Assignment 3- Reinforcement Learning

Galena-Wagdy-Zareef-20399124

1) Q-Learning agent with Neural Networks

The QLearningNN agent is a reinforcement learning agent that uses a neural network to estimate the Q-values of state-action pairs. The agent uses the Q-learning algorithm, which is a model-free, off-policy, value-based learning algorithm. The agent learns from experience by updating its Q-values based on the observed rewards and the estimated Q-values of the next state.
The QLearningNN agent is implemented as a class that inherits from the Agent class. The agent has several attributes, including the neural network model, the exploration rate (epsilon), the discount factor (discount), and the latest action taken.
The agent interacts with the environment by selecting actions and observing the resulting rewards and next states. The agent updates its Q-values based on the observed rewards and the estimated Q-values of the next state. The agent uses a neural network to estimate the Q-values of state-action pairs.
The neural network model used by the agent is a simple multi-layer perceptron (MLP) with three hidden layers. The model takes as input the state-action pair and outputs the estimated Q-value. The model is trained using the mean squared error loss and the Adam optimizer.

From the output, we can see that the Q-learning agent with neural network approximation is trained on the Catch environment for 1000 episodes. During training, the agent's performance is measured in terms of the return obtained in each episode and the mean return over the last 10 episodes.
The mean return initially fluctuates between -1.0 and 1.0, indicating that the agent is initially exploring and trying out different actions. However, as training progresses, the mean return stabilizes around -0.6, indicating that the agent has learned a policy that performs moderately well on the environment.
The agent is then evaluated on the environment for 10 episodes, and the average return obtained is -0.4 with a standard deviation of 0.917. This suggests that the agent's learned policy is not very effective on the evaluation set and may need further tuning of hyperparameters or changes to the neural network architecture to improve its performance.
Overall, the code provides a good starting point for training and evaluating a Q-learning agent with neural network approximation on the Catch environment, but further work may be needed to optimize the agent's performance.

(mean: -0.4, std: 0.9165)

Experiment with model architectures:

1- Try different activation functions.

first	Second relu	Second sigmoid
Fist relu	The performance of the Q-learning agent varies depending on the combination of activation functions used for the neural network model. For example, in the first run with act1=relu and act2=relu, the agent achieves a mean return of around -0.48 during training and an average return of 0.4 with a high standard deviation during evaluation. In contrast, in the run with act1=sigmoid and act2=relu, the agent achieves a mean return of around -0.68 during training and an average return of 0.6 with a low standard deviation during evaluation. The agent's performance during training generally improves as the number of episodes increases, indicating that the agent is gradually learning a better policy. However, the performance during evaluation is not always consistent with the training performance,	During training, the agent achieves a mean return of around -0.48, which is similar to the performance of the agent with act1=relu and act2=relu. During evaluation, the agent achieves a mean return of -0.2, which is lower than the performance of the agent with act1=relu and act2=relu but higher than the performance of the agent with act1=sigmoid and act2=relu. The standard deviation of the returns obtained during evaluation is high, indicating that the agent's performance is highly variable and not reliable.

	and the agent may not generalize well to new situations. The standard deviation of the returns obtained during evaluation is generally high, indicating that the agent's performance is highly variable and not reliable. This suggests that further tuning of hyperparameters or changes to the neural network architecture may be needed to improve the agent's performance. **(mean: 0.4, std: 0.9165)**	Overall, the performance of the agent with act1=relu and act2=sigmoid is mediocre and may not be suitable for this environment. More experiments with different activation functions and neural network architectures may be needed to find a configuration that performs well on this task. **(mean: -0.2, std: 0.979)**
First sigmoid	During training, the agent achieves a mean return that fluctuates between -0.52 and -0.72, which is similar to the performance of the agent with act1=relu and act2=sigmoid. During evaluation, the agent achieves a mean return of -0.2, which is the same as the performance of the agent with act1=relu and act2=sigmoid. The standard deviation of the returns obtained during evaluation is high, indicating that the agent's performance is highly variable and not reliable. Overall, the performance of the agent with act1=sigmoid and act2=relu is mediocre and may not be suitable for this environment. More experiments with different activation functions and neural network architectures may be needed to find a configuration that performs well on this task. **(mean: -0.2, std: 0.979)**	During training, the agent achieves a mean return that fluctuates between -0.6 and -0.806, which is worse than the performance of the previous two activation function combinations. During evaluation, the agent achieves a mean return of -0.8, which is the lowest of all the activation function combinations tested. The standard deviation of the returns obtained during evaluation is relatively low, indicating that the agent's performance is less variable than in the previous experiments. Overall, the performance of the agent with act1=sigmoid and act2=sigmoid is not good and may not be suitable for this environment. More experiments with different activation functions and neural network architectures may be needed to find a configuration that performs well on this task. **(mean: -0.8, std: 0.6)**

2- Change the layer sizes:

	second 30	Second 20
First 60	The agent's average returns start low and fluctuate throughout the training process, but overall there is no clear trend of improvement. The agent is evaluated on 10 episodes and receives a mean return of -0.2 with a standard deviation of 0.9798.	The agent's performance is similar to the first trial, with fluctuating returns and no clear trend of improvement. The agent is evaluated on 10 episodes and receives a mean return of -0.2 with a standard deviation of 0.9798, which is the same as the first trial.
First 50	The agent's performance starts low but shows a slight improvement after the 20th episode. The agent's average returns continue to fluctuate but overall show a slight trend of improvement. The agent is evaluated on 10 episodes and receives a mean return of -0.2 with a standard deviation of 0.9798, which is the same as the first two trials.	The agent's performance is consistently low throughout the training process with no clear trend of improvement. The agent is evaluated on 10 episodes and receives a mean return of -0.6 with a standard deviation of 0.8, which is the lowest mean return and the lowest standard deviation among all trials.

The impact of the choice of hidden layer sizes on the agent's performance depends on the complexity of the task and the amount and quality of training data available. In this particular task, the agent's performance may not be sensitive to the choice of hidden layer sizes because the task is relatively simple, and the agent can learn to perform it effectively with a variety of architectures. However, in more complex tasks or environments, the choice of hidden layer sizes can have a significant impact on the agent's ability to learn and perform optimally. For example, in tasks

where the input data is high-dimensional or noisy, larger hidden layers may be required to effectively capture important features and patterns in the data. On the other hand, in tasks where the input data is low-dimensional or less complex, smaller hidden layers may be sufficient and larger hidden layers may lead to overfitting. In addition to the choice of hidden layer sizes, other factors such as the activation functions, the learning rate, and the optimization algorithm used for training the neural network can also have a significant impact on the agent's performance. Therefore, it is important to carefully tune these hyperparameters and experiment with different architectures to find the best configuration for a particular task.

3- Change the number of layers:

Trial 1: From the output, it seems that an agent is being trained using reinforcement learning. The agent's performance is being evaluated using two metrics: the return and the mean return. The return is a measure of the agent's performance in a single episode, while the mean return is the average return over a certain number of episodes.
During the training process, the agent's performance seems to be quite erratic, with its mean return fluctuating between negative and positive values. The agent seems to be struggling to learn the task, as indicated by its inconsistent performance.
After the training process, the agent is evaluated on a separate set of episodes, and the mean return and standard deviation are reported. The mean return of the agent during evaluation is -0.8, which indicates that the agent is still performing poorly even after training.
Overall, it seems that the agent's training process has not been successful in achieving good performance, and further improvements may be needed to improve its performance.
(mean: -0.8, std: 0.6)

Trial 2: From the training output, we can see that the agent is not performing very well, as the mean return is mostly negative, and the agent is not able to consistently achieve positive rewards. The evaluation output also confirms that the agent is not performing well, as the mean return is 0.0 and the standard deviation is 1.0, indicating that the agent is not able to consistently achieve positive rewards in the evaluation environment.
It's possible that the agent needs more training or that the hyperparameters need to be tuned in order to improve its performance. It may also be helpful to analyze the agent's behavior and try to identify any patterns or areas where it is struggling, in order to diagnose the problem and find ways to improve the agent's performance.
(mean: 0.0, std: 1.0)

2) Q-Learning agent with NNs and a Replay Buffer:

This is the implementation of a Q-learning agent using a neural network and a replay buffer. The agent stores the experience tuples of (state, action, next_state) in a replay buffer and samples a batch of these tuples to update the Q-values using the Bellman equation. The Q-values are estimated using a neural network, which takes in the state-action pairs as inputs and outputs the Q-values for each possible action.
The agent uses an epsilon-greedy policy to select actions, where it selects random actions with probability epsilon and selects the action with the highest Q-value estimate for the given observation with probability 1 - epsilon. The epsilon value can be set using the set_epsilon() method.
The _make_input() method concatenates the flattened observation and the one-hot encoded action to create the input tensor for the neural network. The _q_func() method takes in the latest observation and the action to be taken, creates the input tensor using the _make_input() method, and returns the Q-value estimate for the given action.
The update() method samples a batch of experience tuples from the replay buffer and updates the Q-values using the Bellman equation and the neural network. It calculates the TD target and the TD error for each tuple, and updates the Q-value for the observation-action pair using the TD error and the neural network model. The gradients of the loss function with respect to the model's trainable variables are computed using TensorFlow's GradientTape, and the optimizer is used to apply the updates to the model's weights.

The output of the code shows that the agent was able to learn to play the Catch game environment using Q-learning with a neural network function approximator. The mean return, which is a measure of the agent's performance, improved gradually over the course of training, indicating that the agent was learning and making progress. However, the mean return did not consistently improve with every episode, which is normal for reinforcement learning algorithms. The agent needs to explore different actions to find an optimal policy, which can lead to fluctuations in performance before converging to a stable policy.

The use of a neural network as the function approximator allows the agent to learn a complex mapping from the state-action space to the Q-values. This can be advantageous for environments with high-dimensional state spaces, where traditional tabular methods become infeasible.
Overall, the output of the code suggests that the Q-learning algorithm with a neural network function approximator is a promising approach for learning to play simple games such as Catch. Further research could explore the performance of this approach on more complex environments and compare it to other reinforcement learning algorithms.

Experiment with Replay Buffer settings:

1- Modify the sampling method (e.g. give higher priority to recent items instead of sampling uniformly)
Based on the training output, the agent is being trained for a game and its performance is being evaluated after every 10 episodes. The training output shows the episode number, the return obtained by the agent in that episode, and the mean return over the last 10 episodes. The return is the cumulative reward obtained by the agent during the episode. The agent is not performing well during training, as it is consistently getting a return of -1.0, which means it is losing the game in every episode. The mean return over the last 10 episodes is also decreasing over time, indicating that the agent is not learning to improve its performance.
After training, the agent's performance is evaluated on the game environment using the evaluate() function. The output shows the return obtained by the agent in each of the 10 evaluation episodes, followed by the mean and standard deviation of the returns. In this case, the agent is again not performing well during evaluation, as it is getting a negative mean return of -0.6. The standard deviation of 0.8 indicates that the agent's performance is quite variable across episodes.
It is possible that the hyperparameters used for training the agent are not well-tuned, or that the neural network model used for function approximation is not able to capture the relevant features of the game state. Further experimentation and tuning may be necessary to improve the agent's performance.
(mean: -0.6, std: 0.8)

2- Change the eviction strategy
The output shows the training and evaluation results of the QLearningNNReplayEvictionStrategy agent in the Catch game environment. During training, the agent plays 100 episodes and updates its Q-value estimates based on the transitions stored in the replay buffer. At each timestep, the agent selects an action based on an epsilon-greedy exploration strategy, where it chooses the best action based on its Q-value estimates with probability 1-epsilon and a random action with probability epsilon.
The training output shows the return of each episode, which is the sum of rewards obtained during the episode. The mean return is also shown, which is the average return over the last 10 episodes. The return is negative when the agent fails to catch any fruit and positive when it catches at least one fruit. As we can see, the agent performs poorly during training, with a mean return of around -0.6 to -0.7, indicating that it fails to catch many fruits.
After training, the agent is evaluated on 10 episodes using the greedy policy (i.e., epsilon=0), where it always chooses the action with the highest Q-value estimate. The output shows the return of each episode and the mean and standard deviation of the returns. The evaluation results are also poor, with a mean return of -0.8, indicating that the agent still fails to catch many fruits even with the greedy policy.
Overall, these results suggest that the QLearningNNReplayEvictionStrategy agent is not very effective in learning a good policy for the Catch game environment and may require further modifications to improve its performance. Possible modifications include using different neural network architectures, adjusting the hyperparameters of the agent (e.g., learning rate, discount factor, epsilon), or implementing more sophisticated exploration and exploitation strategies. Additionally, it may be helpful to visualize the agent's behavior during training and evaluation to gain more insights into its strengths and weaknesses.
(mean: -0.8, std: 0.6)

3- Change the size of the replay buffer (i.e. the maximum number of entries)
The output shows the training and evaluation results of the QLearningNNReplaySize agent in the Catch game environment. During training, the agent plays 100 episodes and updates its Q-value estimates based on the transitions stored in the replay buffer. At each timestep, the agent selects an action based on an epsilon-greedy exploration strategy, where it chooses the best action based on its Q-value estimates with probability 1-epsilon and a random action with probability epsilon.
The training output shows the return of each episode, which is the sum of rewards obtained during the episode. The mean return is also shown, which is the average return over the last 10 episodes. As we can see, the agent performs

similarly as the QLearningNNReplayEvictionStrategy agent during training, with a mean return of around -0.6 to -0.7, indicating that it fails to catch many fruits.
After training, the agent is evaluated on 10 episodes using the greedy policy (i.e., epsilon=0), where it always chooses the action with the highest Q-value estimate. The output shows the return of each episode and the mean and standard deviation of the returns. The evaluation results are also poor, with a mean return of -0.2, indicating that the agent still fails to catch many fruits even with the greedy policy.
Overall, the performance of the QLearningNNReplaySize agent is similar to the QLearningNNReplayEvictionStrategy agent, and both agents perform poorly in the Catch game environment. This suggests that further modifications may be necessary to improve the performance of the agent, such as trying different neural network architectures, adjusting the hyperparameters, or implementing more sophisticated exploration and exploitation strategies. Additionally, it may be helpful to visualize the agent's behavior during training and evaluation to gain more insights into its strengths and weaknesses.
(mean: -0.2, std: 0.979)

4- Change the size of the samples:
The output shows the training and evaluation results of the QLearningNNReplaySamples agent in the Catch game environment. During training, the agent plays 100 episodes and updates its Q-value estimates based on a fixed number of transitions sampled from the replay buffer for each update step. At each timestep, the agent selects an action based on an epsilon-greedy exploration strategy, where it chooses the best action based on its Q-value estimates with probability 1-epsilon and a random action with probability epsilon.
The training output shows the return of each episode, which is the sum of rewards obtained during the episode. The mean return is also shown, which is the average return over the last 10 episodes. As we can see, the agent performs similarly as the previous agents during training, with a mean return of around -0.5 to -0.6, indicating that it fails to catch many fruits.
After training, the agent is evaluated on 10 episodes using the greedy policy (i.e., epsilon=0), where it always chooses the action with the highest Q-value estimate. The output shows the return of each episode and the mean and standard deviation of the returns. The evaluation results are also poor, with a mean return of -0.6 and a high standard deviation of 0.8, indicating that the agent still fails to catch many fruits even with the greedy policy.
Overall, the performance of the QLearningNNReplaySamples agent is similar to the previous Q-learning agents, and all agents perform poorly in the Catch game environment. This suggests that further modifications may be necessary to improve the performance of the agent, such as trying different neural network architectures, adjusting the hyperparameters, or implementing more sophisticated exploration and exploitation strategies. Additionally, it may be helpful to visualize the agent's behavior during training and evaluation to gain more insights into its strengths and weaknesses.
(mean: -0.6, std: 0.8)