Reinforcement Learning Coursework 3

WU Zhi   17040772   UCABWUA@UCL.AC.UK
Department of Computer Science
University College London

Part 1: Implement Agents

1.2 Linear Model

The parameters of all these linear transformations must be trained by gradient descent. Write down the update to the parameters of the models and implement the update in the model below.

Assuming the loss function is $J(\theta)$, S' is the next state, R is reward and G is the discount gained from environment. Therefore, the loss function can be written as:

$$J(T_a) = \frac{1}{2}(S' - T_a s)^2$$

$$J(R_a) = \frac{1}{2}(R - R_a s)^2$$

$$J(G_a) = \frac{1}{2}(G - G_a s)^2$$

Therefore, the update equations of parameters are:

$$T_a = T_a - \alpha\Delta(T_a) = T_a - \alpha\frac{\partial J(T_a)}{\partial T_a} = T_a - \alpha(S' - T_a s) * -s$$

$$R_a = R_a - \alpha\Delta(R_a) = R_a - \alpha\frac{\partial J(R_a)}{\partial R_a} = R_a - \alpha(R - R_a s) * -s$$

$$G_a = G_a - \alpha\Delta(G_a) = G_a - \alpha\frac{\partial J(G_a)}{\partial G_a} = G_a - \alpha(G - G_a s) * -s$$

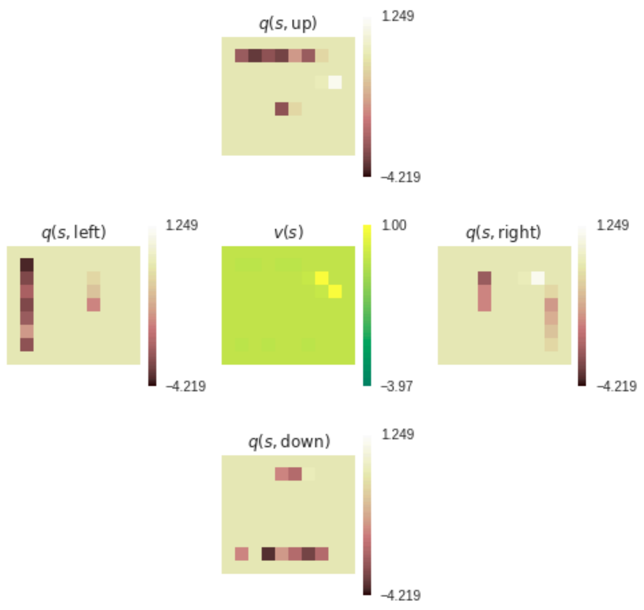
Where α is step_size

Part 2: Analyse Results

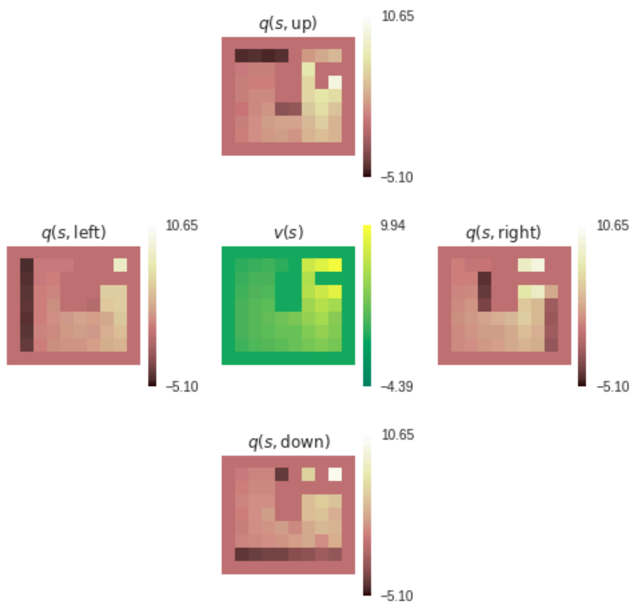
2.1 Tabular Learning

2.1.1 Data Efficiency

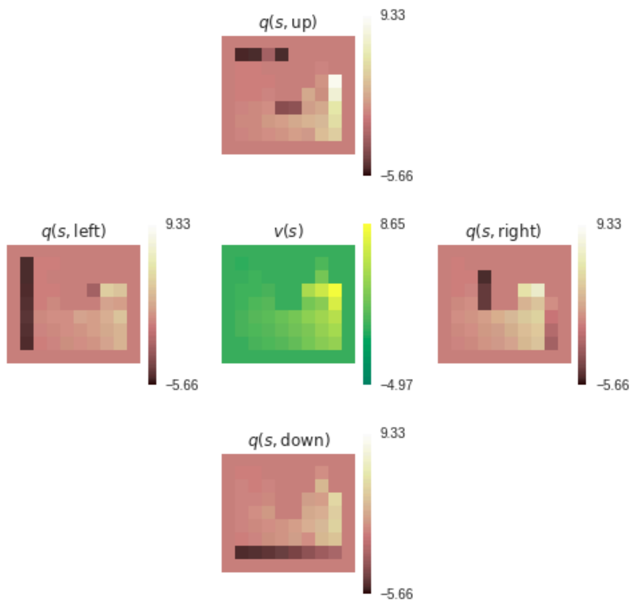
Online Q-learning



Experience Replay

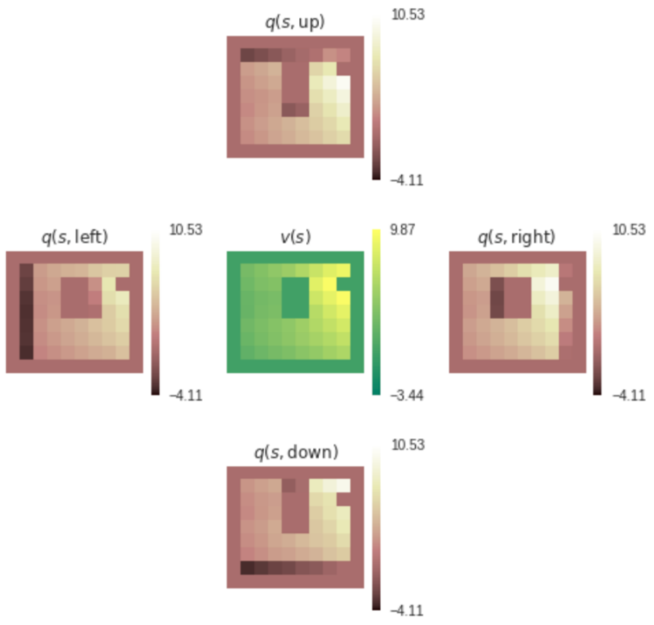


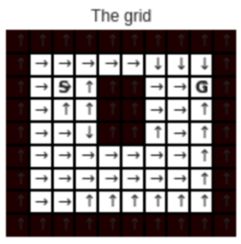
DynaQ



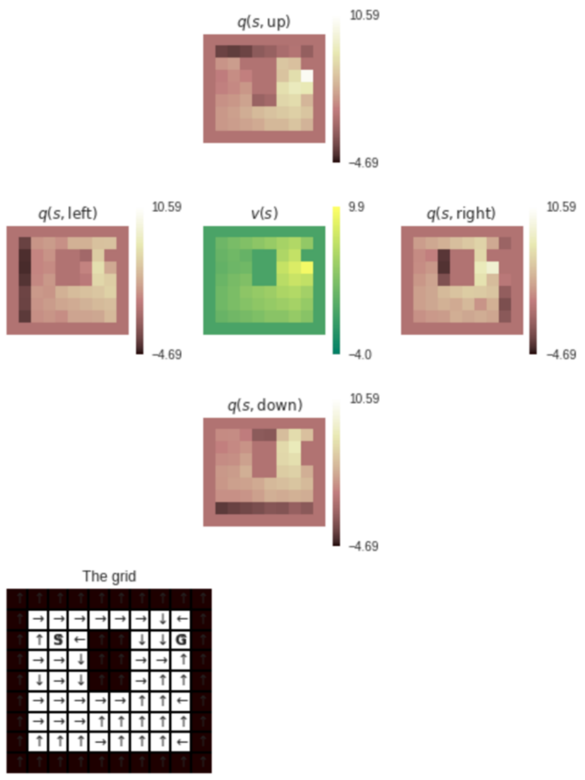
2.1.2 Computational Cost

Online Q-learning

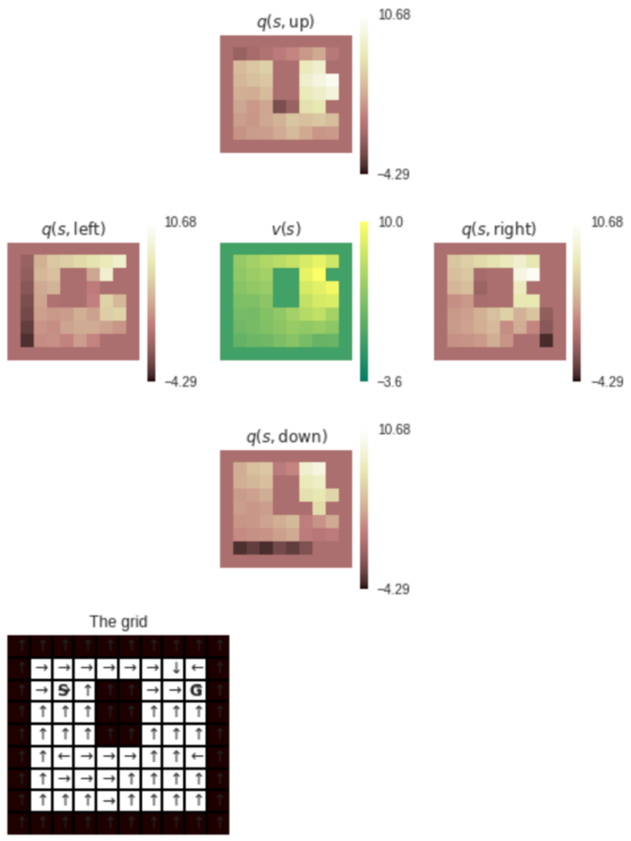




Experience Replay

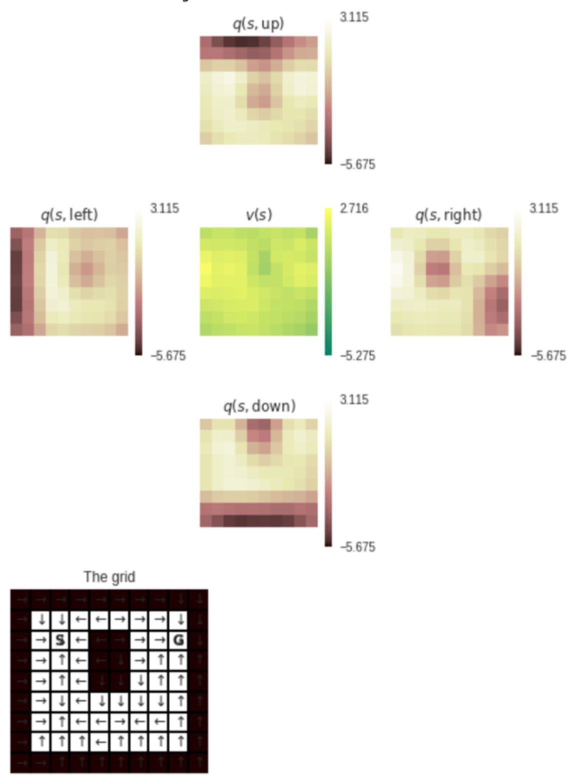


DynaQ

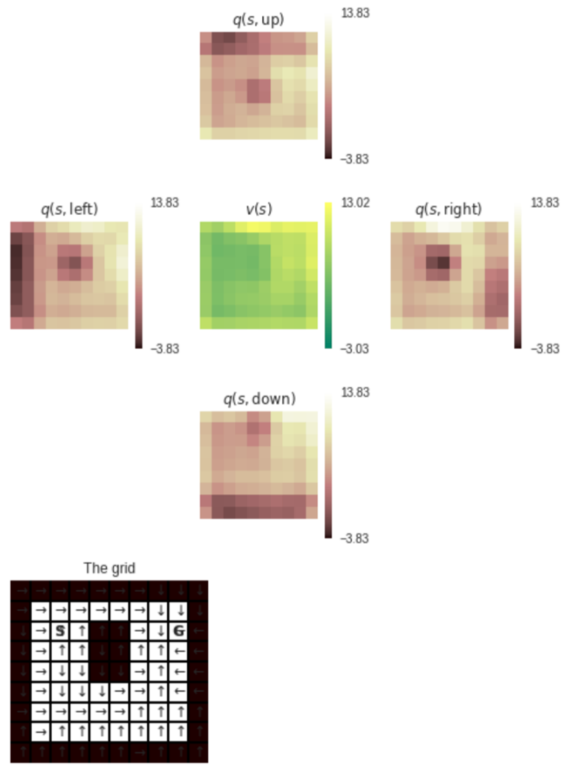


2.3 Linear function approximation

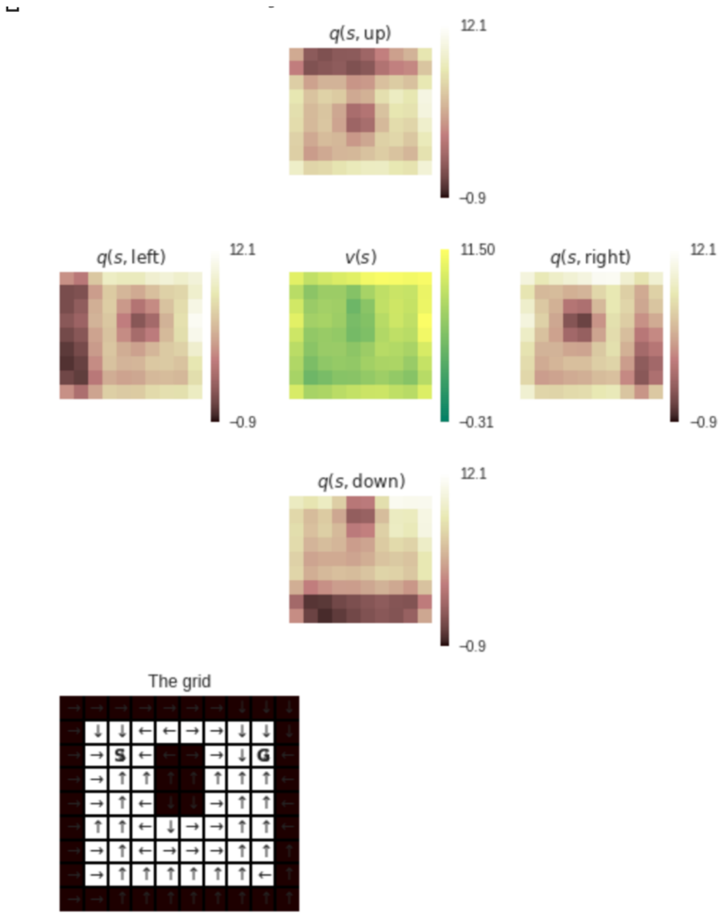
Online Q-learning with Linear Function Approximation



Experience Replay with Linear Function Approximation

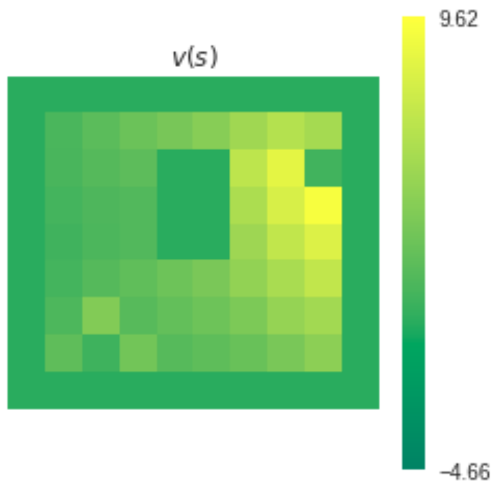


DynaQ with Linear Function Approximation

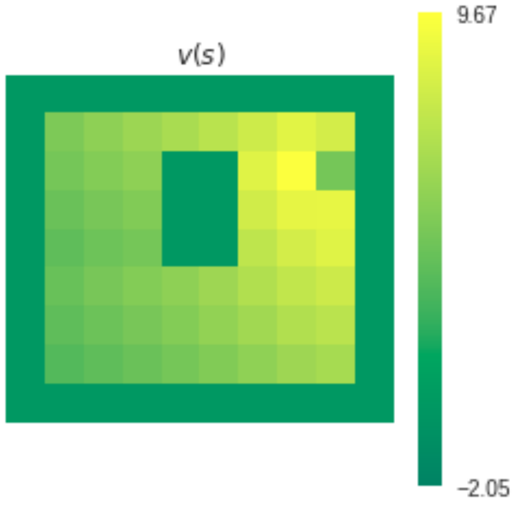


2.4 Non stationary Environments

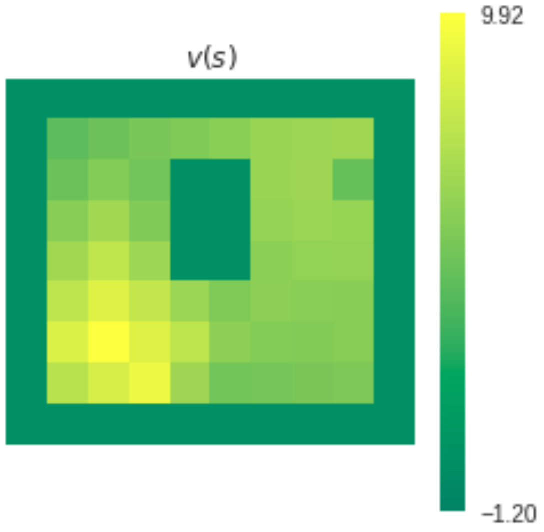
Online Q-learning



Experience Replay



Dyna



Question

Basic Tabular Learning

[5 pts] Why is the ExperienceReplay agent so much more data efficient than online Q-learning?

Answer:

```
ExperienceReplay agent use previous experience, and learn with it multiple times.
This is key when gaining experience is costly, you can get full use of it. The Q-
learning updates are incremental and do not converge quickly, so multiple passes
with the same data is beneficial, especially when there is low variance in
immediate outcomes (reward, next state) given the same state, action pair.
```

[5 pts] If we run the experiments for the same number of updates, rather than the same number of steps in the environment, which among online Q-learning and Experience Replay performs better? Why?

Answer:

When the number of updates is same, online Q-learning will perform better.

For online Q-learning, it update once per step, and it always update the latest experience. For experience Replay, it learnt old experience many times in one step. If the number of update is same, online Q-learning can have more steps, and it means it can not have more experience, so it perform better than experience replay.

[5 pts] Which among online Q-learning and Dyna-Q is more data efficient? why?

Answer:

Dyna-Q have more data efficient.

Dyna-Q agent use previous experience, and learn with it multiple times. The difference between Dyna-Q and experience replay is the value in transition of Dyna-Q will update if new experience input. The Q-learning updates are incremental and do not converge quickly, so multiple passes with the same data is beneficial, especially when there is low variance in immediate outcomes (reward, next state) given the same state, action pair.

[5 pts] If we run the experiments for the same number of updates, rather than the same number of steps in the environment, which among online Q-learning and Dyna-Q performs better? Why?

Answer:

When the number of updates is same, online Q-learning will perform better. Although Dyna-Q perform good, but it is not stable, whereas online Q-learning is good and stable.

For online Q-learning, it update once per step, and it always update the latest experience, so it is incremental. For Dyna-Q, it learnt old experience many times in one step. It update transition when replay bufffer update. However, if the number of update is same, online Q-learning can have more steps, and it means it can not have more experience, and it may cause part of action value converge quickly, but it is not stable for training process.

Linear function approximation

[5 pts] The value estimates with function approximation are considerably more blurry than in the tabular setting despite more training steps and interactions with the environment, why is this the case?

Answer:

When the value estimates with function approximation, the predicted reward, discount and next state will all be a linear function of the state. However, in the real world, predicted reward, discount and next state may be non-linear model, and Our feature selection is not perfect to provide enough information and can not perfectly represent each state, so linear function can not converge action value exactly, so the lineae model is more blurry than the tabublar model.

[5 pts] Inspect the policies derived by training agents with linear function approximation on `FeatureGrid` (as shown by `plot_greedy_policy`). How does this compare to the optimal policy? Are there any inconsistencies you can spot? What is the reason of these?

Answer:

```
It is obvious that the policy does not converge to the optimal policy. Even near
the goal location, the policy shows wrong action. Becasuse the approximation
accuracy is fundamentally limited by the information provided by the features.
Hence, the feature selection is not perfect to provide enough information and can
not perfectly represent each state.
```

Learning in a non stationary environment

Consider now the tabular but non-stationary setting of section 2.4.

After an initial pretraining phase, the goal location is moved to a new location, where the agent is allowed to train for some (shorter) time.

[10 pts] Compare the value estimates of online Q-learning and Experience Replay, after training also on the new goal location, explain what you see.

Answer:

```
Compare the value estimates of online Q-learning and Experience Replay, although
both  online Q-learning and Experience Replay perform bad when the goal moved to
a new location, but online Q-learning perform better.

For online q-learning, it can be found that the action values around of new goal
location has increased sightly. The Q-learning updates are incremental and do not
converge quickly, and it update once in per step. Shorter time is not enough for
Q-learning to find a new goal, and it did not converge as well.

For experience Replay, the action value figure almost same as before (no new goal
added). In training process(old goal), the replay buffer keep adding new
experience. After goal location moving to new location, there are too much old
experience in replay buffer. Hence, in short training time, replay buffer just
have few new exprience(to new goal). In the loop repeat, the training data
(random in replay buffer) lead the action value to old goal because of very litte
new experience in replay buffer.
```

[10 pts] Compare the value estimates of online Q-learning and Dyna-Q, after training also on the new goal location, explain what you see.

Back up your observations with visualizations of the value/policy.

Answer:

```
Compare the value estimates of online Q-learning and Dyna-Q, Dyna-Q perform
better.

For online q-learning, it can be found that the action values around of new goal
location has increased sightly. The Q-learning updates are incremental and do not
converge quickly, and it update once in per step. Shorter time is not enough for
Q-learning to find a new goal, and it did not converge as well.

For Dyna-Q, the action values around new goal have high values. In training
process, although the replay buffer keep adding new experience, the transition
will update discount, reward and next_state if same state and action are added in
replay buffer, so Dyna-Q always learning the latest experience from model. After
goal location moving to new location, when new expericence (new goal) add in
replay buffer, the old experience (old goal) will be updated. In each step, the
repeat loop update the action value, so the result is acceptable.
```