Görkem Kadir Solun 22003214 – CS461 HW4

1 Informed Search

(a)

A* tree search with any admissible heuristic is guaranteed to find the optimal path.

A heuristic h′ is admissible if $h'(X) \le h^*(X)$ for all nodes X.

(i)

It is admissible as $\frac{h(X)}{2} = h'(X) \le h(X) \le h^*(X)$.

For $h'(X) = \frac{h(X)}{2} \to C = h^*(S)$

(ii)

It is admissible as $h'(X) = \frac{h(X)+h^*(X)}{2} \le \frac{h^*(X)+h^*(X)}{2} = h^*(X)$. This makes $C = h^*(S)$.

(iii)

It is NOT admissible as $0 \le h(X) \le h^*(X) \le h^*(X) + h(X) = h'(X)$. Heuristic h' may overestimate the true cost. Thus, the safest statement we can make is $C \ge h^*(S)$.

(iv)

$Y \in K(X)$ if Y is a neighbor closer to the goal than X in terms of the true cost h*.

Base Case $\boldsymbol{K(X)} = \emptyset$:

It is admissible as $h'(X) \le h^*(X)$.

Recursive Case: $\boldsymbol{K(X)} \ne \emptyset$:

(v)

Case $\boldsymbol{K(X)} = \emptyset$:

It is admissible as $h'(X) = h(X) \le h^*(X)$.

Case: $\boldsymbol{K(X)} \ne \emptyset$:

In this case $h'(X) = \min\left(h(Y) + cost(X,Y)\right)$, where $Y \in K(X)$.

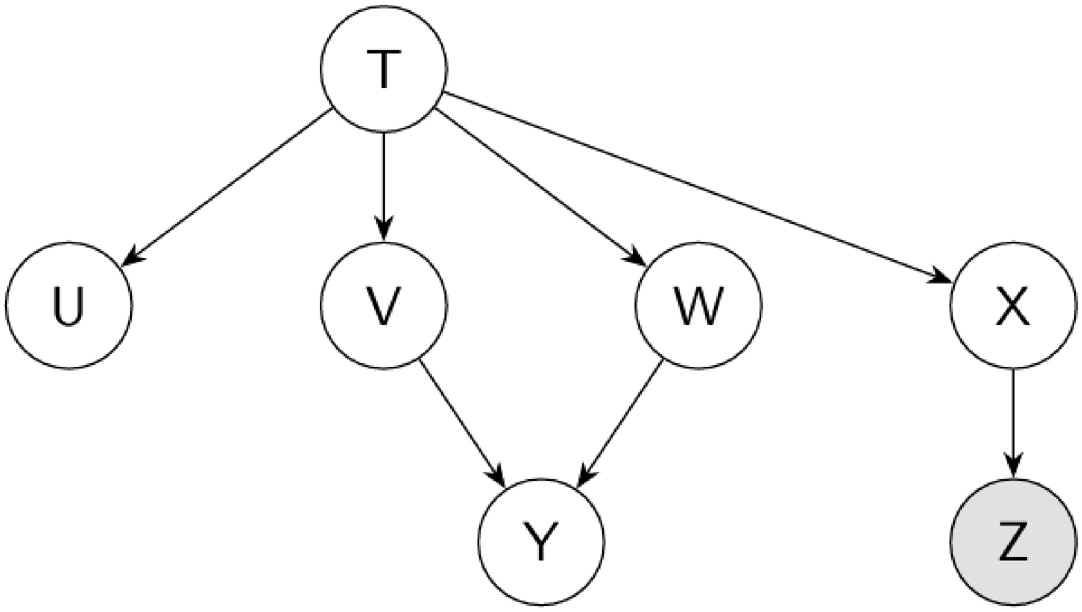
For any neighbor $Y \in K(X)$, we know $h(Y) \le h^*(Y)$. Thus, we get:

$h'(X) = h(Y) + cost(X,Y) \le h^*(Y) + cost(X,Y) = h^*(X)$ as $h^*(Y) + cost(X,Y) = h^*(X)$

Taking the minimum over all $Y \in K(X)$:

$h'(X) \leq h^*(X)$ which makes h' admissible for all cases. This makes $C = h^*(S)$.

2 Bayes Networks - Variable Elimination



1 Z=+z

Initial CPTs are $\{P(U \mid T), P(V \mid T), P(W \mid T), P(X \mid T), P(Y \mid V, W), P(Z \mid X), \mathrm{P(T)}\}$

When evidence $Z = +z$ is observed, the factor $P(Z \mid X)$ is updated. This results in:

$$\{\mathrm{P(T)}, P(U \mid T), P(V \mid T), P(W \mid T), P(X \mid T), P(Y \mid V, W), f(Z = +z|X)\}$$

2 X

$$f_1(Z = +z|T) = \sum_X P(X \mid T) f(Z = +z|X)$$

$$f_1(Z = +z|T) = (X = +x \mid T) f(Z = +z \mid X = +x) + P(X = -x \mid T) f(Z = +z \mid X = -x)$$

$$\{\mathrm{P(T)}, P(U \mid T), P(V \mid T), P(W \mid T), P(Y \mid V, W), f_1(Z = +z|T)\}$$

3 T

$$f_2(U, V, W) = \sum_T P(U \mid T) P(V \mid T) P(W \mid T) P(T) f_1(Z = +z|T)$$

$$f_2(U,V,W,Z=+z) = P(T=+t)P(U \mid T=+t)P(V \mid T=+t)P(W \mid T=+t)f_1(Z=+z|T=+t) + P(T=-t)P(U \mid T=-t)P(V \mid T=-t)P(W \mid T=-t)f_1(Z=+z|T=-t)$$

$$\{P(Y \mid V,W), f_2(U,V,W,Z=+z)\}$$

4 U

$$f_3(V,W,Z=+z) = \sum_U f_2(U,V,W,Z=+z)$$
$$= f_2(U=+u,V,W,Z=+z) + f_2(U=-u,V,W,Z=+z)$$

$$\{P(Y \mid V,W), f_3(V,W,Z=+z)\}$$

5 V

$$f_4(W,Y,Z=+z) = \sum_V P(Y|V,W)f_3(V,W,Z=+z) = P(Y|V=+v,W)f_3(V=+v,W,Z=+z) + P(Y|V=-v,W)f_3(V=-v,W,Z=+z)$$

$$\{f_4(W,Y,Z=+z)\}$$

6 W

$$f_5(Y,Z=+z) = \sum_W f_4(W,Y,Z=+z) = f_4(W=+w,Y,Z=+z) + f_4(W=-w,Y,Z=+z)$$

$$\{f_5(Y,Z=+z)\}$$

7 $P(Y|Z=+z)$

Normalize to get

$$P(Y|Z=+z) = \frac{f_5(Y,Z=+z)}{\sum_Y f_5(Y,Z=+z)}$$

$$P(Y=+y \mid Z=+z) = \frac{f_5(Y=+y,Z=+z)}{f_5(Y=-y,Z=+z) + f_5(Y=+y,Z=+z)}$$

$$P(Y=-y \mid Z=+z) = \frac{f_5(Y=-y,Z=+z)}{f_5(Y=-y,Z=+z) + f_5(Y=+y,Z=+z)}$$

8 Optimality

Given Order

Eliminating X

The factors involving X are $P(X \mid T)$ and $P(Z \mid X)$.

After eliminating X, a new factor $f_1(T)$ is generated, which depends only on T.

Eliminating T

The factors involving T are $P(T), P(U \mid T)$, $P(V \mid T)$, $P(W \mid T)$, and $f_1(T)$.

After eliminating T, a large factor $f_2(U, V, W)$ is generated, involving U, V, and W.

Eliminating U

The factor $f_2(U, V, W)$ depends on U, so eliminating U produces a new factor $f_3(V, W)$.

Eliminating V

The factors involving V are $f_3(V, W)$ and $P(Y \mid V, W)$.

After eliminating V, a factor $f_4(W, Y)$ is generated.

Eliminating W

The factor $f_4(W, Y)$ depends on W, so eliminating W produces a final factor $f_5(Y)$.

Observation

Eliminating T early introduces a large intermediate factor $f_2(U, V, W)$, which involves three variables. This step could have been avoided with a different order.

New Order

To reduce the size of intermediate factors, variables should be eliminated in an order that minimizes dependencies. A better order is U, X, W, V, T.

Eliminate U first because it depends only on T, producing a small factor.

Eliminate X next because it involves only T and Z, keeping the factor sizes small.

Eliminate W next because it reduces the dependencies on V and Y.

Eliminate V before T to avoid creating large factors involving multiple variables.

Finally, eliminate T, reducing the computational load significantly.

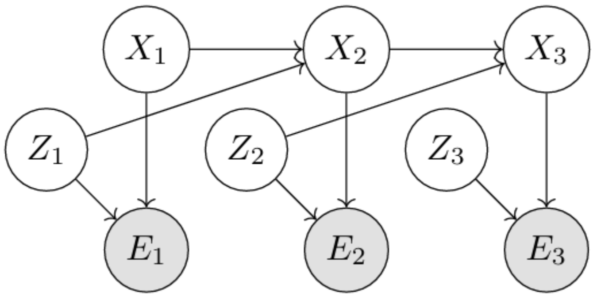
3 HMM-Filtering

Consider the two HMM-like models below. Find the elapse time $(P(X_t, Z_t|e_{1:t-1}))$ and observation $(P(X_t, Z_t|e_{1:t}))$ updates.

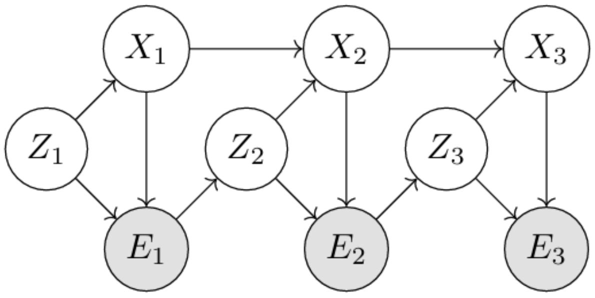
Hint: Recall that for standard HMMs, they are in the following forms, respectively:

$$P(X_t|e_{1:t-1}) = \sum_{x_{t-1}} P(X_t|x_{t-1})P(x_{t-1}|e_{1:t-1})$$

$$P(X_t|e_{1:t}) \propto P(X_t|e_{1:t-1})P(e_t|X_t)$$



(a)



(b)

(a)

Elapse Time

$$\begin{aligned} P(X_t, Z_t|E_{1:t-1}) &= \sum_{x_{t-1}, z_{t-1}} P(X_t, Z_t, X_{t-1}, Z_{t-1}|E_{1:t-1}) \\ &= \sum_{x_{t-1}, z_{t-1}} P(X_t, Z_t|X_{t-1}, Z_{t-1}, E_{1:t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1}) \end{aligned}$$

As the next state conditionally independent of the old observations by Markov property.

$$P(X_t, Z_t|E_{1:t-1}) = \sum_{x_{t-1},z_{t-1}} P(X_t, Z_t|X_{t-1}, Z_{t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$
$$= \sum_{x_{t-1},z_{t-1}} P(X_t|Z_t, X_{t-1}, Z_{t-1})P(Z_t|X_{t-1}, Z_{t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$
$$= \sum_{x_{t-1},z_{t-1}} P(X_t|X_{t-1}, Z_{t-1})P(Z_t)P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$

For the time being there is no E_t that can make Z_t dependent. This makes $Z_t \perp \{X_t, X_{t-1}, Z_{t-1}\}$.

Observation

$$P(X_t, Z_t|E_{1:t}) = \frac{P(X_t, Z_t, E_t|E_{1:t-1})}{P(E_t|E_{1:t-1})} = \frac{P(E_t|E_{1:t-1}, X_t, Z_t)P(X_t, Z_t|E_{1:t-1})}{P(E_t|E_{1:t-1})}$$

As $E_{1:t-1} \perp E_t \mid X_t$ from the structure (BN),

$$P(X_t, Z_t|E_{1:t}) = \frac{P(E_t|X_t, Z_t)P(X_t, Z_t|E_{1:t-1})}{P(E_t|E_{1:t-1})}$$

$P(E_t|E_{1:t-1})$ is normalization, we can remove if we want brevity. Exact calculation of normalization is:

$$P(E_t|E_{1:t-1}) = \sum_{x_t,z_t} P(E_t|X_t, Z_t)P(X_t, Z_t|E_{1:t-1})$$

Don't forget that we know $P(X_t, Z_t|E_{1:t-1})$ from elapse time.

(b)

Elapse Time

$$P(X_t, Z_t|E_{1:t-1}) = \sum_{x_{t-1},z_{t-1}} P(X_t, Z_t, X_{t-1}, Z_{t-1}|E_{1:t-1})$$
$$= \sum_{x_{t-1},z_{t-1}} P(X_t, Z_t|X_{t-1}, Z_{t-1}, E_{1:t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$
$$= \sum_{x_{t-1},z_{t-1}} P(X_t|X_{t-1}, Z_t, Z_{t-1}, E_{1:t-1})P(Z_t|X_{t-1}, Z_{t-1}, E_{1:t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$

From the structure (BN) we can deduce that $X_t \perp E_{1:t-1}, Z_{t-1} \; given \; X_{t-1}, Z_t$.

From the structure (BN) we can deduce that $Z_t \perp Z_{t-1}, X_{t-1} \; given \; E_{1:t-1}$.

$$P(X_t, Z_t|E_{1:t-1}) = \sum_{x_{t-1},z_{t-1}} P(X_t|X_{t-1}, Z_t)P(Z_t|E_{1:t-1})P(X_{t-1}, Z_{t-1}|E_{1:t-1})$$

Observation

$$P(X_t, Z_t | E_{1:t}) = \frac{P(X_t, Z_t, E_t | E_{1:t-1})}{P(E_t | E_{1:t-1})} = \frac{P(E_t | E_{1:t-1}, X_t, Z_t) P(X_t, Z_t | E_{1:t-1})}{P(E_t | E_{1:t-1})}$$

As $E_t \perp E_{1:t-1} |\, X_t, Z_t$ from the structure (BN),

$$P(X_t, Z_t | E_{1:t}) = \frac{P(E_t | X_t, Z_t) P(X_t, Z_t | E_{1:t-1})}{P(E_t | E_{1:t-1})}$$

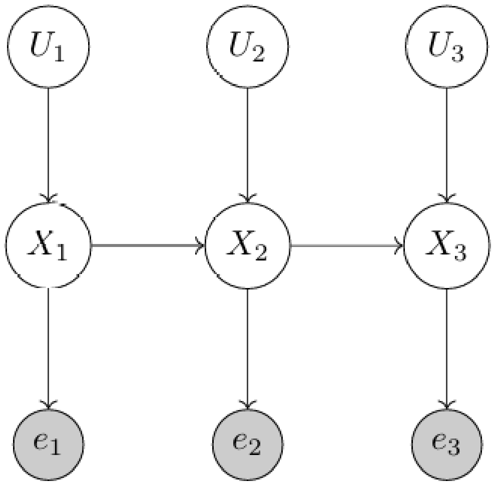
$P(E_t | E_{1:t-1})$ is normalization, we can remove if we want brevity. Exact calculation of normalization is:

$$P(E_t | E_{1:t-1}) = \sum_{x_t, z_t} P(E_t | X_t, Z_t) P(X_t, Z_t | E_{1:t-1})$$

Don't forget that we know $P(X_t, Z_t | E_{1:t-1})$ from elapse time.

4 HMM: Viterbi Algorithm

Consider the HMM structure below. The agent performs an action U_t from some state-independent policy at every time step t, and also receives some evidence e_t of the true state X_t.



In this setting, which (if any) of the expressions below are maximized by the Viterbi algorithm? Explain your reasoning below.
Hint: Remember that the Viterbi algorithm for vanilla HMMs with no control actions update is:

$$m_{1:t+1} = P(e_{t+1} | X_{t+1}) \cdot \max_{x_t} P(X_{t+1} | x_t) \cdot m_{1:t}$$

The **Viterbi algorithm** finds the most probable sequence of hidden states ($X_{1:t}$) given the observed evidence ($e_{1:t}$). At each step, it updates the likelihood of the most probable path leading to each state at time t+1 using a recursive relationship:

$$m_{1:t+1}(X_{t+1}) = P(e_{t+1}|X_{t+1}) \max_{x_t}(P(X_{t+1}|X_t)m_{1:t}(X_t))$$

The key points:

- $m_{1:t}(x_t)$ is the maximum probability of the most likely state sequence up to time t ending in state x_t.
- The algorithm focuses on maximizing the joint probability of the sequence of states and observations.

Once we have actions U_t in the mix, the assumption in a controlled HMM is that:

The next state depends on the previous state and the current action: $P(X_t|X_{t-1}, U_t)$.

The observation depends on the current state: $P(E_t|X_t)$.

The actions themselves can be either exogenously specified or generated from some policy ($P(U_t|X_t)$ or state-independent).

Then the joint distribution that is relevant to the hidden-state sequence looks like:

$$P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1}, U_t)P(E_t|X_t)$$

Viterbi in that case would search

$$\max_{X_{1:T}} P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1}, U_t)P(E_t|X_t)$$

There is no factor $P(U_t|U_{t-1})$ in this objective if the actions are known inputs (we do not need to explain the actions, only the hidden states). Also, we have $P(X_t|X_{t-1}, U_t)$ as X_t depends on both X_{t-1} and U_t.

Viterbi update in this DBN would be:

$$m_{1:t}(X_t) = P(E_t|X_t) \max_{X_{t-1}} P(X_t|X_{t-1}, U_t)m_{1:t-1}(X_{t-1})$$

1 $P(X_{1:t})$

The Viterbi algorithm does **not** maximize $P(X_{1:t})$. This is the marginal probability of the sequence of states $X_{1:t}$, independent of the evidence $E_{1:t}$.

$P(X_{1:t})$ integrates over all possible evidence sequences, while the Viterbi algorithm conditions on the observed evidence $E_{1:t}$.

Viterbi is designed to find the most probable single sequence of states, not to marginalize over all possible sequences.

2 $P(U_{1:t})$

The Viterbi algorithm does **not** maximize $P(U_{1:t})$.

The actions U_t are independent of the hidden states X_t and evidence E_t. In this HMM structure, the actions U_t are state-independent, meaning $P(U_t|X_t, E_t) = P(U_t)$. Thus, the probability of an action sequence $P(U_{1:t}) = \prod_{t=1}^{T} P(U_t)$ depends solely on the prior probabilities of actions and is unaffected by the states or observations. The algorithm is designed to maximize the probability of the state sequence, not the action sequence.

3 $P(E_{1:t})$

The Viterbi algorithm does **not** maximize $P(E_{1:t})$.

$P(E_{1:t})$ involves summing over all possible hidden state sequences $X_{1:t}$, while the Viterbi algorithm selects only the single most likely sequence. The two are fundamentally different calculations.

The Viterbi algorithm maximizes the joint probability of the evidence and a single state sequence $P(X_{1:t}, E_{1:t})$, rather than marginalizing over all state sequences.

Computing $P(E_{1:t})$ requires the forward algorithm, which performs a sum-product recursion over all possible hidden states. This is distinct from the max-product recursion of the Viterbi algorithm.

4 $P(X_{1:t}, U_{1:t}|E_{1:t})$

The Viterbi algorithm does **not** maximize $P(X_{1:t}, U_{1:t}|E_{1:t})$.

Because the policy is state-independent, U_t does not depend on X_t. Typically, we would treat U_t as observed just like E_t, or at least as given from an external distribution.

If $U_{1:t}$ is actually observed or given, then

$$P(X_{1:t}, U_{1:t}|E_{1:t}) = P(X_{1:t}|E_{1:t}, U_{1:t})P(U_{1:t}|E_{1:t})$$

But $U_{1:t}$ is fixed in the data or experiment. Thus, there is no $\max_{X_{1:t}, U_{1:t}} P(X_{1:t}, U_{1:t}|E_{1:t})$ to do as those actions are known. Consequently, the Viterbi algorithm does not try to maximize $P(X_{1:t}, U_{1:t}|E_{1:t})$ as a function of both X and U. Rather, U is given or observed, so Viterbi only needs to find the most likely $X_{1:t}$. Because there is no reason to maximize over the known actions $U_{1:t}$.

5 $P(X_{1:t}, U_{1:t}, E_{1:t})$

The Viterbi algorithm does **not** maximize $P(X_{1:t}, U_{1:t}, E_{1:t})$. Same reasoning as the question 4 can be applied.

The expression $P(X_{1:t}, U_{1:t}, E_{1:t})$ expands as:

$$P(X_{1:t}, U_{1:t}, E_{1:t}) = P(U_{1:t})P(X_{1:t}|U_{1:t})P(E_{1:t}|X_{1:t})$$

$P(U_{1:t})$ is the probability of the sequence of actions, typically determined by the policy. If the actions are independent of states and evidence, this is just a constant term.

$P(X_{1:t}|U_{1:t})$ is the state sequence probabilities conditioned on the actions.

$P(E_{1:t}|X_{1:t})$ is the probability of the evidence given the state sequence.

The Viterbi algorithm does not explicitly maximize $P(U_{1:t})P(X_{1:t}|U_{1:t})P(E_{1:t}|X_{1:t})$, as the actions U_t are typically determined by a policy external to the algorithm. Therefore, the inclusion of $U_{1:t}$ in $P(X_{1:t}, U_{1:t}, E_{1:t})$ means this expression is not directly maximized.

6 $P(U_1)P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{t} P(U_t|U_{t-1})P(X_t|U_t)P(E_t|X_t)$

The Viterbi algorithm does **not** maximize

$$P(U_1)P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{t} P(U_t|U_{t-1})P(X_t|U_t)P(E_t|X_t)$$

None of the provided products match what the Viterbi algorithm for this DBN is really doing, because the given product omits the dependence on X_{t-1} in $P(X_t|X_{t-1}, U_t)$ and unnecessarily includes $P(U_t|U_{t-1})$.

If U_t are known exogenous inputs, then the Viterbi objective for the hidden states is proportional to:

$$P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{T}[P(X_t|X_{t-1}, U_t)P(E_t|X_t)]$$

That is consistent with the Bayes-net structure shown and is what the standard controlled-HMM Viterbi recursion implements.

7 $P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{t} P(U_t|X_{t-1})P(X_t|X_{t-1})P(E_t|X_t)$

The Viterbi algorithm does **not** maximize

$$P(X_1|U_1)P(E_1|X_1)\prod_{t=2}^{t}P(U_t|X_{t-1})P(X_t|X_{t-1})P(E_t|X_t)$$

If the actions U_t are known and fixed in advance, or determined by a known policy that does not depend on the hidden state, then from the viewpoint of filtering or smoothing over X_t, we just treat U_t as known inputs to the transition. In other words, once you condition on the fact that the action U_t happened exogenously, you no longer multiply by $P(U_t|X_{t-1})$ instead you use the transition $P(X_t|X_{t-1}, U_t)$.

$$P(U_t|X_{t-1})P(X_t|X_{t-1})P(E_t|X_t) \neq P(X_t|X_{t-1}, U_t)P(E_t|X_t)$$

If the policy is truly state-independent, then $P(U_t|X_{t-1})$ would collapse to a constant $P(U_t)$, which would factor out of the Viterbi maximization anyway.

5 1D Gridworld MDP

Consider a 1D gridworld MDP. In this MDP, available actions are *left*, *right*, and *stay*. Stay always results in the agent staying in its current square. Left and right are successful in moving in the intended direction **half of the time**. The other half of the time, the agent stays in its current square. An agent cannot try to move left at the leftmost square, and cannot try to move right on the rightmost square. Staying still on a square gives a reward equivalent to the number on that square and all other transitions give zero reward (meaning any transitions in which the agent moves to a different square give zero reward). Initially, let $V_0(s) = 0, \forall s$ and $\gamma = 0.5$.

4	0	0	36

Perform value iteration and put the values you find in the given empty grids.

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V_k(s')]$$

$P(s'|s,a)$ is the probability of transitioning to s' from s using action a.

$R(s,a,s')$ is the reward received from transitioning from s to s' with action a.

$\gamma = 0.5$ is the discount factor and $V_0(s) = 0\ \forall s$.

Left: Moves left with probability 0.5, stays with probability 0.5.

Right: Moves right with probability 0.5, stays with probability 0.5.

Stay: Always stays in the current square.

(a) $V_1(s)$ if an unwanted stay is rewarded 0.

S=1

Stay: $4 + 0.5V_0(1) = 4 + 0.5 \cdot 0 = 4$

Right: $0.5\big(0 + 0.5V_0(2)\big) + 0.5\big(0 + 0.5V_0(1)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(1) = max(4,0) = 4$$

S=2

Stay: $0 + 0.5V_0(2) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_0(1)\big) + 0.5\big(0 + 0.5V_0(2)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_0(3)\big) + 0.5\big(0 + 0.5V_0(2)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(2) = max(0,0,0) = 0$$

S=3

Stay: $0 + 0.5V_0(3) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_0(2)\big) + 0.5\big(0 + 0.5V_0(3)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_0(4)\big) + 0.5\big(0 + 0.5V_0(3)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(3) = max(0,0,0) = 0$$

S=4

Stay: $36 + 0.5V_0(4) = 36 + 0.5 \cdot 0 = 36$

Left: $0.5\big(0 + 0.5V_0(3)\big) + 0.5\big(0 + 0.5V_0(4)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(4) = max(36,0) = 36$$

4	**0**	**0**	**36**

(b) $V_2(s)$ if an unwanted stay is rewarded 0

S=1

Stay: $4 + 0.5V_1(1) = 4 + 0.5 \cdot 4 = 6$

Right: $0.5\big(0 + 0.5V_1(2)\big) + 0.5\big(0 + 0.5V_1(1)\big) = 0.5(0) + 0.5(0.5 \cdot 4) = 1$

$$V_2(1) = max(6,1) = 6$$

S=2

Stay: $0 + 0.5V_1(2) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_1(1)\big) + 0.5\big(0 + 0.5V_1(2)\big) = 0.5(2) + 0.5(0) = 1$

Right: $0.5\big(0 + 0.5V_1(3)\big) + 0.5\big(0 + 0.5V_1(2)\big) = 0.5(0) + 0.5(0) = 0$

$$V_2(2) = max(0,1,0) = 1$$

S=3

Stay: $0 + 0.5V_1(3) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_1(2)\big) + 0.5\big(0 + 0.5V_1(3)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_1(4)\big) + 0.5\big(0 + 0.5V_1(3)\big) = 0.5(0.5 \cdot 36) + 0.5(0) = 9$

$$V_2(3) = max(0,0,9) = 9$$

S=4

Stay: $36 + 0.5V_1(4) = 36 + 0.5 \cdot 36 = 54$

Left: $0.5\big(0 + 0.5V_1(3)\big) + 0.5\big(0 + 0.5V_1(4)\big) = 0.5(0) + 0.5(0.5 \cdot 36) = 9$

$$V_2(4) = max(54,9) = 54$$

6	**1**	**9**	**54**

(c) $V^*(s)$ if an unwanted stay is rewarded 0

S=4

Optimal action is staying.

$$V^*(4) = 36 + 36\gamma^1 + 36\gamma^2 + \cdots$$

$$V^*(4) = \frac{36}{1-\gamma} = 72$$

S=3

Stay: $0 + 0.5V^*(3)$

Left: $0.5\big(0 + 0.5V^*(2)\big) + 0.5\big(0 + 0.5V^*(3)\big) = 0.25V^*(2) + 0.25V^*(3)$

Right: $0.5(0 + 0.5V^*(4)) + 0.5(0 + 0.5V^*(3)) = 0.25V^*(4) + 0.25V^*(3) = 18 + 0.25V^*(3)$

The maximum is certainly the right one. Solving it gives 24.

$$V^*(3) = 18 + 0.25V^*(3) = 24$$

S=2

Stay: $0 + 0.5V^*(2)$

Left: $0.5(0 + 0.5V^*(1)) + 0.5(0 + 0.5V^*(2)) = 0.25V^*(1) + 0.25V^*(2)$

Right: $0.5(0 + 0.5V^*(3)) + 0.5(0 + 0.5V^*(2)) = 0.25V^*(3) + 0.25V^*(2) = 6 + 0.25V^*(2)$

We need to determine whether going left or right is the optimal choice. Moving right is optimal. Although we might consider staying at 1 to be optimal, it is not. Staying at 1 results in a value of 8 for position 1 (as calculated below). This contributes to a value of 2 $(0.25 \times 8 = 2)$, which is less than 6, making staying at 1 suboptimal.

$$V^*(3) = \max(2 + 0.25V^*(2), 6 + 0.25V^*(3)) = 8$$

S=1

Stay: $4 + 0.5V^*(1)$

Right: $0.5(0 + 0.5V^*(2)) + 0.5(0 + 0.5V^*(1)) = 0.25V^*(2) + 0.25V^*(1) = 2 + 0.25V^*(1)$

We see that staying at 1 is optimal.

$$V^*(1) = 4 + 4\gamma^1 + 4\gamma^2 + \cdots$$

$$V^*(1) = \frac{4}{1-\gamma} = 8$$

8	**8**	**24**	**72**

(d) Optimal policy

stay	**right**	**right**	**stay**

(a) $V_1(s)$ if an unwanted stay is rewarded where the agent is

S=1

Stay: $4 + 0.5V_0(1) = 4 + 0.5 \cdot 0 = 4$

Right: $0.5(0 + 0.5V_0(2)) + 0.5(4 + 0.5V_0(1)) = 0.5(0) + 0.5(4) = 2$

$$V_1(1) = max(4,2) = 4$$

S=2

Stay: $0 + 0.5V_0(2) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_0(1)\big) + 0.5\big(0 + 0.5V_0(2)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_0(3)\big) + 0.5\big(0 + 0.5V_0(2)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(2) = max(0,0,0) = 0$$

S=3

Stay: $0 + 0.5V_0(3) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_0(2)\big) + 0.5\big(0 + 0.5V_0(3)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_0(4)\big) + 0.5\big(0 + 0.5V_0(3)\big) = 0.5(0) + 0.5(0) = 0$

$$V_1(3) = max(0,0,0) = 0$$

S=4

Stay: $36 + 0.5V_0(4) = 36 + 0.5 \cdot 0 = 36$

Left: $0.5\big(0 + 0.5V_0(3)\big) + 0.5\big(36 + 0.5V_0(4)\big) = 0.5(0) + 0.5(36) = 18$

$$V_1(4) = max(36,18) = 36$$

4	**0**	**0**	**36**

(b) $V_2(s)$ if an unwanted stay is rewarded where the agent is

S=1

Stay: $4 + 0.5V_1(1) = 4 + 0.5 \cdot 4 = 6$

Right: $0.5\big(0 + 0.5V_1(2)\big) + 0.5\big(4 + 0.5V_1(1)\big) = 0.5(0) + 0.5(4 + 0.5 \cdot 4) = 3$

$$V_2(1) = max(6,3) = 6$$

S=2

Stay: $0 + 0.5V_1(2) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_1(1)\big) + 0.5\big(0 + 0.5V_1(2)\big) = 0.5(2) + 0.5(0) = 1$

Right: $0.5\big(0 + 0.5V_1(3)\big) + 0.5\big(0 + 0.5V_1(2)\big) = 0.5(0) + 0.5(0) = 0$

$$V_2(2) = max(0,1,0) = 1$$

S=3

Stay: $0 + 0.5V_1(3) = 0 + 0.5 \cdot 0 = 0$

Left: $0.5\big(0 + 0.5V_1(2)\big) + 0.5\big(0 + 0.5V_1(3)\big) = 0.5(0) + 0.5(0) = 0$

Right: $0.5\big(0 + 0.5V_1(4)\big) + 0.5\big(0 + 0.5V_1(3)\big) = 0.5(0.5 \cdot 36) + 0.5(0) = 9$

$$V_2(3) = max(0,0,9) = 9$$

S=4

Stay: $36 + 0.5V_1(4) = 36 + 0.5 \cdot 36 = 54$

Left: $0.5\big(0 + 0.5V_1(3)\big) + 0.5\big(36 + 0.5V_1(4)\big) = 0.5(0) + 0.5(36 + 0.5 \cdot 36) = 27$

$$V_2(4) = max(54,27) = 54$$

6	**1**	**9**	**54**

6 Discount Shaping

In MDPs, we usually select the discount factor as a hyperparameter then compute the value function. However, although much less common, finding an appropriate discount factor for a desired value could have some use as well (e.g. reverse engineering a real-life process). Consider a gridworld-like MDP with five states $s_1, ..., s_5$. The agent can either stay (a_S) or continue (a_C). You are also given the following for transition kernel $T(s, a, s')$ and reward function $R(s, a)$:

$$T(s_i, a_S, s_i) = 1 \quad \text{for } i \in \{1, 2, 3, 4\}$$
$$T(s_{i+1}, a_C, s_i) = 1 \quad \text{for } i \in \{1, 2, 3, 4\}$$
$$T(s_5, a, s_5) = 1 \quad \text{for all actions } a$$
$$R(s_i, a) = 0 \quad \text{for } i \in \{1, 2, 3, 5\} \text{ and for all actions } a$$
$$R(s_4, a_S) = 0, R(s_4, a_C) = 10$$

If the desired optimal value for state s_1 is $V^*(s_1) = 1$, what is the discount factor γ?

$$V^*(s) = \max_a \sum_{S'} T(s, a, s')[R(s, a) + \gamma V^*(s')]$$

State 5

State s_5 is a terminal state, and all rewards are 0. Thus:

$$V^*(s_5) = 0$$

State 4

Stay (a_S): Remain in s_4, receiving no reward.

Continue (a_C): Move to s_5 and receive a reward of 10.

$$Q(s_4, a_S) = R(s_4, a_S) + \gamma V^*(s_4) = 0 + \gamma V^*(s_4)$$

$$Q(s_4, a_C) = R(s_4, a_C) + \gamma V^*(s_5) = 10 + \gamma \cdot 0 = 10$$

The optimal action is a_C, since $Q(s_4, a_C) > Q(s_4, a_S)$. Thus:

$$V^*(s_4) = \max\big(Q(s_4, a_C), Q(s_4, a_S)\big) = 10$$

State 3

$$Q(s_3, a_S) = R(s_3, a_S) + \gamma V^*(s_3) = 0 + \gamma V^*(s_3)$$

$$Q(s_3, a_C) = R(s_3, a_C) + \gamma V^*(s_4) = 0 + \gamma \cdot 10 = 10\gamma$$

$$V^*(s_3) = \max\big(Q(s_3, a_C), Q(s_3, a_S)\big) = 10\gamma$$

State 2

$$Q(s_2, a_S) = R(s_2, a_S) + \gamma V^*(s_2) = 0 + \gamma V^*(s_2)$$

$$Q(s_2, a_C) = R(s_2, a_C) + \gamma V^*(s_3) = 0 + \gamma \cdot 10\gamma = 10\gamma^2$$

$$V^*(s_2) = \max\big(Q(s_2, a_C), Q(s_2, a_S)\big) = 10\gamma^2$$

State 1

$$Q(s_1, a_S) = R(s_1, a_S) + \gamma V^*(s_1) = 0 + \gamma V^*(s_1)$$

$$Q(s_1, a_C) = R(s_1, a_C) + \gamma V^*(s_2) = 0 + \gamma \cdot 10\gamma^2 = 10\gamma^3$$

$$V^*(s_1) = \max\big(Q(s_1, a_C), Q(s_1, a_S)\big) = 10\gamma^3$$

Solve for γ

$$V^*(s_1) = 10\gamma^3 = 1$$

$$\gamma = \left(\frac{1}{10}\right)^{\frac{1}{3}} \approx 0.464$$

7 Reinforcement Learning I

Imagine an unknown game which has only two states {A, B} and in each state the agent has two actions to choose from: {Up, Down}. Suppose a game agent chooses actions according to some policy π and generates the following sequence of actions and rewards in this unknown game:

t	s_t	a_t	s_{t+1}	r_t
0	A	Down	B	8
1	B	Down	B	-4
2	B	Up	B	0
3	B	Up	A	2
4	A	Up	A	-2

Assume a discount factor $\gamma = 0.5$ and a learning rate $\alpha = 0.5$.

(a) In model-based reinforcement learning, we first estimate the transition function $T(s, a, s')$ and the reward function $R(s, a, s')$. Fill in the following estimates of T and R, obtained from the experience above. Write "n/a" if not applicable or undefined.

$\hat{T}(A, Up, A) =$ ____ $\qquad$ $\hat{R}(A, Up, A) =$ ____

$\hat{T}(A, Up, B) =$ ____ $\qquad$ $\hat{R}(A, Up, B) =$ ____

$\hat{T}(B, Up, A) =$ ____ $\qquad$ $\hat{R}(B, Up, A) =$ ____

$\hat{T}(B, Up, B) =$ ____ $\qquad$ $\hat{R}(B, Up, B) =$ ____

$$\check{T}(s, a, s') = \frac{Number\ of\ transitions\ from\ (s, a)\ to\ s'}{Total\ number\ of\ times\ action\ a\ was\ taken\ in\ state\ s}$$

$\hat{R}(s, a, s')$ is by averaging all rewards observed when (s, a) to s'

(a)

$$\check{T}(A, Up, A) = \frac{1}{1} = 1, \check{T}(A, Up, B) = \frac{0}{1} = 0, \check{T}(B, Up, A) = \frac{1}{2}, \check{T}(B, Up, B) = \frac{1}{2}$$

$$\hat{R}(A, Up, A) = -2, \hat{R}(A, Up, B) = n/a, \hat{R}(B, Up, A) = 2, \hat{R}(B, Up, B) = 0$$

(b)

$$Q(s_t, a_t) = (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a'))$$

$$Q(A, Down) = (1-\alpha)Q(A, Down) + \alpha(r_0 + \gamma \max_{a'} Q(B, a'))$$

Initially $Q(A, Down) = 0, Q(B, Up) = 0, Q(B, Down) = 0$

$$Q(A, Down) = (1-0.5)0 + 0.5\left(8 + 0.5 \max_{a'} Q(B, a')\right) = 4$$

$$Q(B, Down) = (1-\alpha)Q(B, Down) + \alpha(r_1 + \gamma \max_{a'} Q(B, a'))$$

$$Q(B, Down) = (1-0.5)0 + 0.5\left(-4 + 0.5 \max_{a'} Q(B, a')\right) = -2$$

$$Q(B, Up) = (1-\alpha)Q(B, Up) + \alpha(r_2 + \gamma \max_{a'} Q(B, a'))$$

$$Q(B, Up) = 0, Q(B, Down) = -2$$

$$Q(B, Up) = (1-0.5)0 + 0.5(0 + 0.5 \max_{a'} Q(B, a')) = 0$$

$$Q(B, Up) = (1-\alpha)Q(B, Up) + \alpha(r_3 + \gamma \max_{a'} Q(A, a'))$$

$$Q(A, Down) = 4, Q(A, Up) = 0$$

$$Q(B, Up) = (1-0.5)0 + 0.5(2 + 0.5 \max_{a'} Q(A, a')) = 2$$

$$Q(A, Up) = (1-\alpha)Q(A, Up) + \alpha(r_4 + \gamma \max_{a'} Q(A, a'))$$

$$Q(A, Down) = 4, Q(A, Up) = 0$$

$$Q(A, Up) = (1-0.5)0 + 0.5(-2 + 0.5 \max_{a'} Q(A, a')) = 0$$

Finally, $Q(A, Down) = 4, Q(B, Up) = 2.$

8 Approximate Q-Learning

Suppose that one weekend, you decide to go to an amusement park with your friends. You start at the amusement park feeling well, receiving positive rewards from rides, with some rides offering greater rewards than others. However, each ride carries a risk of making you feel sick. If you continue to go on rides while feeling sick, there is a chance you may recover, but the rewards you earn from rides will likely be reduced and could even be negative.

You have no prior experience visiting amusement parks, so you are unsure about the exact rewards you will receive from each ride (whether feeling well or sick). Similarly, you do not know the probability of getting sick on any particular ride or recovering while sick. What you do know about the rides is summarized below:

Actions / Rides	**Type**	**Wait**	**Speed**
Big Dipper	Rollercoaster	Long	Fast
Wild Mouse	Rollercoaster	Short	Slow
Hair Raiser	Drop tower	Short	Fast
Moon Ranger	Pendulum	Short	Slow
Leave the Park	Leave	Short	Slow

Let's formulate this problem as a two state MDP: well and sick. The actions are the choice of what ride to take. 'Leave the Park' action terminates the run. Taking a ride can either transition to the same state with some probability or take you to the other state. We'll use a feature-based approximation to the Q-values, defined by the following four features and associated weights:

Features	**Initial Weights**
$f_0(\text{state, action}) = 1$ (this is a bias feature that is always 1)	$w_0 = 1$
$f_1(\text{state, action}) = \begin{cases} 1 & \text{if action type is Rollercoaster} \\ 0 & \text{otherwise} \end{cases}$	$w_1 = 2$
$f_2(\text{state, action}) = \begin{cases} 1 & \text{if action wait is Short} \\ 0 & \text{otherwise} \end{cases}$	$w_2 = 1$
$f_3(\text{state, action}) = \begin{cases} 1 & \text{if action speed is Fast} \\ 0 & \text{otherwise} \end{cases}$	$w_3 = 0.5$

(a) $Q(Well, BigDipper)$

$$Q(s,a) = w_i f_i(s,a) = w_0 f_0(s,a) + w_1 f_1(s,a) + w_2 f_2(s,a) + w_3 f_3(s,a)$$

$$Q(Well, BigDipper) = 1 + 2 + 0 + 0.5 = 3.5$$

(b) $(Well, BigDipper, Sick, -10.5)$

$$w_i = w_i + \alpha\left[\left(r + \gamma \max_{a'} Q(s', a')\right) - Q(s, a)\right] f_i(s, a)$$

Assume all initial Q-values are 0 for the new state-action pairs $Q(Sick, a') = 0$

$$r + \gamma \max_{a'} Q(s', a') - Q(s, a) = r + \gamma \max_{a'} Q(Sick, a') - Q(Well, BigDipper)$$
$$= -10.5 + 0.5 \cdot 0 - 3.5 = -14$$

$$w_i = w_i + 0.5[-14] f_i(Well, BigDipper)$$

$$w_0 = 1 + 0.5[-14]1 = -6$$

$$w_1 = 2 + 0.5[-14]1 = -5$$

$$w_2 = 1 + 0.5[-14]0 = 1$$

$$w_3 = 0.5 + 0.5[-14]1 = -6.5$$

(c) is $Q(Well, a) = Q(Sick, a)$

Even though the features f_i depend only on the action, the weights w_i are updated based on the state-dependent rewards and transitions, as the Q-values for the same action in the Well and Sick states will diverge because $Q(Well, a) \neq Q(Sick, a)$

(d) ϵ-greedy

$$Q(s, a) = w_i f_i(s, a) = w_0 f_0(s, a) + w_1 f_1(s, a) + w_2 f_2(s, a) + w_3 f_3(s, a)$$

An ϵ-greedy policy chooses the action with the highest Q-value most of the time with probability 1−ϵ, but occasionally with probability ϵ, it selects a random action.

$$Q(Well, BigDipper) = 1 + 2 + 0 + 0.5 = 3.5$$

$$Q(Well, WildMouse) = 1 + 2 + 1 + 0 = 4\ (highest)$$

$$Q(Well, HairRaiser) = 1 + 0 + 1 + 0.5 = 2.5$$

$$Q(Well, MoonRanger) = 1 + 0 + 1 + 0 = 2$$

$$Q(Well, LeaveThePark) = 1 + 0 + 1 + 0 = 2$$

WildMouse will be chosen with probability $1 - \frac{4\epsilon}{5}$.

Each other action (BigDipper, HairRaiser, MoonRanger, LeaveThePark) will be chosen with probability $\frac{\epsilon}{5}$.

9 Deep RL- Value-based Methods

(a) Tabular Q-learning vs DQN

Representation of Q-Values

Tabular Q-Learning: Uses a Q-table to store Q-values for each state-action pair. This approach is feasible only for environments with discrete and small state-action spaces.

DQN: Approximates the Q-function using a deep neural network, which enables it to handle high-dimensional and continuous state spaces. The neural network takes the state as input and outputs Q-values for all possible actions.

Experience Replay

Tabular Q-learning: Updates the Q-values immediately after observing a transition (state, action, reward, next state). This direct update can lead to instability and inefficient learning.

DQN: Utilizes an experience replay buffer to store past transitions (s, a, r, s') and samples mini batches of transitions randomly during training. This improves data efficiency and breaks the correlation between consecutive updates, stabilizing the learning process.

Target Network

Tabular Q-learning: Directly uses the current Q-values to compute the target for updates, which can lead to instability in function approximation.

DQN: Introduces a target network that is periodically updated to match the current Q-network. This decouples the target computation from the updates, further stabilizing training.

(b) Performance impact of three differences

Experience Replay Improves Performance

Deep neural networks require large and diverse datasets to learn effectively, and experience replay addresses this need by breaking the temporal correlation of data collected during reinforcement learning.

Breaking Temporal Correlation

Without experience replay, consecutive transitions are highly correlated since they occur sequentially in the environment. Training a neural network on such correlated data can lead to poor convergence and instability. Experience replay stores transitions in a buffer and samples them randomly, ensuring that updates are based on uncorrelated data. This leads to more stable gradient updates and improves the convergence of the Q-network.

Efficient Use of Data

Each transition is reused multiple times during training, which increases the sample efficiency of the algorithm. This is particularly important when data collection is expensive or when interacting with the environment is slow.

Stabilizing Learning

By sampling from a diverse set of experiences (different states, actions, and rewards), the Q-network is exposed to a more representative distribution of the state-action space. This reduces the risk of overfitting to recent experiences and helps the network generalize better.

(c) Low Frequency Update

Stale Target Values

The target network provides the target Q-values for the Bellman update.

$$y_t = r + \gamma \max_{a'} Q_{target}(s', a'; \theta^-)$$

With very infrequent updates every 2^{64} steps, the target network becomes effectively frozen for almost the entirety of training. This means the Q-values used for targets are based on an outdated and inaccurate representation of the Q-function. The Bellman targets y_t will become less reflective of the true expected return. This will introduce significant errors into the Q-network's updates, leading to poor convergence or divergence from the optimal policy.

Slower Training Progress

The target network acts as a stabilizing mechanism, ensuring that the rapidly changing current Q-function does not overly influence updates to the Q-network. With infrequent updates, the target network fails to capture improvements in the Q-function over time. As a result, the training becomes inefficient, and the agent struggles to improve its policy since it is learning based on outdated target values. The Q-network will take much longer to converge to approximate the Q-function accurately. The agent may exhibit erratic behavior or fail to learn a meaningful policy.

Reduced Policy Performance

The agent's performance depends on the Q-network accurately approximating the optimal Q-function. If the target network does not update often enough, the Q-network may underestimate or overestimate Q-values, leading to suboptimal action selection. Also, the Q-network may fail to adapt to dynamic environmental changes if the targets remain static. The agent's policy will be suboptimal and may perform poorly or fail to solve the task.

10 Deep RL- Policy Gradient and Actor-Critic Methods

(a) Causality in Policy Gradient to Reduce Variance

In reinforcement learning, policy gradient methods aim to optimize the policy by computing gradients of the expected return concerning the policy parameters. However, high variance in these gradient estimates can hinder learning. The concept of causality is employed in two significant ways to reduce variance in policy gradient methods:

Reward-to-Go

The idea of causality ensures that actions at time t are only credited with rewards occurring after t ($t < t'$), as actions cannot affect rewards from the past.

$$G_t = \sum_{t'=t}^{T} r_{t'}$$

Here, G_t is the reward-to-go starting from time t. In this approach, instead of using the total episodic reward for all actions, the reward-to-go is used, focusing the credit assignment. This reduces variance because the contribution of irrelevant (earlier) rewards to the gradient is eliminated. Reward-to-Go trims the contribution of irrelevant rewards to gradient calculations, reducing variance caused by long-term returns that actions do not influence.

Baselines

A baseline is a scalar function $b(s_t)$ subtracted from the reward before calculating the policy gradient. This baseline does not introduce bias but reduces variance by centering the rewards. The modified policy gradient with a baseline is:

$$\nabla_\theta J(\theta) = E[\nabla_\theta log\pi_\theta(a_t \mid s_t) \cdot (G_t - b(s_t))]$$

A common choice for the baseline is the value function, which estimates the expected future reward from the current state $b(s_t) = V^\pi(s_t)$. By subtracting $V^\pi(s_t)$, the variance is reduced because the difference $G_t - V^\pi(s_t)$ called the advantage has lower variability than G_t alone. Baselines further reduce the variability of the gradient by centering the returns, making the updates more consistent.

(b) Policy Gradient with a Baseline

The policy gradient for the actor is:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a \mid s) \cdot A^\pi(s,a))]$$

where $A^\pi(s,a)$ is the advantage function, defined as:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

$Q^\pi(s,a)$ is the expected return when taking action a in state s and following policy π. $V^\pi(s)$ is the expected return when starting in state s and following policy π. The advantage function measures how much better or worse an action a is compared to the average performance in state s.

Role of the Value Function

The critic estimates $V_\phi(s)$, an approximation of $V^\pi(s)$, and this serves as the baseline in the advantage function. By substituting $A^\pi(s,a)$ with $Q^\pi(s,a) - V_\phi(s)$, the policy gradient becomes:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a \mid s) \cdot (r + \gamma V_\phi(s') - V_\phi(s)))]$$

$r + \gamma V_\phi(s')$ is a one-step estimate of $Q^\pi(s,a)$. $V_\phi(s)$ is the baseline provided by the critic.

How the Baseline Reduces Variance

It creates a centering the Advantage. Subtracting $V_\phi(s)$ centers the reward signal, ensuring that actions are judged relative to the average return expected in state s. This reduces the variability of gradient estimates by eliminating irrelevant shifts in the reward scale.

It gives an unbiased gradient. The baseline does not introduce bias because $E[A^\pi(s,a)] = E[Q^\pi(s,a)] - E[V^\pi(s)]$, and $E[Q^\pi(s,a)] = E[V^\pi(s)]$ under the policy π. Therefore, the expectation of the gradient remains unchanged.

It provides efficient learning. Lower variance enables the actor to update its policy parameters more consistently, accelerating convergence to an optimal policy.

Integration in Actor-Critic Algorithms

Actor updates θ to improve the policy using the gradient computed with the advantage function $A^\pi(s,a)$. Critic updates ϕ to minimize the error in the value function approximation by minimizing the Temporal Difference (TD) error $L(\phi) = E[\left(r + \gamma V_\phi(s') - V_\phi(s)\right)^2]$. By estimating $V^\pi(s)$, the critic provides a dynamic and state-specific baseline, enhancing the stability and efficiency of the actor's policy updates.

(c) Compare the Actor-critic and Policy Gradient

Method	Bias Level	Variance Level
Actor-Critic	High	Low
Policy Gradient	Low	High

Actor-Critic

The bias level is high. The critic in actor-critic methods approximates the value function $V^{\pi}(s)$ using function approximation like neural networks. This approximation introduces bias into the gradient estimates, particularly when the value function is not well-learned or due to inaccuracies in the critic's estimation.

The variance level is low. By incorporating the critic as a baseline, the actor-critic method reduces the variance of the policy gradient. The centered advantage term $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ ensures that only relative performance influences the gradient, stabilizing the updates.

Policy Gradient

The bias level is low. Policy gradient methods rely on Monte Carlo estimates of the returns without approximations, so they are unbiased. However, this comes at the cost of using high-variance sample estimates.

The variance level is high. Policy gradient methods do not use a baseline or critic to reduce variance. As a result, the gradient estimates can fluctuate significantly due to the reliance on raw returns, leading to unstable and noisy updates.