Assignment 04 Mohammad

Implementation

Implement DQN

Deep Q Network is implemented in `run_main_dqn.py`. The stable baselines framework was used to implement this.

> 💡 At least one other Deep RL algorithm of your choice or own design.

The other algorithm chosen is Double DQN and it is implemented in `run_main_dqn.py`. The mode to switch between DQN and Double DQN is controlled by a boolean flag. The stable baselines framework was used to implement this.

> 💡 Describing each algorithm you used, define the states, actions, dynamics. Define the mathematical formulation of your algorithm, show the Bellman updates you use. Clearly specify the different hyper-parameters values (learning rate, discount factor, epsilon etc) you used for the algorithms and justify the reasons for these choices.

Overview of all Hyper-Parameters Used

Throughout this report, various hyper-parameters will be discussed. Below is their description, range and symbols.

The Hyper-Parameters used are

- γ : Discount Rate
- α : Learning Rate / StepSize

Discount rate γ controls how much to discount future rewards and is in the range $[0, 1]$.

Learning rate α determines the speed of "learning" and is in the range of $[0, 1]$. This is sometimes known as step-size.

Dynamics

All algorithms are performed on grid-world with the same dynamics function.

$$p(s\prime, r|s, a) = Pr\{S_t = s\prime, R_t = r|S_{t-1} = s, A_t = a\}$$

The environment is deterministic, therefore a valid transition will have a probability of 1, and everything else will have a probability of 0.

Deep Q Network

Description

Deep Q Network is a parameterised Q learning. A neural net is used instead of a Q table. The input to the neural network is the current state, and the output is the Q values for all the actions that can be taken from the state.

States

DQN does not store state information other then what is in the neural network. The neural network takes as input 4 coordinates, signifying the agents location as input, and outputs the estimated Q values for each action.

Actions

Actions are chosen via the ϵ greedy method, and is implemented in the stable baselines implementation of Double DQN.

Mathematical Formulation

Below is the update rule used to train the neural network inside the stable baselines framework.

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underbrace{\alpha}_{\text{Learning Rate}}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underbrace{\gamma}_{\text{Discount rate}} \underbrace{max\,Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

Hyper Parameters

The Hyper-Parameters used are:

- γ : Discount Rate = 0.99

- α : Learning Rate = 0.001

A small learning rate was used to avoid large updates to the Q table.

A large discount rate γ was used to give the agent a larger (forward) horizon for rewards.

Double Q Network

Description

Double Q network is analogous to Double Q Learning, where Q values are sampled from two separate "tables". However, in this case, it is no longer a table but a neural network. Double Q Network uses a target model that is used to get the greedy action but uses a separate model for the updates. This is to remove bias and according to google deep mind, converge faster (double the speed of regular DQN).

States

Double DQN does not store state information other then what is in the neural network. The neural network takes as input 4 coordinates, signifying the agent's location as input, and outputs the estimated Q values for each action.

Actions

Actions are chosen via the ϵ greedy method and is implemented in the stable-baselines implementation of Double DQN.

Mathematical Formulation

Below is the update rule used to train the neural network inside the stable baselines framework. One model is the model used for predictions, the other is the target model used for updates.

$$Q_{t+1}^{A}(s_t, a_t) = Q_t^{A}(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^{B}\left(s_{t+1}, \arg\max_a Q_t^{A}(s_{t+1}, a)\right) - Q_t^{A}(s_t, a_t)\right), \text{ and}$$

$$Q_{t+1}^{B}(s_t, a_t) = Q_t^{B}(s_t, a_t) + \alpha_t(s_t, a_t)\left(r_t + \gamma Q_t^{A}\left(s_{t+1}, \arg\max_a Q_t^{B}(s_{t+1}, a)\right) - Q_t^{B}(s_t, a_t)\right).$$

Hyper Parameters

The Hyper-Parameters used are:

- γ : Discount Rate = 0.99
- α : Learning Rate = 0.001

A small learning rate was used to avoid large updates to the Q table.

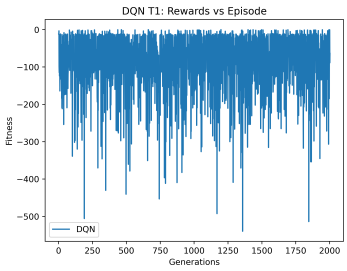
A large discount rate γ was used to give the agent a larger (forward) horizon for rewards.

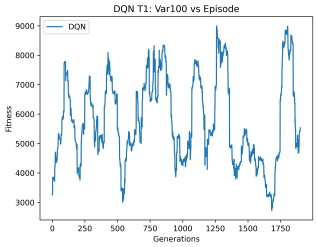
Quantitative Analysis

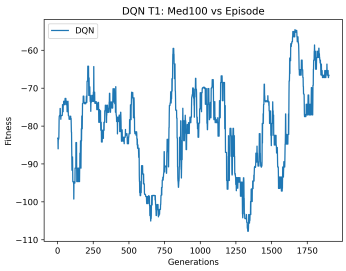
Some quantitative analysis of the results, a default plot for comparing all algorithms is given. You can do more plots than this.

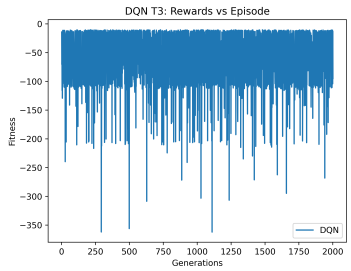
Deep Q Network (DQN)

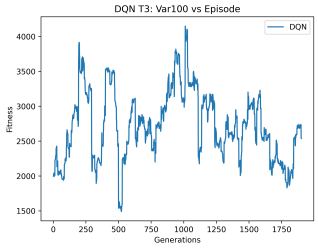
The graphs for the different tasks are present below. From the graphs, it is visible that no task reached convergence. For future experimentation, different hyper-parameters should be used, as well as more episodes should be used. Due to time constraints, these are not possible at this time.

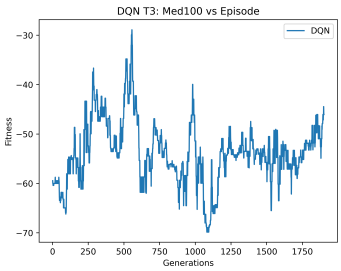






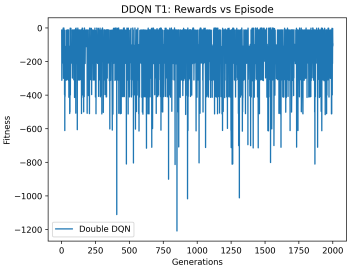


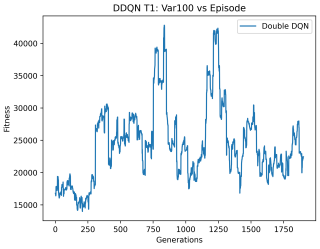


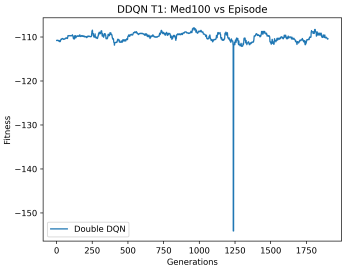


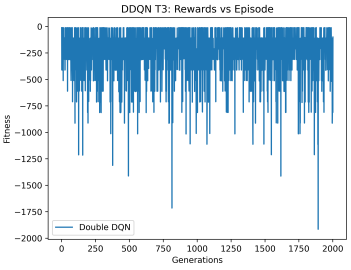
Double Deep Q Network

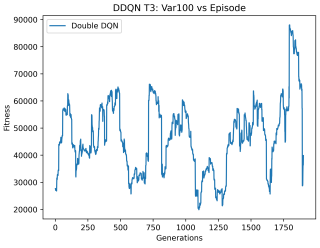
The graphs for the different tasks are present below. From the graphs, it is visible that no task reached convergence. For future experimentation, different hyper-parameters should be used, as well as more episodes should be used. Due to time constraints, these are not possible at this time.

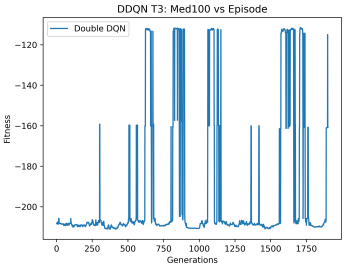












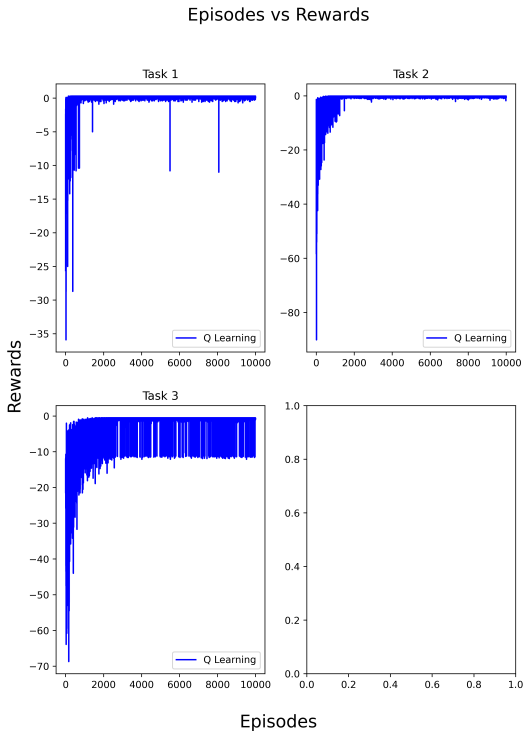
Deep Q Network vs Double Q Network

According to google deep mind, Double Q Network usually converges twice as fast as regular DQN, however this could not be confirmed in this experiment. A reason for lack of convergence other than choice of hyper parameters may be the

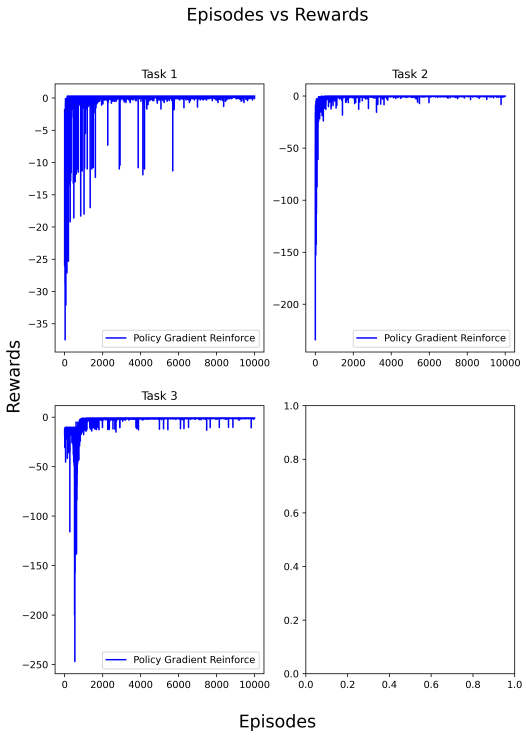
choice of feature representation. For future tests, on-hot encoding of the environment should be used opposed to just plain coordinates.

Comparison to Q Learning & Policy Gradient

> 💡 Compare the results of these algorithms to the others that you had implemented in Assignment 2. Give reasons for the observed comparative performances.



Q Learning



Policy Gradient

```
Task 0, Q Learning :
  max reward = 0.30000000000000004
  medLast100=0.30000000000000004
  varLast100=0.021699999999999994
Task 1, Q Learning :
  max reward = -0.09999999999999987
  medLast100=-0.30000000000000004
  varLast100=0.07138400000000013
```

```
Task 0, Policy Gradient Reinforce :
  max reward = 0.30000000000000004
  medLast100=0.30000000000000004
  varLast100=0.006216
Task 1, Policy Gradient Reinforce :
  max reward = -0.09999999999999987
  medLast100=-0.09999999999999987
  varLast100=0.012675000000000023
```

```
Task 2, Q Learning :
  max reward = -0.5000000000000002
  medLast100=-0.7000000000000004
  varLast100=3.3708360000000006
```

```
Task 2, Policy Gradient Reinforce :
  max reward = -0.7000000000000004
  medLast100=-0.9000000000000006
  varLast100=0.03592400000000003
```

Both algorithms performed better than the implemented DQN and Double DQN. This is due to either a bad choice of hyper-parameters or bad choice of feature representation. However, if DQN or Double DQN were to converge, they would be better choices as they do not have a reliance on a Q table. The two algorithms just need to store the neural network models.

Qualitative Analysis

Some qualitative analysis of you observations where one algorithm works well in each case, what you noticed along the way, explain the differences in performance related to the algorithms.

Although it could not be confirmed if Double DQN converges faster then regular DQN, the conversion between the two algorithms is trivial. As such, if double DQN did provide better results in this experiment it would be deemed better then DQN. Double DQN uses the same amount of space as regular DQN. The advantage of these deep RL algorithms is the lack of reliance on a Q table. However, as neural nets seem somewhat like a black box, it is hard to debug them at times.

It should be noted that a version of the two algorithms were attempted using tensor-flow keras, however the two algorithms where un-optimized and never converged (and took too long to run). The code for these two can be found under `./deep_rl`.