Exam 2

Markov Decision Problems (MDP)

Key topics include the basics of Markov Decision Problems (MDP), value iteration, policy iteration, the Bellman equation, discount factors, and evaluation of policies in gridworld environments. These concepts are crucial for understanding decision-making under uncertainty in AI.

MDPs provide a framework for modeling decision-making where outcomes are partly random and partly under the control of a decision-maker.

Markov Decision Problems (MDP)

MDPs consist of states, actions, transition functions, and rewards, used to model decision-making under uncertainty.

- **States**: Configurations of the environment.
- **Actions**: Possible moves or decisions available to the agent.
- **Transition Function**: Probability of reaching a new state given a current state and action.
- **Rewards**: Immediate gain or loss resulting from an action in a state.

Value Iteration

Value iteration is an algorithm for computing the optimal policy and value function in an MDP by iteratively updating the value of each state using the Bellman equation.

Value Iteration

Value iteration iteratively computes the optimal value function, converging to the true values for each state.

- **Bellman Equation**: Updates the value of a state based on the expected utility of possible actions.

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

  - $V(s)$: Value of state s. - $T(s,a,s')$: Transition probability from state s to s' using action a. - $R(s,a,s')$: Reward received after transitioning. - γ: Discount factor.

- **Convergence**: Guaranteed for MDPs with a finite number of states and a discount factor $0 < \gamma < 1$.
- **Optimal Policy**: Derived from the value function by selecting actions that maximize expected rewards.

Policy Iteration

Policy iteration alternates between policy evaluation and policy improvement to find the optimal policy.

Policy Iteration

Policy iteration involves evaluating a policy to determine its value function and then improving the policy based on the value function.

- **Policy Evaluation**: Computes the value function for a given policy using the Bellman equation.
- **Policy Improvement**: Updates the policy by choosing actions that maximize the expected value.
- **Convergence**: Guaranteed for MDPs with a finite number of states and actions.

Bellman Equation

The Bellman equation provides a recursive definition of the value function, expressing the value of a state in terms of the expected rewards and the values of successor states.

Bellman Equation

The Bellman equation defines the value of a state based on the maximum expected utility of subsequent states.

- **Equation**:
$$V(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s')\right]$$
- **Explanation**: - The value of state s is the maximum expected return achievable by any action a. - The equation considers all possible successor states s', weighted by the probability $T(s,a,s')$. - Rewards $R(s,a,s')$ and discounted future values $\gamma V(s')$ are summed to compute expected utility.

Discount Factor

The discount factor in an MDP determines the present value of future rewards, balancing immediate and long-term benefits.

Discount Factor

The discount factor γ influences the agent's preference for immediate versus future rewards.

- **Value Range**: $0 < \gamma < 1$ for convergence.
- **Impact on Policy**: Higher values favor long-term rewards, while lower values favor immediate rewards.
- **Policy Sensitivity**: Changes in γ can lead to different optimal policies.

Evaluation in Gridworld

Gridworld is a simple environment used to illustrate MDP concepts, where an agent moves in a grid to maximize rewards.

Evaluation in Gridworld

In gridworld, agents evaluate policies based on the expected rewards for different actions in grid states.

- **Actions**: Typically include moving in cardinal directions or exiting the grid.
- **Reward Structure**: Rewards can vary based on the grid location and action.
- **Policy Evaluation**: Involves calculating the expected rewards for different policies.

Key Concepts

Key Concepts

This section summarizes the key concepts related to Markov Decision Problems, emphasizing their definitions, properties, and applications in AI.

- **Markov Decision Problems (MDP)**
  - **States**: Environment configurations.
  - **Actions**: Possible decisions.
  - **Transition Function**: State change probabilities.
  - **Rewards**: Gains or losses from actions.
- **Value Iteration**
  - **Bellman Equation**: State value updates based on maximum expected utility.
  - **Convergence**: Guaranteed with $0 < \gamma < 1$.
  - **Optimal Policy**: Maximizes expected rewards.
- **Policy Iteration**
  - **Policy Evaluation**: Computes value functions.

  - **Policy Improvement**: Updates policies for maximum value.
  - **Convergence**: Guaranteed for finite states and actions.
- **Bellman Equation**
  - **Equation**: Recursive definition of state value.
  - **Explanation**: Value based on expected returns of actions and successor states.
- **Discount Factor**
  - **Value Range**: $0 < \gamma < 1$.
  - **Impact on Policy**: Balances short and long-term rewards.
  - **Policy Sensitivity**: Affects optimal policy decisions.
- **Evaluation in Gridworld**
  - **Actions**: Movement and exits.
  - **Reward Structure**: Varies by grid location.
  - **Policy Evaluation**: Calculates expected rewards.

Reinforcement Learning

Key topics include the Markov Decision Process (MDP), value iteration, policy iteration, and Q-learning.

Markov Decision Process (MDP)

MDP is a framework for modeling decision-making situations where outcomes are partly random and partly under the control of a decision maker.

MDP in Reinforcement Learning

An MDP is defined by:

- **States (S)**: The set of all possible states.
- **Actions (A)**: The set of all possible actions.
- **Transition Function (T)**: Probability of moving from one state to another given an action.
- **Reward Function (R)**: Immediate reward received after transitioning from one state to another.
- **Policy (π)**: A strategy that specifies the action to take in each state.

Value Iteration

Value iteration is a method of computing the optimal policy by iteratively improving the value function.

Value Iteration in Reinforcement Learning

Value iteration involves:

- **Value Function (V)**: Estimates the expected return from each state.
- **Bellman Equation**: $V(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')]$
- **Convergence**: Iteratively updating $V(s)$ until it converges to the optimal value function.

Policy Iteration

Policy iteration alternates between evaluating the current policy and improving it.

Policy Iteration in Reinforcement Learning

Policy iteration involves:

- **Policy Evaluation**: Computing the value function for the current policy.

- **Policy Improvement**: Updating the policy by choosing actions that maximize the value function.
- **Convergence**: Repeating evaluation and improvement until the policy converges to the optimal policy.

Q-Learning

Q-learning is a model-free reinforcement learning algorithm that seeks to learn the value of the optimal policy.

Q-Learning in Reinforcement Learning

Q-learning involves:

- **Q-Function (Q)**: Estimates the expected return of taking an action in a given state.
- **Update Rule**: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- **Exploration vs. Exploitation**: Balancing between exploring new actions and exploiting known actions that yield high rewards.

Approximate Reinforcement Learning

Key topics include feature-based representations, function approximation, and policy gradient methods.

Feature-Based Representations

Feature-based representations use features to approximate the value function or policy.

Feature-Based Representations in Approximate RL

Using features involves:

- **Features (f)**: Relevant characteristics of the state or state-action pair.
- **Linear Combination**: Approximating Q-values using a linear combination of features, $Q(s,a) \approx \mathbf{w}^T\mathbf{f}(s,a)$.
- **Weight Vector (w)**: Parameters to be learned that determine the contribution of each feature.

Function Approximation

Function approximation is used to generalize the value function or policy from limited data.

Function Approximation in Approximate RL

Function approximation techniques include:

- **Linear Function Approximation**: Approximating functions using a linear combination of features.
- **Non-linear Function Approximation**: Using methods like neural networks to approximate functions.
- **Gradient Descent**: Updating weights using the gradient of the loss function.

Policy Gradient Methods

Policy gradient methods directly parameterize the policy and optimize it using gradient ascent.

Policy Gradient Methods in Approximate RL

Policy gradient methods involve:

- **Parameterization**: Representing the policy using a parameterized function $\pi_\theta(a|s)$.
- **Gradient Ascent**: Updating the policy parameters using the gradient of the expected return, $\nabla_\theta J(\theta)$.
- **REINFORCE Algorithm**: A Monte Carlo method to estimate policy gradients and update the parameters.

Key Concepts

Key Concepts in Reinforcement Learning and Approximate RL

This section covers the core principles related to reinforcement learning and approximate reinforcement learning.

Reinforcement Learning:

- **MDP**: Framework for decision-making under uncertainty.
- **Value Iteration**: Iterative method to compute the optimal value function.
- **Policy Iteration**: Alternates between policy evaluation and improvement.
- **Q-Learning**: Model-free algorithm to learn the optimal Q-function.

Approximate Reinforcement Learning:

- **Feature-Based Representations**: Using features to approximate value functions or policies.
- **Function Approximation**: Generalizing value functions or policies from limited data.
- **Policy Gradient Methods**: Directly optimizing policies using gradient ascent.

Approximate Reinforcement Learning

Key topics include feature-based representation for Q-learning, action selection using approximate Q-functions, weight updates in linear function approximation, and handling stochastic rewards. These concepts are critical for implementing and understanding reinforcement learning with function approximation techniques.

Feature-Based Representation

In feature-based representation, the state-action space is represented using a set of features, which simplifies the learning process in environments with large or continuous state spaces.

Feature-Based Representation

Feature-based representation uses a vector of features to represent the state-action space, enabling efficient learning in large or continuous environments.

- **Features**: Functions that map state-action pairs to numerical values, representing different aspects of the state.
- **Linear Function Approximation**: The Q-value $Q(s, a)$ is approximated as a linear combination of features.
- **Weight Vector**: A set of weights w associated with the features, adjusted during learning to approximate the true Q-values.

Action Selection Using Approximate Q-Functions

Agents select actions based on the estimated Q-values from the feature-based representation, aiming to maximize expected rewards.

Action Selection Using Approximate Q-Functions

Actions are selected based on approximate Q-values computed from feature-based representations, guiding agents towards optimal behavior.

- **Q-Value Calculation**: $Q(s, a) = \sum_i w_i f_i(s, a)$, where f_i are features and w_i are weights.
- **Policy**: Selects the action a that maximizes $Q(s, a)$.
- **Exploration vs. Exploitation**: Balances exploring new actions and exploiting known good actions.

Weight Updates in Linear Function Approximation

Weight updates adjust the feature weights to reduce the difference between predicted and actual rewards, refining the policy.

Weight Updates in Linear Function Approximation

Weights in the feature-based representation are updated to improve the accuracy of Q-value predictions.

- **Update Rule**: $w_i \leftarrow w_i + \alpha\,(r + \gamma \max_{a'} Q(s', a') - Q(s, a))\, f_i(s, a)$
- **Learning Rate (α)**: Determines the size of weight updates.
- **Temporal Difference (TD) Error**: The difference between predicted and actual reward, guiding the weight adjustment.

Handling Stochastic Rewards

Stochastic rewards introduce variability in the rewards received, requiring strategies that can handle such uncertainties.

Handling Stochastic Rewards

Strategies for handling stochastic rewards are crucial for making robust decisions in environments with inherent uncertainty.

- **Expected Value**: Q-values represent expected returns, taking into account all possible outcomes.
- **Reward Averaging**: Using averages of observed rewards to estimate Q-values.
- **Robustness**: Ensuring that policies can handle fluctuations in rewards and still perform well.

Key Concepts

Key Concepts

This section summarizes the key concepts related to approximate reinforcement learning, emphasizing their definitions, properties, and applications in AI.

- **Feature-Based Representation**
  - **Features**: Numerical representations of state-action pairs.
  - **Linear Function Approximation**: Q-values approximated using features and weights.
  - **Weight Vector**: Adjusted during learning to match true Q-values.
- **Action Selection Using Approximate Q-Functions**
  - **Q-Value Calculation**: Summation of weighted features.
  - **Policy**: Chooses actions that maximize Q-values.
  - **Exploration vs. Exploitation**: Balances exploring new actions and exploiting known good actions.
- **Weight Updates in Linear Function Approximation**
  - **Update Rule**: Adjusts weights based on TD error.
  - **Learning Rate (α)**: Controls the magnitude of updates.
  - **TD Error**: Guides weight adjustments to improve predictions.
- **Handling Stochastic Rewards**
  - **Expected Value**: Accounts for all possible reward outcomes.
  - **Reward Averaging**: Averages observed rewards for Q-value estimates.
  - **Robustness**: Maintains policy performance despite reward variability.

Approximate Reinforcement Learning And Probability

Key topics include feature-based representation for Q-learning, action selection with feature weights, probability distributions, and conditional independence in probability theory. These concepts are fundamental for understanding decision-making in uncertain environments and the role of probability in reinforcement learning.

Feature-Based Representation in Q-Learning

Feature-based representation simplifies the representation of Q-values by using a set of features, especially in environments with large or continuous state spaces.

Feature-Based Representation in Q-Learning

Features represent state-action pairs using a vector of attributes, allowing efficient learning in complex environments.

- **Features**: Numerical values representing various aspects of the state or state-action pairs.
- **Linear Approximation**: The Q-value $Q(s,a)$ is approximated by $Q(s,a) = \sum_i w_i f_i(s,a)$.
- **Weight Vector** w: Adjusted during learning to approximate the true Q-values accurately.

Action Selection with Feature Weights

Agents use the Q-values derived from the feature weights to select actions, typically choosing the action that maximizes the Q-value.

Action Selection with Feature Weights

Actions are chosen based on the Q-values calculated using feature weights, guiding the agent towards optimal decisions.

- **Q-Value Calculation**: $Q(s,a) = \sum_i w_i f_i(s,a)$.
- **Policy**: Selects the action a that maximizes $Q(s,a)$.
- **Exploration vs. Exploitation**: Balances exploring new actions and exploiting known rewarding actions.

Probability Distributions

Understanding and utilizing probability distributions is essential for modeling uncertainty and making decisions under uncertainty.

Probability Distributions

Probability distributions describe the likelihood of different outcomes, essential for decision-making under uncertainty.

- **Discrete Distributions**: Probability assigned to discrete outcomes or events.
- **Conditional Probability**: The probability of an event given the occurrence of another event, denoted $P(A|B)$.
- **Joint Probability**: The probability of two events occurring together, denoted $P(A,B)$.

Conditional Independence

Conditional independence is a key concept in probability theory, indicating that two events are independent given the occurrence of a third event.

Conditional Independence

Conditional independence simplifies the representation and computation of probabilities in complex models.

- **Definition**: Events A and B are conditionally independent given C if $P(A,B|C) = P(A|C)P(B|C)$.

- **Notation**: Denoted as $A \perp B|C$.
- **Applications**: Used in Bayesian networks and other probabilistic models to simplify calculations.

Key Concepts

Key Concepts

This section summarizes the key concepts related to approximate reinforcement learning and probability, emphasizing their definitions, properties, and applications in AI.

- **Feature-Based Representation in Q-Learning**
  - **Features**: Attributes representing state-action pairs.
  - **Linear Approximation**: Q-values approximated using features and weights.
  - **Weight Vector**: Adjusted during learning to approximate true Q-values.
- **Action Selection with Feature Weights**
  - **Q-Value Calculation**: Sum of weighted features.
  - **Policy**: Chooses actions that maximize Q-values.
  - **Exploration vs. Exploitation**: Balances exploration and exploitation.
- **Probability Distributions**
  - **Discrete Distributions**: Likelihood of discrete outcomes.
  - **Conditional Probability**: Probability of an event given another.
  - **Joint Probability**: Probability of events occurring together.
- **Conditional Independence**
  - **Definition**: Independence of events given another event.
  - **Notation**: $A \perp B|C$.
  - **Applications**: Simplifies probabilistic calculations.

Bayes Network

Key topics include the concepts of marginal and conditional probabilities, the chain rule for Bayesian networks, independence and conditional independence, and the use of graphical models to represent probabilistic dependencies. These concepts are crucial for understanding and utilizing Bayes Networks in probabilistic reasoning.

Marginal and Conditional Probabilities

Marginal and conditional probabilities are fundamental components of probabilistic reasoning, providing the likelihood of events in different contexts.

Marginal and Conditional Probabilities

Marginal and conditional probabilities quantify the likelihood of events and their interdependencies.

- **Marginal Probability**: The probability of an event occurring, irrespective of other variables.
- **Conditional Probability**: The probability of an event given that another event has occurred, denoted $P(A|B)$.
- **Example**: $P(X_2 = 0) = 0.420$, $P(X_0 = 0, X_1 = 1) = 0.240$.

Chain Rule for Bayesian Networks

The chain rule is used to compute joint distributions from conditional probabilities specified by a Bayesian network's structure.

Chain Rule for Bayesian Networks

The chain rule allows the computation of joint probability distributions from conditional probabilities in a Bayesian network.

- **Equation**: $P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i|\text{Parents}(X_i))$
- **Application**: Used to compute joint distributions like $P(X = 0, Y = 0) = P(X)P(Y|X)$.
- **Example Calculation**: $P(X = 0, Y = 0, Z = 0) = P(X)P(Y|X)P(Z|Y)$

Independence and Conditional Independence

Independence and conditional independence are key concepts in Bayesian networks, defining how variables are related or independent.

Independence and Conditional Independence

These concepts describe whether variables in a Bayesian network are independent or conditionally independent given other variables.

- **Independence**: Two variables X and Y are independent if $P(X, Y) = P(X)P(Y)$.
- **Conditional Independence**: X and Y are conditionally independent given Z if $P(X, Y|Z) = P(X|Z)P(Y|Z)$.
- **Example**: X is independent of Y, but X is not independent of Y given Z.

Graphical Models and Probabilistic Dependencies

Graphical models visually represent the dependencies among variables in a Bayesian network, helping to understand and compute joint distributions.

Graphical Models and Probabilistic Dependencies

Graphical models illustrate the conditional dependencies and independencies among variables in a Bayesian network.

- **Nodes and Edges**: Nodes represent random variables, and edges indicate direct dependencies.
- **Directed Acyclic Graph (DAG)**: Represents the structure of a Bayesian network.
- **Applications**: Useful in genetic inheritance models, such as modeling handedness and genetic influences.

Key Concepts

Key Concepts

This section summarizes the key concepts related to Bayesian networks, emphasizing their definitions, properties, and applications in AI.

- **Marginal and Conditional Probabilities**
  - **Marginal Probability**: Probability of an event irrespective of other variables.
  - **Conditional Probability**: Probability of an event given another event.
- **Chain Rule for Bayesian Networks**
  - **Joint Distribution Calculation**: Using conditional probabilities from the network.
  - **Application**: Computes joint probabilities from a set of conditional probabilities.
- **Independence and Conditional Independence**
  - **Independence**: No probabilistic influence between variables.
  - **Conditional Independence**: Independence given a third variable.
- **Graphical Models and Probabilistic Dependencies**

- **Nodes and Edges**: Represent variables and their dependencies.
- **DAG Structure**: Defines the Bayesian network's structure.
- **Applications**: Useful in genetic modeling and other probabilistic systems.