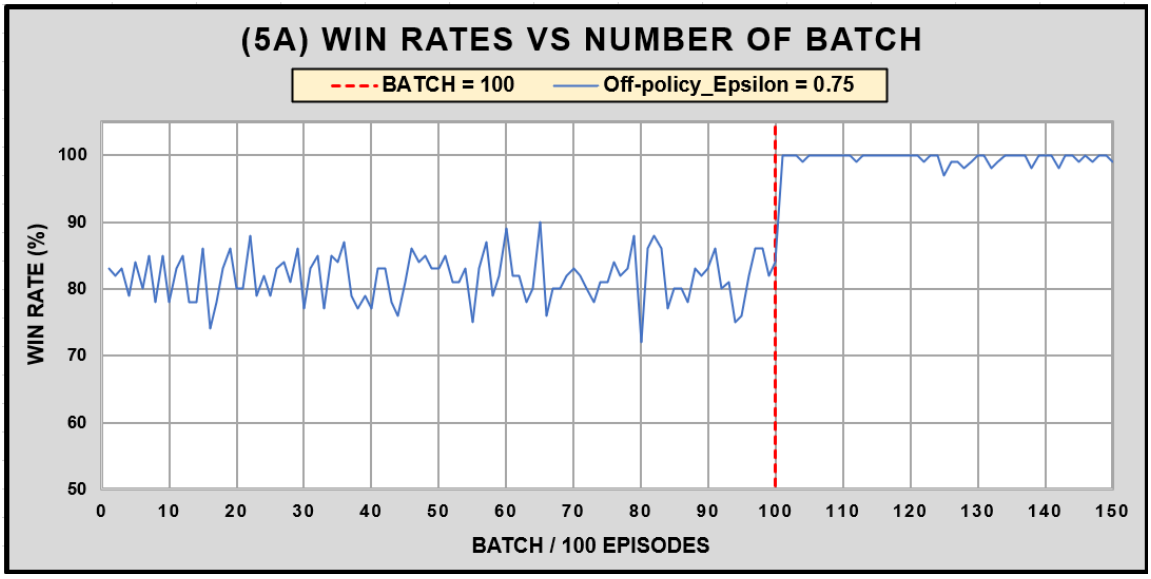
I trained my robot using the optimal hyper-parameters identified for the neural network, setting the exploration rate (epsilon) at 0.75 for 100 batches (10,000 episodes). In the subsequent testing phase, I set the exploration rate (epsilon) to 0 and conducted tests over 50 batches (5,000 episodes) to validate whether my robot had successfully learned to defeat its opponents.

(1) Win Rate

The win rate, calculated as the ratio of battles won to a fixed number of battles (episodes), directly reflects the robot's effectiveness in achieving its primary goal:

winning battles.

From the data, it's evident that my robot's win rate improved from 82% in the training phase to about 100% in the testing phase, confirming that win rate is an effective metric for observing the robot's learning performance.

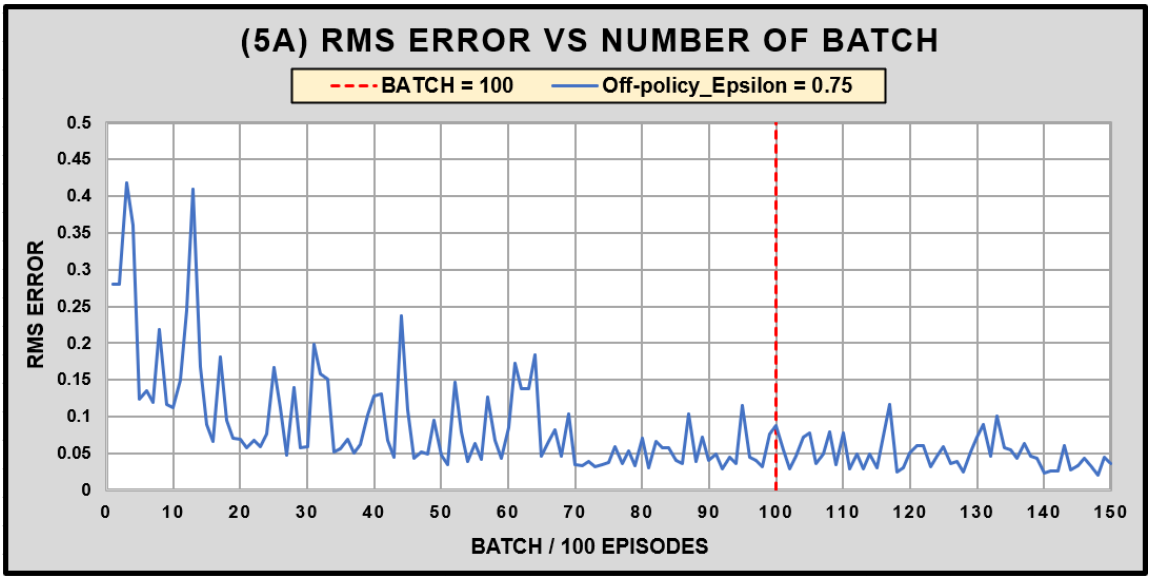


(2) RMS Error:

RMS error, in this context, refers to the error in the robot's prediction of Q-values or its estimation of the environmental state, serving as a measure of the accuracy of the robot's learning algorithm.

The data showed a continuous decline in RMS error during the training phase, dropping from around 0.3 to a level close to 0.05. During the testing phase, the RMS error remained at this level. This indicates that after continuous learning, the robot's actions increasingly aligned with the expected pattern of the Q-function, stabilizing at the minimum RMS error level once a consistent winning strategy was found.

This demonstrates that RMS error is a reliable indicator of the robot's learning status and its ability to learn behavior patterns effectively against opponents.



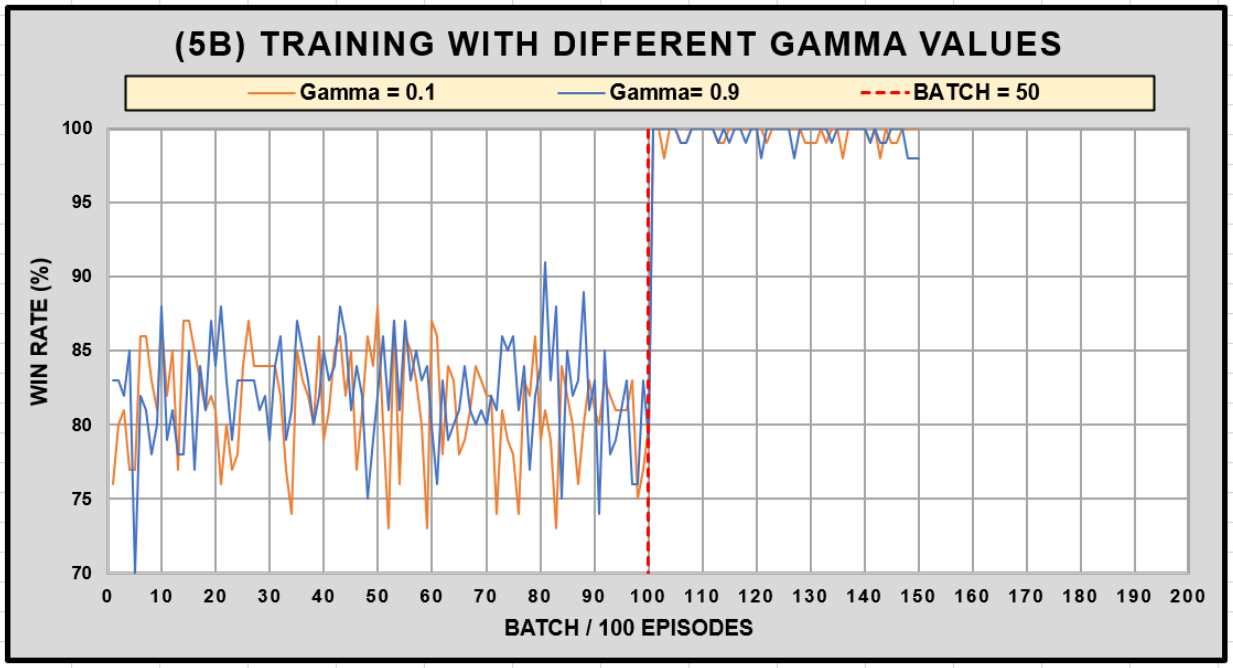
(b) The discount factor γ can be used to modify influence of future reward. Measure the performance of your robot for different values of γ and plot your results. Would you expect higher or lower values to be better and why? (3 pts)

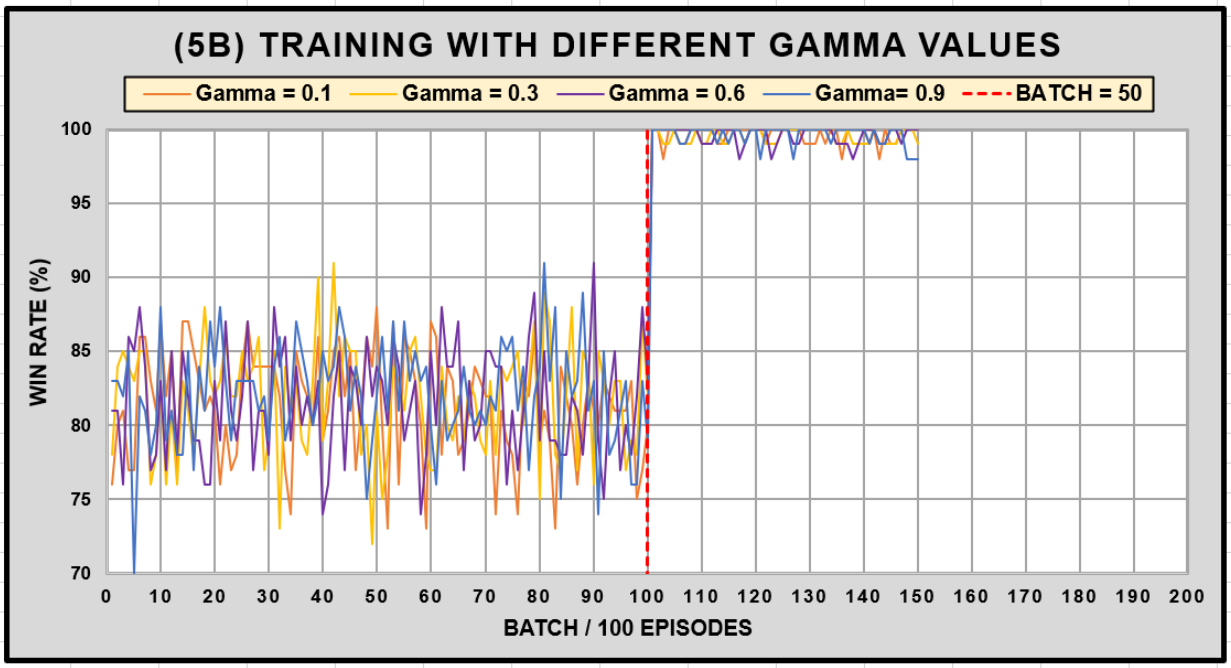
Answer:

The discount factor (γ) plays a crucial role in how future rewards influence an agent's learning process. A lower γ, closer to 0, leads the agent to prioritize immediate rewards, focusing on short-term gains. In contrast, a higher γ, closer to 1, makes the agent more forward-looking, placing greater value on future rewards.

For tasks where future rewards are significant, a higher γ typically leads to better results. However, in tasks dominated by immediate rewards, a lower γ might be more suitable.

From the comparative charts, it's observed that although a higher discount factor (γ = 0.9) shows slightly better performance compared to a lower one (γ = 0.1), the overall difference in outcomes is not pronounced.





The minimal performance variation across different γ values could be attributed to several factors. First, the nature of the task plays a significant role. In Robocode, where rewards may be more immediate and less dependent on long-term strategies, the impact of γ might be limited. Also, some reinforcement learning algorithms may be less sensitive

to changes in the discount factor, especially in environments where rewards are not significantly delayed.

Other elements such as the learning rate, exploration strategy, or the agent's architecture could also overshadow the impact of γ. I believe that the discount factor is likely influenced by factors like the learning rate or the nature of immediate rewards. Considering the Bellman equation, a high learning rate or the predominance of immediate rewards could diminish the effect of the discount factor, leading to similar outcomes when fine-tuning γ.

(c) Theory question: With a look-up table, the TD learning algorithm is proven to converge – i.e. will arrive at a stable set of Q-values for all visited states. This is not so when the Q-function is approximated. Explain this convergence in terms of the Bellman equation and also why when using approximation, convergence is no longer guaranteed. (3 pts)

Answer:

(1) TD Learning with a Look-Up Table

In TD learning using a look-up table, each state-action pair (S, A) has an associated Q-value, Q(S, A). The updates to these Q-values are guided by the Bellman equation:

$$\boldsymbol{Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma\, max_a Q(S', a) - Q(S,A)]}$$

This equation iteratively adjusts the Q-values towards the expected return, capturing the immediate reward plus the discounted future rewards. The updates continue until the Q-values stabilize, ensuring convergence. This stability is feasible due to the direct and isolated updates for each state-action pair in the look-up table.

(2) Challenges with Function Approximation:

When approximating the Q-function, typically with a parameterized model like a neural network, the update rule modifies the parameters of the model instead of individual Q-values. The approximate Bellman update then becomes (θ represents the parameters of the approximation model):

$$\boldsymbol{\theta \leftarrow \theta + \alpha[R + \gamma\, max_a Q(S', a;\, \theta) - Q(S, A; \theta)]\nabla_\theta Q(S', a;\, \theta)}$$

The challenge arises because the function approximator may not capture the complex relationship between state-action pairs and their Q-values. This can lead to inaccuracies in representing the Q-function, and the recursive application of the Bellman equation may amplify these inaccuracies.

(3) Convergence Issues with Approximation

The function approximator's inability to perfectly model the Q-function leads to a phenomenon known as "interference," where updating the value for one state-action pair affects the values of others. This is unlike the isolated updates in a look-up table. As a result, the convergence guaranteed by the Bellman equation in the look-up table scenario is no longer assured with function approximation. The approximator might oscillate or diverge instead of converging to a stable set of Q-values, primarily due to the compounded inaccuracies in the recursive updates.

In summary, while the Bellman equation ensures convergence of Q-values in a look-up table through direct and isolated updates, the use of function approximation introduces potential inaccuracies and interference among state-action pairs. This complexity can prevent the convergence of the TD learning algorithm when a function approximator is employed.

(d) When using a neural network for supervised learning, performance of training is typically measured by computing a total error over the training set. When using the NN for online learning of the Q-function in robocode this is not possible since there is no a-priori training set to work with. Suggest how you might monitor learning performance of the neural net now. (3 pts)

Answer:

Monitoring the learning performance of a neural network in an online learning environment, where there is no predefined training set, presents unique challenges. However, there are effective strategies listed below to assess how well the neural network is learning the Q-function.

(1) Average Reward

The average reward is a key indicator of how well the neural network is learning. By averaging the rewards received over a set number of recent episodes, we can track the performance trend of the agent.

This method smooths out the variability inherent in individual episodes, providing a more stable measure of performance. An increasing trend in the average reward indicates that the agent is learning to make better decisions over time.

(2) Win Rate

Win rate is a straightforward metric in competitive environments. It is calculated as the ratio of battles won to the total number of battles fought over a certain period.

Monitoring the win rate helps in understanding the effectiveness of the learned strategies. A consistent or increasing win rate suggests that the neural network is effectively applying its learned knowledge to win battles. It's also a direct measure of the agent's performance in its intended environment.

(3) Changes in Q-Values

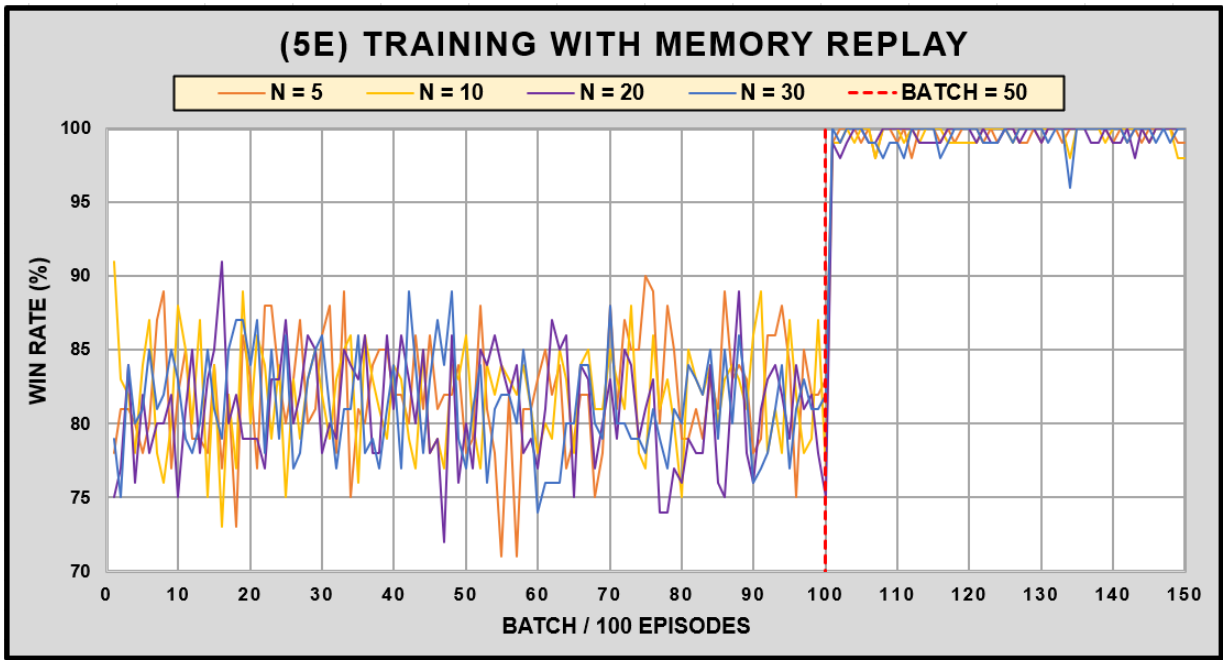
Observing the changes in Q-values provides insight into the learning and convergence of the neural network. In the early stages of learning, large updates to Q-values are common as the network acquires new information.

As learning progresses, the magnitude of these updates should decrease, indicating that the network is converging towards optimal Q-values. Persistent large changes may suggest ongoing learning or instability in the learning process.

Through these three methods, we can comprehensively monitor the learning performance of a neural network in an online setting. The average reward and win rate provide direct measures of the robot’s effectiveness, while tracking changes in Q-values offers insights into the underlying learning process and convergence of the network.

(e) At each time step, the neural net in your robot performs a back propagation using a single training vector provided by the RL agent. Modify your code so that it keeps an array of the last say n training vectors and at each time step performs n back propagations. Using graphs compare the performance of your robot for different values of n. (4 pts)

Answer:



After experimenting with changing the size of the replay memory, I observed no significant variation in the win rate. This could be due to several reasons:

1. Complexity of the Environment and Task

In a relatively simple environment or task, a smaller replay memory might be adequate for the neural network to learn effectively. In such scenarios, enlarging the replay memory may not lead to noticeable performance enhancements.

2. Quality of Experience in the Replay Memory

The effectiveness of a larger replay memory heavily relies on the diversity and quality of the experiences it contains. If the stored experiences are too homogenous or not representative of the varied situations the robot faces, merely increasing their quantity

won't necessarily boost learning.

3. Efficiency of the Learning Algorithm

The architecture of the neural network and the proficiency of the learning algorithm are significant. If the algorithm is already maximizing learning from a smaller set of experiences, adding more training data may have a limited impact on its performance.

I believe that the simplicity of the environment might be why the robot's additional learning from replay memory isn't clearly reflected in the win rate. By increasing the opponent's strength, where the robot can't easily find a consistent winning strategy, the need for replay memory might become more apparent. In such a situation, the effects of replay memory on learning could be more observable.

(6) Overall Conclusions

(a) This question is open-ended and offers you an opportunity to reflect on what you have learned overall through this project. For example, what insights are you able to offer with regard to the practical issues surrounding the application of RL & BP to your problem? E.g. What could you do to improve the performance of your robot? How would you suggest convergence problems be addressed? What advice would you give when applying RL with neural network based function approximation to other practical applications? (4 pts)

Answer:

From this course, I have gained substantial knowledge and understanding of reinforcement learning and neural networks, encompassing Backpropagation, Q-learning, Lookup Tables (LUTs), and more. Building a neural network from scratch and integrating it with reinforcement learning algorithms has deepened my understanding of the intricate details involved. This experience has equipped me with the necessary knowledge and practical experience to further delve into this field.

What could you do to improve the performance of your robot?

The efficiency and success of a robot's learning in reinforcement learning (RL) are

significantly influenced by the selection and definition of the state and action spaces, as well as how the overall rewards are structured. These aspects fundamentally determine the robot's basic behavior patterns and its potential for learning. Analyzing the correlation between these variables and each environment is one of the most fascinating aspects of RL.

To enhance overall performance, enhancing the Neural Network Architecture can be a strategic move. Incorporating more diverse types of layers, such as convolutional layers for spatial data, could be beneficial. Constructing a deeper neural network will increase its complexity and potentially its ability to process and learn from more intricate patterns and features within the environment. This approach not only enhances the network's ability to discern finer details and nuances in the data but also allows it to make more informed decisions based on a broader context.

How would you suggest convergence problems be addressed?

If faced with convergence problems in neural network training, there are several strategies to consider:

(1) Adjust the Learning Rate

A learning rate that is too high may cause the network to overshoot the optimal solution, while a too-low rate might lead to very slow convergence or getting stuck in a local minimum. To address this, experimenting with different learning rates or adopting adaptive learning rate methods like Adam could be beneficial.

(2) Use Experience Replay

Implementing experience replay can stabilize learning by breaking the correlation between sequential learning updates. Prioritized experience replay, which samples more critical experiences more frequently, can be especially effective.

(3) Gradient Clipping

In scenarios where the neural network undergoes large updates (gradient explosions), gradient clipping can help. It limits the size of gradient steps, leading to more stable training.

(4) Reduce Overfitting

Overfitting often results in convergence issues. To mitigate or prevent overfitting, incorporating dropout or L1/L2 regularization techniques can be effective. Additionally, employing early stopping is another viable approach.

These methods help in managing the stability and efficiency of the learning process, thereby improving the chances of achieving convergence in neural network training.

What advice would you give when applying RL with neural network based function approximation to other practical applications?

I believe that understanding the application domain is crucial for successful implementation in reinforcement learning. A thorough grasp of the domain enables the selection of the most suitable algorithms and neural network architectures. Following this, careful design of the state-action space, reward function, and the balance between exploration and exploitation is essential.

Reinforcement learning involves a multitude of influencing factors, and one of the most challenging aspects is the interplay between these factors, making analysis and experimentation complex. Initially focusing on the primary factors can lead to more efficient development of the project or application.

(b) Theory question: Imagine a closed-loop control system for automatically delivering anesthetic to a patient under going surgery. You intend to train the controller using the approach used in this project. Discuss any concerns with this and identify one potential variation that could alleviate those concerns. (3 pts)

Answer:

The most critical concern in using RL for medical applications like anesthesia delivery is patient safety. Unlike games or simulations, errors in medical applications can have severe or even life-threatening consequences. Ethical considerations also arise, as experimenting with different dosages or strategies directly on patients to improve the learning algorithm is not feasible or ethical.

Another concern would be the unpredictability of human physiology. Human physiological responses to anesthesia are highly individualized and can be unpredictable. This complexity makes it challenging to model and learn an appropriate control policy that works universally. The RL model might not adequately capture all the nuances of human physiology, leading to suboptimal or dangerous decisions.

A potential variation to alleviate these concerns could be the use of a Supervised Learning (SL) approach in conjunction with RL, trained initially on historical, anonymized patient data where the outcomes of various anesthesia dosages are known.

By starting with supervised learning on historical data, the model can learn safe and proven practices. The subsequent refinement through RL in a simulated environment allows for exploration and optimization without direct patient involvement. This approach balances the need for a robust, well-trained model with the imperative of patient safety and ethical considerations.