Experiment No.7
Function Approximation in Reinforcement Learning: Using function approximation techniques, such as linear regression or neural networks, to approximate value functions in reinforcement learning problems.
Date of Performance:
Date of Submission:

EXPERIMENT 7

Aim: Function Approximation in Reinforcement Learning: Using function approximation techniques, such as linear regression or neural networks, to approximate value functions in reinforcement learning problems.

Objective: objective is to make decisions and learn optimal policies in complex environments with large state or action spaces for agent.

Theory:

Traditionally, in reinforcement learning, value functions are represented using tabular methods, where each state or state-action pair has a separate entry in a table. However, in many real-world problems, the state or action space may be too large to store all values explicitly in a table. Function approximation provides a solution to this problem by approximating the value function using a compact representation.

Linear regression is one of the simplest methods for function approximation. It involves fitting a linear model to the data, where the input features are the state or state-action representation, and the output is the estimated value function. However, linear regression may not capture complex relationships in the data and may not be suitable for high-dimensional state spaces.

Neural networks, on the other hand, offer a more flexible approach to function approximation. They can capture complex nonlinear relationships in the data and generalize well to unseen states. In reinforcement learning, neural networks are commonly used as function approximators for value functions. Techniques like deep Q-networks (DQN) and policy gradient methods leverage neural networks to approximate the Q-value or policy directly from the state space.

When using function approximation in reinforcement learning, it's essential to consider issues such as function approximation error, stability, and convergence. Techniques like experience replay and target networks are often employed to stabilize training and improve convergence when using neural networks for function approximation in reinforcement learning. Additionally, regularization techniques can help prevent overfitting and improve generalization Performance

Algorithm for Linear Regression

1. **Initialize**: Initialize the weights $\mathbf{w}$ of the linear regression model randomly or with some predefined values.
2. **Collect Data**: Interact with the environment to collect data consisting of state-action pairs (s, a) and corresponding rewards r or next states s', depending on whether you're using Monte Carlo or Temporal Difference learning.
3. **Feature Extraction**: Extract features from the state-action pairs. These features can be handcrafted based on domain knowledge or learned automatically. Let $\mathbf{x}(s, a)$ be the feature vector for state-action pair (s, a).
4. **Update Weights**: Use the collected data and features to update the weights $\mathbf{w}$ of the linear regression model. This can be done using methods such as ordinary least squares (OLS) or gradient descent.
   - For OLS: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, where $\mathbf{X}$ is the matrix of feature vectors and $\mathbf{y}$ is the vector of target values (rewards or next state values).
   - For gradient descent: $\mathbf{w} \leftarrow \mathbf{w} - \alpha\nabla_{\mathbf{w}}L$, where α is the learning rate and L is the loss function, such as mean squared error (MSE) between predicted and actual values.

5. Repeat: Repeat steps 2-4 until convergence or a predefined stopping criterion is met.

6. Policy Improvement (Optional): If using the value function for policy improvement, update the policy based on the learned value function. This can be done using methods like policy iteration or policy gradient methods.

This algorithm provides a basic framework for using linear regression to approximate value functions in reinforcement learning problems. However, it's important to note that in practice, there are many variations and optimizations that can be applied based on specific requirements and constraints of the problem domain. Additionally, techniques like eligibility traces and function approximation with linear models can also be incorporated to enhance learning efficiency and stability.

Code:

```
!pip install numpy gym
import numpy as np
import gym

class LinearApproximator:
    def __init__(self, num_features, learning_rate=0.01):
        self.weights = np.zeros(num_features)  # Initialize weights
        self.lr = learning_rate

    def predict(self, features):
        return np.dot(features, self.weights)

    def update(self, features, target):
        prediction = self.predict(features)
        error = target - prediction
        self.weights += self.lr * error * features


def featurize_state(state):
    return np.array(state)


def train_agent(env, approximator, num_episodes=100, gamma=0.99, epsilon=0.1):
    for episode in range(num_episodes):
        state = env.reset()
        done = False
        total_reward = 0

        while not done:
            # Choose action epsilon-greedily
            if np.random.rand() < epsilon:
                action = env.action_space.sample()
            else:
                features = featurize_state(state)
```

```
            next_state, reward, done, _ = env.step(action)
            total_reward += reward

            # Update weights using TD(0) update
            next_features = featurize_state(next_state)
            td_target = reward + gamma * approximator.predict(next_features)
            approximator.update(features, td_target)

            state = next_state

        print(f"Episode {episode + 1}, Total Reward: {total_reward}")


if __name__ == "__main__":
    env = gym.make("CartPole-v1")
    num_features = env.observation_space.shape[0]
    approximator = LinearApproximator(num_features)

    train_agent(env, approximator)
```

Output:

```
Episode 1, Total Reward: 10.0
Episode 2, Total Reward: 12.0
Episode 3, Total Reward: 9.0
Episode 4, Total Reward: 12.0
Episode 5, Total Reward: 10.0
Episode 6, Total Reward: 9.0
Episode 7, Total Reward: 12.0
Episode 8, Total Reward: 11.0
Episode 9, Total Reward: 9.0
Episode 10, Total Reward: 9.0
Episode 11, Total Reward: 9.0
Episode 12, Total Reward: 10.0
Episode 13, Total Reward: 12.0
Episode 14, Total Reward: 8.0
Episode 15, Total Reward: 9.0
Episode 16, Total Reward: 10.0
Episode 17, Total Reward: 9.0
Episode 18, Total Reward: 8.0
Episode 19, Total Reward: 9.0
Episode 20, Total Reward: 8.0
Episode 21, Total Reward: 9.0
Episode 22, Total Reward: 9.0
Episode 23, Total Reward: 8.0
Episode 24, Total Reward: 8.0
```

```
Episode 73, Total Reward: 9.0
Episode 74, Total Reward: 9.0
Episode 75, Total Reward: 9.0
Episode 76, Total Reward: 10.0
Episode 77, Total Reward: 8.0
Episode 78, Total Reward: 9.0
Episode 79, Total Reward: 10.0
Episode 80, Total Reward: 9.0
Episode 81, Total Reward: 10.0
Episode 82, Total Reward: 10.0
Episode 83, Total Reward: 11.0
Episode 84, Total Reward: 10.0
Episode 85, Total Reward: 9.0
Episode 86, Total Reward: 11.0
Episode 87, Total Reward: 10.0
Episode 88, Total Reward: 9.0
Episode 89, Total Reward: 12.0
Episode 90, Total Reward: 9.0
Episode 91, Total Reward: 10.0
Episode 92, Total Reward: 9.0
Episode 93, Total Reward: 10.0
Episode 94, Total Reward: 8.0
Episode 95, Total Reward: 12.0
Episode 96, Total Reward: 11.0
Episode 97, Total Reward: 10.0
Episode 98, Total Reward: 10.0
Episode 99, Total Reward: 8.0
Episode 100, Total Reward: 10.0
```

Conclusion:

1. **Explain difference between Neural network and linear regression**.

1**. Structure:**

Linear Regression: Linear regression is a simple and straightforward method used for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the input features and the target variable.

Neural Network: Neural networks are a more complex and flexible class of models inspired by the structure of the human brain. They consist of interconnected layers of neurons, including input, hidden, and output layers. Each neuron applies an activation function to the weighted sum of its inputs to produce an output.

2. Complexity:

Linear Regression: Linear regression models are relatively simple and have a linear relationship between the input features and the target variable. They are easy to interpret and understand, making them suitable for tasks where the relationship between variables is linear.

Neural Network: Neural networks are highly flexible and can capture complex nonlinear relationships between inputs and outputs. They can learn intricate patterns and features from data, making them suitable for tasks such as image recognition, natural language processing, and complex prediction problems.

3. **Learning and Training:**

Linear Regression: In linear regression, the model parameters (coefficients) are estimated using techniques such as ordinary least squares (OLS) or gradient descent. The objective is to minimize the difference between the actual and predicted values of the target variable.

Neural Network: Neural networks are trained using optimization algorithms such as gradient descent and backpropagation. During training, the network adjusts its weights and biases based on the error between the predicted and actual outputs, iteratively improving its performance over time.

4. Application:

Linear Regression: Linear regression is commonly used for tasks such as predicting house prices, estimating sales revenue, or modeling the relationship between variables in scientific studies. It's suitable when the relationship between variables is linear and well-understood.

Neural Network: Neural networks are used for a wide range of tasks, including image and speech recognition, natural language processing, time series prediction, and more. They excel in tasks where the relationship between inputs and outputs is complex or nonlinear.

5. Interpretability:

Linear Regression: Linear regression models are highly interpretable, as the coefficients indicate

the strength and direction of the relationship between each input feature and the target variable.

Neural Network: Neural networks are often considered black-box models, meaning it can be challenging to interpret how they make predictions, especially in deep neural networks with many layers. Interpretability techniques such as feature importance analysis or visualization can help gain insights into their behavior.

2. **Limitation of Linear regression** .

1. Limited Expressiveness: Linear regression can only capture linear relationships between variables. It may fail to model complex interactions or nonlinear patterns present in the data, leading to underfitting and poor predictive performance.

2. Extrapolation: Linear regression is not suitable for extrapolating beyond the range of the observed data. Predictions made outside the range of the training data may be unreliable and can lead to inaccurate results.

3. Sensitive to Data Distribution: Linear regression assumes that the residuals are normally distributed. If the residuals exhibit non-normality, such as skewness or heavy tails, the assumptions of linear regression may be violated, leading to biased parameter estimates and inaccurate inference.

4. Limited Performance on Complex Tasks: While linear regression is useful for simple prediction tasks with linear relationships, it may struggle to achieve high accuracy on complex prediction tasks with intricate interactions and nonlinear patterns in the data.