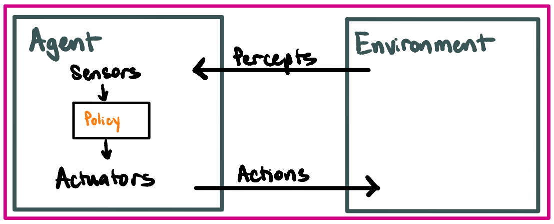
CS 188 Midterm 1 Study Guide

Jeffrey Shen

① Introduction



- An agent percieves its environment through sensors and acts upon it through actuators
- A rational agent chooses action to max expected value

② Search

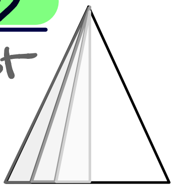
Search Problems have:

- State Space S
- Initial State S_0
- Goal test $G(s)$
- Action cost $C(s,a,s')$
- Transition model $T(s,a)$
- Actions of state $A(s)$

Solutions: action sequence that reaches goal state

↳ Optimal Solution: least cost among solutions

Depth First Search (DFS)

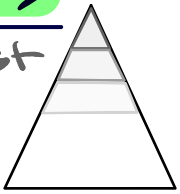


Strat: expand deepest node first

Use: LIFO stack

Optimal?: No, finds "leftmost" sol

Breadth First Search (BFS)

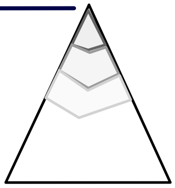


Strat: expand shallowest node first

Use: FIFO Queue

Optimal: If costs are equal

Uniform Cost Search (UCS)

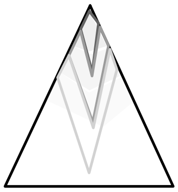


Strat: expand lowest cost from root $g(n)$

Use: Priority Queue by cost

Optimal?: Yes

A^* Search



Strat: expand node most likely on optimal path using heuristic

Use: Priority Queue by $g(n)+h(n)$

Optimal: Yes, if heuristic admissible & consistent

Admissible: $0 \le h(n) \le h^*(n)$

heuristic ≤ actual cost ↑ true cost to goal

Consistent: $h(A) - h(C) \le c(A,C)$

heuristic "arc" cost ≤ actual arc cost

Graph Search: track explored nodes in list ↑ prevent revisiting nodes

Tree Search: map paths w/ tree, node appear multiple times ↑ less memory

Greedy Search

Strategy: Expand node w/ lowest heuristic value, believed closest to goal

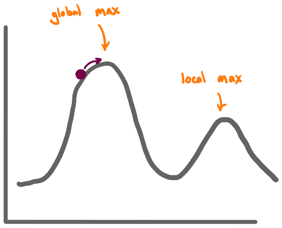
Use: priority queue with heuristics, estimated forward cost

Optimal: No, not guarenteed to find goal state, unpredictable

Local Search

Hill Climbing

Idea: Start wherever, repeat: move to the best neighboring state. If no neighbors better than current, quit



```
function HILL-CLIMBING(problem) returns state
    current ← make-node(problem.initial state)
    loop do
        neighbor ← highest valued successor of current
        if neighbor.value ≤ current.value then
            return current.state
        current ← neighbor
```

Random-restart hill climbing trivally complete
- restarts from random initial state

Simulated Annealing

Idea: random walk + hill climbing, choose randomly, choose worse w/ some probability according to temperature, temp starts high w/ more "bad" moves allowed and decreases

```
function SIMULATED_ANNEALING(problem, schedule) returns state
    current ← problem.initial-state
    for t=1 to ∞ do
        T ← schedule(t)
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← next.value − current.value
        if ΔE > 0 then current ← next
            else current ← next w/ prob $e^{\Delta E/T}$
```

If T decreased slowly enough, will converge to optimal state

Local Beam Search

multiple interaction searches

Idea: track k states at each iteration, k threads share information good threads attract other threads, chooses k best successors

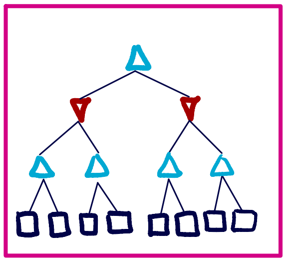
Genetic Algorithms

Break and recombine states

Idea: beam search k states in population, each state evaluated w/ evaluation function (fitness func), offspring produced by crossing parent strings at crossover point

③ Games

Zero sum games: our gain is directly equivalent to opponent's loss



Minimax

algorithm to counter optimal opponent moves

Idea: Zero sum algo assuming opponent plays optimally

terminal utilities/state value: optimal score attainable

```
function minimax-decision(s) returns action
    return action a in Actions(s) w/ highest minimax_value(Result(s,a))
function minimax_value(s) returns a value
    if Terminal-Test(s) then return Utiliy(s)
    if Player(s)= MAX then return $\max_{a \in Actions(s)}$ minimax_value(Result(s,a))
    if Player(s)= MIN then return $\min_{a \in Actions(s)}$ minimax_value(Result(s,a))
```

Alpha-Beta Pruning

optimization to minimax : $O(b^m) \Rightarrow O(b^{m/2})$

tree depth

branching factor

Idea: Stop looking as soon as you know n's value can at best equal optimal value of n's parent

α: MAX's best option on path to root

β: MIN's best option on path to root

```
def max-value(state, $\alpha$, $\beta$)
    initialize $v = -\infty$
    for each successor of state:
        $v = \max(v$, value(successor, $\alpha$, $\beta$))
        if $v \geq \beta$ return v
        $\alpha = \max(\alpha, v)$
    return v
```

```
def min-value(state, $\alpha$, $\beta$)
    initialize $v = +\infty$
    for each successor of state:
        $v = \min(v$, value(successor, $\alpha$, $\beta$))
        if $v \leq \alpha$ return v
        $\beta = \min(\beta, v)$
    return v
```

Evaluation functions are used in **depth-limited minimax** to output an estimate of the true minimax value of the node

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

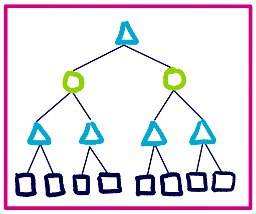
weight associated w/ feature

features of state

Expectimax

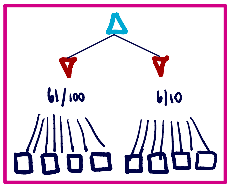
algorithm with randomness using Expected value

Idea: Chance nodes calculated by taking expected utility of children



```
function decision(s) returns action
    return action a in Actions(s) w/ highest value(Result(s,a))
function value(s) returns a value
    if Terminal-Test(s) then return Utility(s)
    if Player(s)= MAX then return $\max_{a \in Actions(s)}$ value(Result(s,a))
    if Player(s)= MIN then return $\min_{a \in Actions(s)}$ value(Result(s,a))
    if Player(s)=CHANCE then return $\text{sum}_{a \in Actions(s)}$ Pr(a) * value(Result(s,a))
```

Monte Carlo Tree Search (MCTS)



Idea:

1) Evaluation by rollouts: From state s play many times using policy and count wins and losses

2) Selective search: explore parts of the tree, without constraints on horizon, that will improve decision at root

UCB Algorithm

total wins for Player(Parent(n))

$$UCB1(n) = \frac{U(n)}{N(n)} + C\sqrt{\frac{\log N(PARENT(n))}{N(n)}}$$

user specific param to balance exploration vs. exploitation

total rollouts from n

4 Logic

Maintain a knowledge base w/ logical sentences that can make logical inferences

Logic Syntax

Symbol	Meaning	Description
¬	not	¬P true iff P is false
∧	and (conjunction)	A ∧ B true iff both A true and B true
∨	or (disjunction)	A ∨ B true iff either A true or B true
⇒	implication	A ⇒ B true unless A true and B false
⇔, ≡	biconditional	A ⇔ B (A≡B) true iff either both A,B true or false

Conjunctive normal form (CNF): conjunction of clauses that are disjunction of literals

$(P_1 \vee \dots \vee P_i) \wedge \dots \wedge (P_j \vee \dots \vee P_n)$

Converting to CNF:

1) Eliminate $\Leftrightarrow$
2) Eliminate $\Rightarrow$
3) Not's ($\neg$) only on literals
4) Reduce and distribute

$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$

$\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$

$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta),\ \neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$

$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)),$

$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

model: assignment of true/false to all proposition symbols

valid: true in all models

satisfiable: at least one model where true

unsatisfiable: not true in any models

First Order Logic (FOL)

uses objects as base components

Use **function symbols** to name objects in terms

quantifiers: 1) universal quantifier $\forall$: "for all"

2) existential quantifier $\exists$: "there exists"

Operator Precedence
$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Propositional Logical Inference

Entails ($\models$): in all models where A is true, B is as well, $M(A) \subseteq M(B)$

Show by:

1) $A \models B$ iff $A \Rightarrow B$ valid — direct proof
2) $A \models B$ iff $A \wedge \neg B$ unsatisfiable — proof by contradiction

Model Checking

DPLL Algorithm

depth-first, back-tracking search w/ tricks SAT solver

Idea: Solve satisfiability problem given in CNF, continue assigning truth values until found or cannot be found, backtrack w/ conditions:

1) **Early Termination:** clause true if any symbol true, sentence false if any single clause is false
2) **Pure Symbol Heuristic:** symbol only shows up in pos/neg form assigned immediately

   $(A \vee B)(\neg B \vee C) \wedge (\neg C \vee A)$ A=True
3) **Unit Clause Heuristic:** clause w/ one literal or literal w/ many falses

Theorem Proving

Resolution Algorithm

Idea: iteratively apply to knowledge base either q inferred or nothing left to infer

Forward Chaining Algorithm

Idea: data driven reasoning, iterating through every implication statement where LHS known, adding RHS to facts using Generalized Modus Ponens

Solve inference in FOL by Generalized Modus Ponens or propositionalization translating problem into propositional logic and using SAT solver

⑤ Probability

$\perp\!\!\!\perp$: Conditional independence

Marginal distribution: $P(A,B) = \sum_{c} P(A,B,C)$

Bayes' rule:

$$P(A|B,C) = \frac{P(A,B,C)}{P(B,C)} = \frac{P(A,B|C)}{P(B|C)}$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Chain Rule: $P(A,B,C) = P(A,B|C)P(C) = P(A|B,C)P(B|C)\,P(C)$

A conditionally independent given C: $P(A,B|C) = P(A|C)P(B|C)$

A independent of B given C: $P(A|B,C) = P(A|C)$

A, B independent: $P(A,B) = P(A)P(B)$

$$P(Q_1 \ldots Q_k | e_1 \ldots e_k)$$

Query Variables
unknown, trying to compute

Evidence variables
observed, values known

Hidden Variables
values present but not in distribution we are computing

Bayes Nets

Idea: Joint distribution distributed across smaller probability tables with Directed Acyclic Graph (DAG)

1) DAG w/ node per variable X
2) conditional distribution for each node $P(X|A_1 \ldots A_n)$ where A_i is i^{th} parent of X, stored as conditional probability table (CPT)

Each node is conditionally independent of all ancestor nodes in graph, given all of parents

Bayes Net Inference (calculating joint probability)

Eliminate vars by: 1. Join (multiply together) all factors involving X
2. Sum out X

$$P(X=1 \mid A=+, B=+) = P(X=1)P(A=+ \mid X=1)P(B=+ \mid X=1)$$

Prior Sampling: Randomly generate samples

Rejection Sampling: early reject any sample inconsisent w/ evidence

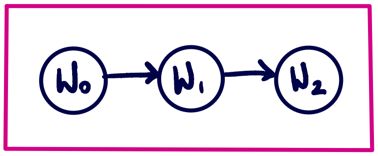
Likelihood Weighting: Ensure we never generate a bad sample

1) manually set variables to equal evidence
2) Weight each sample by probability of the evidence variables given the sampled variables

Gibbs Sampling

Set all variables to random value, repeatedly pick one variable at a time, clear value, resample

⑥ Markov Models



Chain like infinite-length Bayes Net

Need to know initial state and transition model

Mini-Forward Algorithm

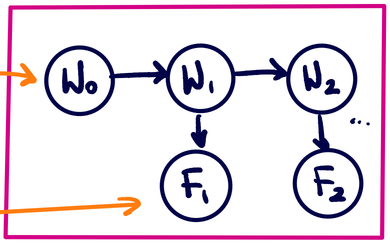
$$Pr(W_{i+1}) = \sum_{w_i} Pr(W_{i+1} \mid w_i)\, Pr(w_i)$$

Stationary Distribution

$$Pr(W_{t+1}) = Pr(W_t) = \sum_{w_t} Pr(W_{t+1} \mid w_t)\, Pr(w_t)$$

Hidden Markov Models

allows to observe some evidence at timestep
and affects belief distribution



State variable
encodes belief

evidence var

Forward Algorithm

$$B'(W_{t+1}) = \sum_{w_i} Pr(W_{t+1}|w_i) B(w_i)$$

Time elapse update

$$P(x_t|o_{0:t-1}) = \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|o_{0:t-1})$$

o: observation

$$B(W_{t+1}) \propto Pr(f_{i+1}|W_{i+1}) B'(W_{i+1})$$

Observation update

$$B(W_{i+1}) \propto Pr(f_{i+1}|W_{i+1}) \sum_{W_i} Pr(W_{i+1}|w_i) B(w_i)$$

$$P(x_t|o_{0:t}) \propto P(x_t, o_t|o_{0:t-1}) = P(o_t|x_t) P(x_t|o_{0:t-1})$$

Viterbi Algorithm

Dynamic Programming algorithm to solve

$$\arg\max_{x_{1:N}} P(x_{1:N}|e_{1:N}) = \arg\max_{x_{1:N}} P(x_{1:N}, e_{1:N})$$

$$m_t[x_t] = P(e_t|x_t) m_{T-1} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

CS 188 Final Study Guide Jeffrey Shen

1 Dynamic Bayes Nets

track multiple variables over time, using multiple source evidence

Idea: Repeat fixed Bayes net structure at each time

Particle Filtering simulating motion of set of particles through state graph to approx prob distr

1) Prediction Step: sample new state from transition model

particle j state $x_t^{(j)}$ $x_{t+1}^{(j)} \sim P(X_{t+1} | x_t^{(j)})$

2) Update Step: weight each sample on evidence, then normalize

$$w^{(j)} = P(e_{t+1} | x_j^{(j)})$$

Exact Inference: Calc exact probability more accurate but hard w/ large belief distribution

2 Rational Decisions

Principle of Maximum Expected Utility (MEU): rational agents always select action to max utility

Preferences:

$A \succ B$: prefers A over B $A \sim B$: indifferent between A and B

Lottery: A recieved w/ probability p, B w/ probability $1-p$

$$L = [p, A; (1-p), B]$$

Axioms of Rationality:

1) Orderability: must prefer A or B, or be indifferent $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
2) Transitivity: if prefers A to B and B to C, prefers A to C $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
3) Continuity: if prefers A to B, B to C, lottery L w/ A, C possible st. indifferent betwen L and B for some p $A \succ B \succ C \Rightarrow \exists p \, [p, A; (1-p), C] \sim B$
4) Substitutability: indifferent between A and B also indifferent for lottery substitutions $A \sim B \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$
5) Monotonicity: prefers A over B, chooses lottery w/ highest A prob $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; (1-p), B] \succeq [q, A; (1-q), B]$

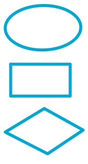
3 Decision Networks

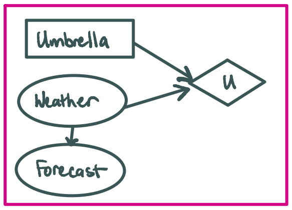
used to model state, actions, utilities of subsequent states

Chance Nodes: outcome has associated probability

Action Nodes: choices between actions we can choose

Utility Nodes: children of chance & action representing utility





Value of Perfect Information (VPI) expected improvement in decision quality from observing value

VPI $(E' | e)$ is value of observing new evidence E' given current evidence e

$$\boxed{VPI(E'|e) = MEU(e, E') - MEU(e) = \left[\sum_{e'} P(e'|e) \max_a \sum_s P(s|e,e') U(s,a)\right] - \max_a \sum_s P(s|e) U(s,a)}$$

VPI Properties

1) Nonnegativity: more informed decision $\forall E', e, \; VPI(E'|e) \geq 0$
2) Nonadditivity: can change how much we care about E_k $VPI(E_j, E_k | e) \neq VPI(E_j|e) + VPI(E_k|e)$
3) Order-independence: order of observation doesn't matter $VPI(E_i, E_j | e) = VPI(E_j, E_i | e)$

④ Markov Decision Process (MDP)

solving nondeterministic search problems

Markov Decision Process properties:

- state space S
- actions A
- Transition function $T(s,a,s')$ probability taking action a at states results in s' $P(s'|s,a)$
- Reward function $R(s,a,s')$ pos/neg reward for each step
- terminal state(s)
- start state S_0
- discount factor γ multiply to reward for exponential decay in reward "time constraint"

Policy π : $S \rightarrow A$, mapping each state to an action

State Optimal Value $U^*(s)$: expected utility of optimal agent from state s

Optimal Q Value $Q^*(s,a)$: expected utility of optimal agent from state s taking action a

Policy Extraction: used to determine policy given some state value function

Value Iteration

DP algorithm to compute MDP until convergence of values

1) $\forall s \in S$, set $U_0(s) = 0$
2) Repeat until convergence:

$$\boxed{\forall s \in S, U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U_k(s')\right]}$$

Runtime: $O(|S|^2|A|)$

Policy Iteration

more efficient method to converge to optimal policy

1) Define an initial policy
2) Use policy evaluation for current policy
3) Use policy improvement to find better policy

$$\boxed{U^{\pi}(s) = \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma U^{\pi}(s')\right]}$$

$$\boxed{\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma U^{\pi_i}(s')\right]}$$

⑤ Machine Learning

learning parameters of specified model given data

Decision Trees

simple nonlinear classifier but susceptible to over-fitting

Entropy: uncertainty about variable ↑ entropy ↑ uncertainty

$$\boxed{H(V) = -\sum_k P(v_k)\log_2 P(v_k)}$$

entropy of binary variable

$$B(q) = -q\log_2 q - (1-q)\log_2(1-q)$$

Information Gain: info gained by splitting on that feature

$$\boxed{Gain(Type) = B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{K} \frac{p_k+n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)}$$

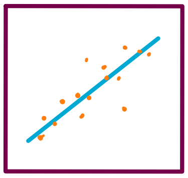
Linear Regression

linear classifier for continous var output

L2 Loss function: metric to measure our model using squared difference $(y-h(x))^2$

Least squares:

$$\boxed{\hat{w} = (X^T X)^{-1} X^T y}$$



Logistic Regression

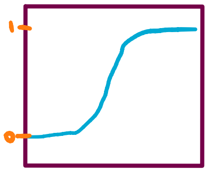
linear classifier for categorical variables

Logistic function:

$$\boxed{h_w(x) = \frac{1}{1+e^{-wx}}}$$

Find optimal using gradients

Models $P(c|x)$ where c: class, x: evidence



Perceptron

Linear classifier: classification using linear combination of features

Activation:

$$\text{activation}_w(x) = h_w(x) = w \cdot f(x)$$

$$\text{classify}(x) = \begin{cases} + & \text{if } h_w(w) > 0 \\ - & \text{if } h_w(w) < 0 \end{cases}$$

Perceptron Algorithm:

1) Initialize all weights to 0: $w = 0$
2) Classify y and compare to true label y^*

if $y = y^*$, do nothing

if $y \neq y^*$, update $\boxed{w \leftarrow w + y^* f(x)}$

Naive Bayes

model features as Bayes Net, assumes each feature is independent from others

Prediction for class label becomes $\text{prediction}(F) = \underset{y_i}{\text{argmax}}\, P(Y=y_i) \prod_j P(F_j = f_j \mid Y = y_i)$, normalize

Maximum Likelihood Estimation (MLE):

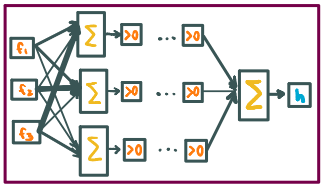
given i.i.d., method to learn probability, count

Likelihood: $\boxed{\mathcal{L}(\theta) = \prod_{i=1}^{N} P_\theta(x_i)}$ take gradient $\frac{\partial}{\partial\theta}\mathcal{L}(\theta) = 0$ to get max, log likelihood $\log \mathcal{L}(\theta)$

Laplace smoothing: mitigate overfitting, assume seen k extra each of each outcome

$$P_{LAP,k}(x) = \frac{count(x) + k}{N + k|X|}$$

Neural Networks



Multi-layer perceptron map data to higher dimension then classify

Universal func approximater: two layer NN w/ sufficient neurons can approx any cont. func

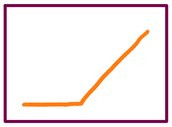
can use indicator function of sgn threshold, softmax func to classify

Log Likelihood: $\boxed{\log \ell(w) = \prod_{i=1}^{n} \log P(y_i \mid f(x_i); w)}$

Activation functions

sigmoid: $\sigma(x) = \left\{ \frac{1}{1+e^{-x}} \right.$

ReLU: $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$



Backpropogation determine gradient of output w/ respect to each of inputs

1) Forward pass: compute values through computation graphs
2) Backwards Pass: compute gradients taking advantage of chain rule

⑥ Reinforcement Learning

method for solving MDP w/o transition & reward functions

Episode: collection of samples which are (s, a, s', r) tuples

Model-based learning: estimate transition, reward functions w/ samples before using policy/value ite

- count times arrived in state and normalize counts, Law of Large Numbers will converge on optimal

Model-free learning: estimate Q values directly w/o constructing model

↳ **Passive reinforcement learning:** agent given policy to follow and learns values of state exploring

1) Direct Evaluation: fix policy π and have agent experience, can compute by averages
2) **Temporal Difference (TD) learning:** learning from every experience

$$\boxed{\begin{aligned} \text{sample} &= R(s, \pi(s), s') + \gamma V^\pi(s') \\ V_k^\pi(s) &\leftarrow (1-\alpha) V_{k-1}^\pi(s) + \alpha \cdot \text{sample} \end{aligned}}$$

↳**Active Reinforcement learning**: learning agent can use feedback to iteratively update policy

3) **Q-Learning**: Bypass model by directly learning Q values

Q-Value Iteration:

$$\boxed{Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]}$$

Q value samples:

$$\boxed{\begin{aligned} sample &= R(s,a,s') + \gamma \max_{a'} Q(s',a') \\ Q(s,a) &\leftarrow (1-\alpha)\, Q(s,a) + \alpha \cdot sample \end{aligned}}$$

Feature Based Representation: allow model to generalize learning experiences, store as linear func

Weight update

$$\boxed{w_i \leftarrow w_i - \alpha \cdot sample \,\frac{\partial Q_w}{\partial w_i}}$$

negative for gradient descent

Exploration vs. Exploitation

ε-greedy: $0 \le \varepsilon \le 1$, act randomly w/ prob ε, should be lowered over time

Exploration functions:
modified update

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} f(s',a')\right]$$

$$f(s,a) = Q(s,a) + \frac{k}{N(s,a)}$$

k ← predetermined

$N(s,a)$ ← # times Q state visited

Regret: metric to measure model, difference between total reward for optimal and reward in our model