Markov decision process

Iteration k

Sk in S

Ak in a

R(Sk, Ak): reward for taking action ak in state Sk]

P(Sk+1|Sk,Ak): probability of moving to Sk+1 from Sk when action Ak is taken

Value Iteration Algorithm

Q(s,a): expected total reward for taking action a in state s

V(s): maximum expected total reward starting from state s (value function)

V(s) = max(a) Q(s,a)

Steps:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

Q value calculation:

calculates the sum of probabilities multiplied by the sum of immediate rewards and discounted future values for each possible next state.

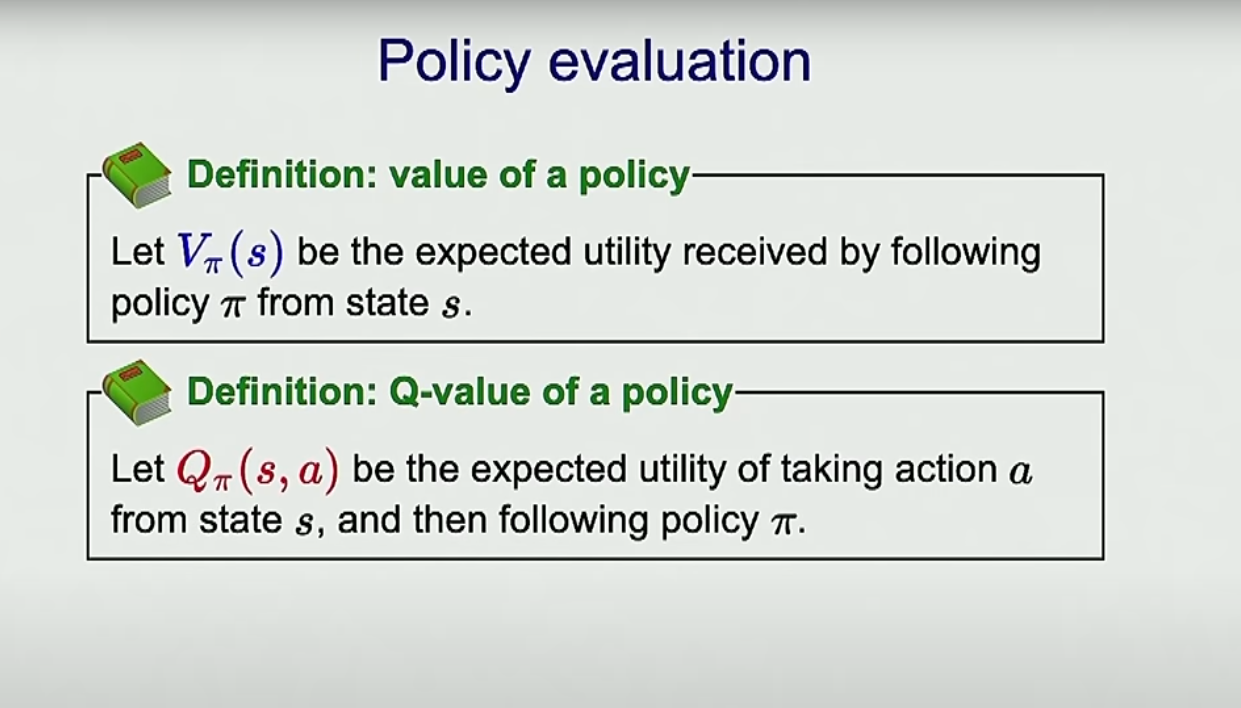
This is the implementation of the value iteration algorithm. It iterates over the specified number of iterations. For each iteration, it updates the value of each state based on the maximum Q-value of available actions.

This implementation finds the action with the highest Q-value for a given state. If there are ties, it breaks them arbitrarily by choosing the first action with the maximum Q-value.

Utility: sum of reward (of random path)

Policy: connect path to action, yields random path

Discount: save for the future



# Q1:

\_ set all states to 0

\_ loop over all states, get possible action for each state

\_ No more action to take, set max value to 0

\_Else calculate the q-values for each action in current state, pick the maximum Q value among those values and set that to max value, then stores it in a dictionary

\_ Run the loop again, the utility value for each state is updated with the maxmum q value calculated

First loop updating q value based on current policy, second loop updates utility value based on the newly calculated q values

Q value:

calculates the sum of probabilities multiplied by the sum of immediate rewards and discounted future values for each possible next state.

# Q2: bridge

Refer to the specification says that noise means how often an agent ends up in an unintended successor state, this is probability. Thus lower the accuracy to the intended action => to 0.0 is the best way

# Q3: Policies

a/

# Prefer the close exit (+1), risking the cliff (-10)

# prefer the close exit => focus on immediate rewards => discount to be close to 0 for shorterm focus

# No information about moving random, and no need to explore the map => noise not imporant => noise to 0.0

# Risking the cliff => no worry about the cliff => livingReward not important => livingReward to 0.0

# and does not require to go further => livingReward will not be important => 0.0

return answerDiscount, answerNoise, answerLivingReward

b/

# Prefer the close exit (+1), but avoiding the cliff (-10)

# prefer the close exit => focus on immediate rewards => discount to be close to 0 for shorterm focus

# avoid the cliff => need to explore => noise to be changed => 0.1

# need to explore map is already provided in Noise => livingReward can be 0.0

c/

# Prefer the distant exit (+10), risking the cliff (-10)

# prefer the distant exit => focus on long term reward => discount to be higher and closer to 1

# no information about moving random, and no need to explore map => noise to default => 0.0

# telling the needs to explore map for long term reward => setup livingReward higher than 0.0

d/

# Prefer the distant exit (+10), avoiding the cliff (-10)

# prefer the distant exit => focus on long term reward => discount to be higher and closer to 1

# avoid the cliff => need to explore => noise to be changed => 0.1

# telling the needs to explore map for long term reward => setup livingReward higher than 0.0

e/

# Avoid both exits and the cliff

# no need to win => discount not important => 0.0

needs to avoid cliff and exits => the need to explore is high => more noise to close 1

answerLivingReward = 0.9

is even higher => close to 1

# Q 4:

Model free learning , AI makes own policy by interacting with environment without needing a model beforehand

Q learning = model free learning tech that can be used to find the optiomal policy by using a q function

Computevaluefromqvalue: get q value for each action, then calculate the max q value

ComputeActionFromQValue: store all the action that have same max q value , then pick out a random one

Update: compute temperal difference error, which is difference between the observed reward and the future rewards based on the q values of the next state

Blends current q value with the new information, scaled by learning rate

# Q5:

GetAction: Simulate coin flip with a success probability given by epsilon, if successful randomly select action computed from q value

Use epsilon greedy strat, choose a random option if coinflip unsuccessful, else take the action with highest q value

# Q 6:

# the problem is:

# epsilon is about exploration and exploitation: => epsilon to 0.0 to be the best because lower the epsilon => more exploitation

# learning rate is about map adaption and stable learning => can be vary for 0.0 to 1

# After trying case for epsilon 0.0 and learning from 0.0 to 1 => the program cant

# => NOT POSSIBLE

# Q7:

Epsilon is positive during training, so Pacman will play poorly even after having learned a good policy: this is because he occasionally makes a random exploratory move into a ghost. As a benchmark, it should take between 1,000 and 1400 games before Pacman's rewards for a 100 episode segment becomes positive, reflecting that he's started winning more than losing. By the end of training, it should remain positive and be fairly high (between 100 and 350).

def \_\_init\_\_(self, epsilon=0.05,gamma=0.8,alpha=0.2, numTraining=0, \*\*args):

"""

These default parameters can be changed from the pacman.py command line.

For example, to change the exploration rate, try:

python pacman.py -p PacmanQLearningAgent -a epsilon=0.1

alpha - learning rate

epsilon - exploration rate

gamma - discount factor

numTraining - number of training episodes, i.e. no learning after these many episodes

# Q8:

First we calculate the approximate q value using the given formula

Then we calculate the difference vector and using that to calculate the weight

# Script:

Bayes’ Theorem: is a principle that describes how to update the probabilities of hypotheses when given evidence.

Bayesian networks are a graphical tool for applying Bayes’ theorem to complex problems involving multiple variables and their dependencies.

Alcohol is the parent node of both heartburn and stomach cancer because it causes both. Stomach cancer is a parent node of heartburn because it does cause heartburn, heartburn is not a parent node

Here u can see the prob of drinking alcohol, with 30% of the world population being an avid consumer. Next is the prob of getting cancer when drinking alcohol. With 20% if you drink alcohol for a long time and 5% if you don’t

Lastly the probability of getting heartburn with drinking alcohol and have cancer. With 90% chance of heartburn while drinking alcohol and have cancer, 60% if you only drink alcohol and 80% if you only have cancer, there is also a 30% chance of getting a heartburn for none of the reasons listed

Hardworking have no parent node but it causes stress, and stress can cause gastritis, drinking problem and heartburm