

Value Iteration in MDP using Bellman Equation

AIM

To implement the Bellman Equation using Value Iteration in an MDP, compute optimal state values, and explore the impact of the discount factor (γ) on balancing immediate and future rewards.

EXPERIMENTAL SETUP

A general Dataset, is required.

LIBRARIES REQUIRED

No External Libraries Required.

STEPS

1. Loading necessary libraries
2. Loading the necessary dataset: MDP Environment Definition
3. Perform Bellman Equation Algorithm for Value Iteration
4. Implement Hyperparameter Tuning
5. Output Analysis

CODE SNIPPET

```
# Define a simple MDP environment
# States, actions, rewards, transitions

# List of states
states <- c("A", "B", "C", "D")

# Actions available in each state
actions <- list(
  A = c("go_B", "go_C"),
  B = c("go_A", "go_D"),
```

```

C = c("go_A", "go_D"),
D = c("go_B", "go_C")
)

# Transition probabilities for each state-action pair (deterministic here)
transitions <- list(
  A = list(go_B = "B", go_C = "C"),
  B = list(go_A = "A", go_D = "D"),
  C = list(go_A = "A", go_D = "D"),
  D = list(go_B = "B", go_C = "C")
)

# Reward function for each state-action pair
rewards <- list(
  A = list(go_B = 1, go_C = 5),
  B = list(go_A = -1, go_D = 3),
  C = list(go_A = 0, go_D = 4),
  D = list(go_B = 2, go_C = 6)
)

# Bellman equation with value iteration
value_iteration <- function(states, actions, transitions, rewards, gamma = 0.9, iterations = 100) {
  # Initialize value function to zero for each state
  values <- setNames(rep(0, length(states)), states)

  # Perform value iteration
  for (i in seq_len(iterations)) {
    new_values <- values # Copy current values

    for (s in states) {
      action_values <- sapply(actions[[s]], function(a) {
        next_state <- transitions[[s]][[a]]
        reward <- rewards[[s]][[a]]
        reward + gamma * values[[next_state]] # Bellman update
      })

      # Assign the maximum value to the current state
      new_values[[s]] <- max(action_values)
    }

    values <- new_values # Update value function
  }

  return(values)
}

# Run value iteration for gamma = 0.9
optimal_values_0.9 <- value_iteration(states, actions, transitions, rewards, gamma = 0.9, iterations = 100)
print("Optimal values for gamma = 0.9:")

```

```
## [1] "Optimal values for gamma = 0.9:"
```

```
optimal_values_0.9
```

```
##           A           B           C           D
## 49.52497 48.47237 49.47237 50.52497
```

```
# Hyperparameter tuning for the discount factor (gamma)
gamma_values <- seq(0.7, 0.99, by = 0.05)
optimal_values <- list()

for (gamma in gamma_values) {
  optimal_values[[paste0("gamma_", gamma)]] <- value_iteration(
    states,
    actions,
    transitions,
    rewards,
    gamma = gamma,
    iterations = 100
  )
}

print("Optimal values for different gamma:")
```

```
## [1] "Optimal values for different gamma:"
```

```
optimal_values
```

```
## $gamma_0.7
##           A           B           C           D
## 16.25490 15.07843 16.07843 17.25490
##
## $gamma_0.75
##           A           B           C           D
## 19.57143 18.42857 19.42857 20.57143
##
## $gamma_0.8
##           A           B           C           D
## 24.55556 23.44444 24.44444 25.55556
##
## $gamma_0.85
##           A           B           C           D
## 32.87387 31.79279 32.79279 33.87387
##
## $gamma_0.9
##           A           B           C           D
## 49.52497 48.47237 49.47237 50.52497
##
## $gamma_0.95
##           A           B           C           D
## 98.91773 97.89816 98.89816 99.91773
```

CONCLUSION

Value Iteration effectively solved the MDP, with gamma tuning revealing trade-offs between short- and long-term gains. The results deepened understanding of reinforcement learning fundamentals and value convergence behavior in decision-making systems.