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 for all nodes X.

### (i)

It is admissible as .

For

### (ii)

It is admissible as This makes .

### (iii)

It is NOT admissible as . Heuristic may overestimate the true cost. Thus, the safest statement we can make is .

### (iv)

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

Base Case :

It is admissible as .

Recursive Case: :

### (v)

Case :

It is admissible as .

Case: :

In this case where

For any neighbor , we know . Thus, we get:

as

Taking the minimum over all :

which makes admissible for all cases. This makes .

# 2 Bayes Networks - Variable Elimination

A diagram of a network

Description automatically generated

## 1 Z=+z

Initial CPTs are

When evidence is observed, the factor is updated. This results in:

## 2 X

## 3 T

## 4 U

## 5 V

## 6 W

## 7

Normalize to get

## 8 Optimality

Given Order

Eliminating X

The factors involving X are and .

After eliminating X, a new factor is generated, which depends only on T.

Eliminating T

The factors involving T are , , , and .

After eliminating T, a large factor is generated, involving U, V, and W.

Eliminating U

The factor depends on U, so eliminating U produces a new factor .

Eliminating V

The factors involving V are and .

After eliminating V, a factor is generated.

Eliminating W

The factor depends on W, so eliminating W produces a final factor .

Observation

Eliminating T early introduces a large intermediate factor , 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

A diagram of a graph

Description automatically generated

## (a)

### Elapse Time

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

For the time being there is no that can make dependent.   
This makes .

### Observation

As from the structure (BN),

is normalization, we can remove if we want brevity. Exact calculation of normalization is:

Don’t forget that we know from elapse time.

## (b)

### Elapse Time

From the structure (BN) we can deduce that .

From the structure (BN) we can deduce that .

### Observation

As from the structure (BN),

is normalization, we can remove if we want brevity. Exact calculation of normalization is:

Don’t forget that we know from elapse time.

# 4 HMM: Viterbi Algorithm

A diagram of a graph

Description automatically generated

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

The key points:

* is the maximum probability of the most likely state sequence up to time t ending in state ​.
* The algorithm focuses on maximizing the joint probability of the sequence of states and observations.

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

The next state depends on the previous state and the current action:

The observation depends on the current state: .

The actions themselves can be either exogenously specified or generated from some policy ( or state‐independent).

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

**Viterbi** in that case would search

There is no factor in this objective if the actions are known inputs (we do not need to explain the actions, only the hidden states). Also, we have    as depends on both ​ and ​.

Viterbi update in this DBN would be:

## 1

The Viterbi algorithm does **not** maximize . This is the marginal probability of the sequence of states ​, independent of the evidence ​.

integrates over all possible evidence sequences, while the Viterbi algorithm conditions on the observed evidence ​.

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

## 2

The Viterbi algorithm does **not** maximize .

The actions ​ are independent of the hidden states ​ and evidence . In this HMM structure, the actions ​ are state-independent, meaning . Thus, the probability of an action sequence 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

The Viterbi algorithm does **not** maximize .

involves summing over all possible hidden state sequences ​, 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 , rather than marginalizing over all state sequences.

Computing 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

The Viterbi algorithm does **not** maximize .

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

If is observed or given, then

But is fixed in the data or experiment. Thus, there is no to do as those actions are known. Consequently, the Viterbi algorithm does not try to maximize as a function of both X and U. Rather, U is given or observed, so Viterbi only needs to find the most likely . Because there is no reason to maximize over the known actions .

## 5

The Viterbi algorithm does **not** maximize . Same reasoning as the question 4 can be applied.

The expression expands as:

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.

is the state sequence probabilities conditioned on the actions.

is the probability of the evidence given the state sequence.

The Viterbi algorithm does not explicitly maximize , as the actions are typically determined by a policy external to the algorithm. Therefore, the inclusion of in means this expression is not directly maximized.

## 6

The Viterbi algorithm does **not** maximize

None of the provided products match what the Viterbi algorithm for this DBN is really doing, because the given product omits the dependence on ​ in and unnecessarily includes .

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

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

## 7

The Viterbi algorithm does **not** maximize

If the actions ​ 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 , we just treat ​ as known inputs to the transition. In other words, once you condition on the fact that the action ​ happened exogenously, you no longer multiply by instead you use the transition .

If the policy is truly state-independent, then would collapse to a constant , which would factor out of the Viterbi maximization anyway.

# 5 1D Gridworld MDP

A screenshot of a math test

Description automatically generated

is the probability of transitioning to s’ from s using action a.

is the reward received from transitioning from s to s’ with action a.

is the discount factor and .

**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) if an unwanted stay is rewarded 0.

### S=1

Stay:

Right:

### S=2

Stay:

Left:

Right:

### S=3

Stay:

Left:

Right:

### S=4

Stay:

Left:

|  |  |  |  |
| --- | --- | --- | --- |
| **4** | **0** | **0** | **36** |

## (b) if an unwanted stay is rewarded 0

### S=1

Stay:

Right:

### S=2

Stay:

Left:

Right:

### S=3

Stay:

Left:

Right:

### S=4

Stay:

Left:

|  |  |  |  |
| --- | --- | --- | --- |
| **6** | **1** | **9** | **54** |

## (c) if an unwanted stay is rewarded 0

### S=4

Optimal action is staying.

### S=3

Stay:

Left:

Right:

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

### S=2

Stay:

Left:

Right:

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 , which is less than 6, making staying at 1 suboptimal.

### S=1

Stay:

Right:

We see that staying at 1 is optimal.

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **8** | **24** | **72** |

## (d) Optimal policy

|  |  |  |  |
| --- | --- | --- | --- |
| **stay** | **right** | **right** | **stay** |

## (a) if an unwanted stay is rewarded where the agent is

### S=1

Stay:

Right:

### S=2

Stay:

Left:

Right:

## S=3

Stay:

Left:

Right:

### S=4

Stay:

Left:

|  |  |  |  |
| --- | --- | --- | --- |
| **4** | **0** | **0** | **36** |

## (b)

### S=1

Stay:

Right:

### S=2

Stay:

Left:

Right:

### S=3

Stay:

Left:

Right:

### S=4

Stay:

Left:

|  |  |  |  |
| --- | --- | --- | --- |
| **6** | **1** | **9** | **54** |

# 6 Discount Shaping

A white text with black text

Description automatically generated

## State 5

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

## State 4

Stay (​): Remain in ​, receiving no reward.

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

The optimal action is ​, since . Thus:

## State 3

## State 2

## State 1

## Solve for

# 7 Reinforcement Learning I

A white sheet with black text

Description automatically generated with medium confidence

is by averaging all rewards observed when (s, a) to s’

## (a)

## (b)

Initially

Finally, .

# 8 Approximate Q-Learning

A close-up of a document

Description automatically generated

## (a)

## (b)

Assume all initial Q-values are 0 for the new state-action pairs

## (c) is

Even though the features depend only on the action, the weights ​ 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

## (d) ϵ-greedy

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.

WildMouse will be chosen with probability .

Each other action (BigDipper, HairRaiser, MoonRanger, LeaveThePark) will be chosen with probability ​.

# 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 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.

With very infrequent updates every 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 ​ 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 (), as actions cannot affect rewards from the past.

Here, ​ 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 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:

A common choice for the baseline is the value function, which estimates the expected future reward from the current state . By subtracting , the variance is reduced because the difference called the advantage has lower variability than ​ 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:

where is the advantage function, defined as:

is the expected return when taking action a in state s and following policy π. 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 , an approximation of , and this serves as the baseline in the advantage function. By substituting with , the policy gradient becomes:

is a one-step estimate of .  is the baseline provided by the critic.

### How the Baseline Reduces Variance

It creates a centering the Advantage.Subtracting 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 , and 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 . Critic updates ϕ to minimize the error in the value function approximation by minimizing the Temporal Difference (TD) error . By estimating , 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 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 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.