Search

- In *partially observable* environments, the agent does not have full information about the state and thus the agent must have an internal estimate of the state of the world. This is in contrast to *fully observable* environments, where the agent has full information about their state.
- 随机的 *Stochastic* environments have uncertainty in the transition model, i.e. taking an action in a specific state may have multiple possible outcomes with different probabilities. This is in contrast to *deterministic* environments, where taking an action in a state has a single outcome that is guaranteed to happen.
- In *multi-agent* environments the agent acts in the environments along with other agents. For this reason the agent might need to randomize its actions in order to avoid being "predictable" by other agents.
- If the environment does not change as the agent acts on it, then this environment is called *static*. This is contrast to *dynamic* environments that change as the agent interacts with it.
- If an environment has *known physics*, then the transition model (even if stochastic) is known to the agent and it can use that when planning a path. If the *physics are unknown* the agent will need to take actions deliberately to learn the unknown dynamics.
- A **transition** model - Outputs the next state when a specific action is taken at current state

CSP: Unary Constraints
Binary Constraints
Unconventional.

1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints

2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other

K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node.

Filtering:
Ordering: MRV (Minimum Remaining Values (variable))
LCV (Least Constraining Value)
prune fewest value

Structure: 总量 子问题变量
①子问题: $O(\frac{n}{c}) d^c) << d^n$
子问题变量 可选值

树结构: $O(nd^2)$

Hill-climbing:
not complete
not optimal

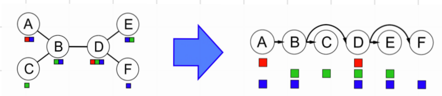
Simulated Annealing
T越慢，收敛速度exponentially slowly.

Alpha-Beta

DFS: · not complete (有圈的话)
· not optimal
· time complexity: $O(b^m)$ → 深度max
branch factor.
· space cpx: $O(bm)$

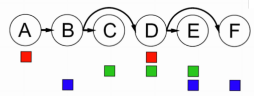
BFS: -不一定optimal. 始终停止!
-complete.
-Time complexity: $1+b+b^2+...+b^s$
$=O(b^s)$
-Space: $O(b^s)$

UCS:
completeness, optimal (edge > 0)
Time Complexity:
C^*: cost of optimal path.
ε: minimal cost btw 2 nodes
depth: about C^*/ε
→ $O(b^{C^*/\varepsilon})$
Space complexity:
frontier 可能包含所有nodes:
→ $O(b^{C^*/\varepsilon})$



①选择结点，排序
②孩子→父亲，弧相容
$D\to F$, $D\to E$, $B\to D$, $B\to C$, $A\to B$

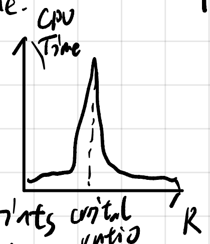
③根到叶子赋值



local search
①random assignment
②random conflicted variable.
③改成 min-conflicted.
both ① incomplete
② suboptimal

critical ratio:

$$R=\frac{\text{number of constraints}}{\text{number of variables}}$$



Alpha-Beta Implementation

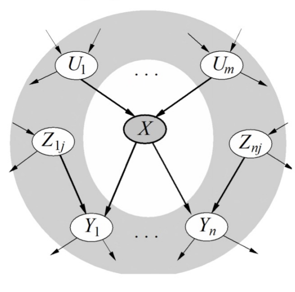
α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

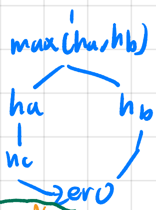
```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

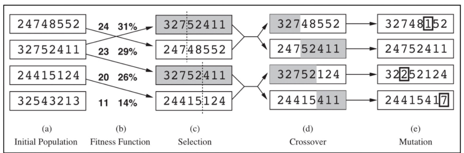
Conditional independence:
$p(A|B)=p(A)$

Each node, given its MarkovBlanket, is conditionally independent of all other nodes in the graph



MarkovBlanket refers to the parents, children, and children's other parents.





(a) Initial Population	(b) Fitness Function		(c) Selection	(d) Crossover	(e) Mutation
24748552	24	31%	32752411	32748552	32748152
32752411	23	29%	24748552	24752411	24752411
24415124	20	26%	32752411	32752124	32252124
32543213	11	14%	24415124	24415411	24415417

Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

- Inference by Enumeration $=\sum_t\sum_r P(L|t)P(r)P(t|r)$
- Variable Elimination $=\sum_t P(L|t)\sum_r P(r)P(t|r)$

A* $\forall n,\ 0\le h(n)\le h^*(n)$

Admissibility:
A & B,
optimal ↑ ↖ suboptimal
终点.
① $f(A)=g(A)+h(A)=g(A)$
$<g(B)+h(B)=f(B)$
② $f(n)\le f(A)<f(B)$
↓
n 肯定先拓展

Consistency: $\forall A,C\quad h(A)-h(C)\le cost(A,C)$
proof: remove a node for expanding → node is optimal

$f(n')=g(n')+h(n')$
$=g(n)+c(n,n')+h(n')$
$\ge g(n)+h(n)$
$=f(n)$

1. $g(A)<g(B)$. Because A is given to be optimal and B is given to be suboptimal, we can conclude that A has a lower backwards cost to the start state than B.
2. $h(A)=h(B)=0$, because we are given that our heuristic satisfies the admissibility constraint. Since both A and B are both goal states, the true optimal cost to a goal state from A or B is simply $h^*(n)=0$; hence $0\le h(n)\le 0$.
3. $f(n)\le f(A)$, because, through admissibility of h, $f(n)=g(n)+h(n)\le g(n)+h^*(n)=g(A)=f(A)$. The total cost through node n is at most the true backward cost of A, which is also the total cost of A.

Informed search

Greedy:
lowest heuristic value.
priority queue
✗ complete ✗ optimal

Variable Elimination

R → T → L

$P(R)$

T	L
+r	0.1
-r	0.9

$P(T|R)$

+r	+t	0.8
+r	-t	0.2
-r	+t	0.1
-r	-t	0.9

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

Join R

$P(R,T)$

+r	+t	0.08
+r	-t	0.02
-r	+t	0.09
-r	-t	0.81

R,T → L

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

Sum out R

+t	0.17
-t	0.83

T → L

$P(L|T)$

+t	+l	0.3
+t	-l	0.7
-t	+l	0.1
-t	-l	0.9

Join T

T,L

$P(T,L)$

+t	+l	0.051
+t	-l	0.119
-t	+l	0.083
-t	-l	0.747

Sum out T

L

+l	0.134
-l	0.866

HMM: stationary Distribute.

$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

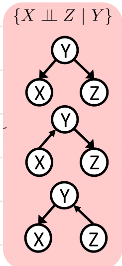
Mini-Forward

$$P(X_t) = \sum_{X_{t-1}} P(X_{t-1}) P(X_t|X_{t-1})$$

Forward Alg:

$$P(X_t|e_{1:t-1}) = \sum_{X_{t-1}} P(X_{t-1}|e_{1:t-1}) \cdot P(X_t|X_{t-1})$$

$$P(X_t|e_{1:t}) \propto_X P(X_t|e_{1:t-1}) \cdot P(e_t|X_t)$$



Sampling.

Prior Sampling.

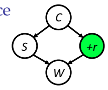
- For i = 1, 2, …, n in topological order
  - Sample x_i from P(X_i | Parents(X_i))
- Return (x_1, x_2, …, x_n)

Rejection Sampling.

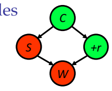
- Input: evidence instantiation
- For i = 1, 2, …, n in topological order
  - Sample x_i from P(X_i | Parents(X_i))
  - If x_i not consistent with evidence
    - Reject: return – no sample is generated in this cycle
- Return (x_1, x_2, …, x_n)

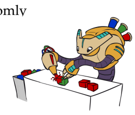
Gibbs Sampling

- Step 1: Fix evidence
  - R = +r

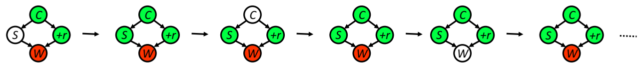


- Step 2: Initialize other variables
  - Randomly





- Steps 3: Repeat:
  - Choose a non-evidence variable X
  - Resample X from P(X | MarkovBlanket(X))



Sample from $P(S|+c,-w,+r)$ Sample from $P(C|+s,-w,+r)$ Sample from $P(W|+s,+c,+r)$

Likelihood Weighting.

- Input: evidence instantiation
- w = 1.0
- for i = 1, 2, …, n in topological order
  - if X_i is an evidence variable
    - X_i = observation x_i for X_i
    - Set w = w * P(x_i | Parents(X_i))
  - else
    - Sample x_i from P(X_i | Parents(X_i))
- return (x_1, x_2, …, x_n), w

most probable path:

Arc weight: $P(X_t|X_{t-1}) \times P(e_t|X_t)$

path weight: Arc weight 乘起来!

Forward Alg: sum of paths

Viterbi Alg: best paths

Time: $O(|X|^2 T)$

Space: $O(|X| T)$

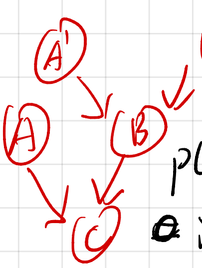
Number of path: $O(|X|^T)$

$$f_t[x_t] = P(x_t, e_{1:t})$$
$$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) f_{t-1}[x_{t-1}]$$

$$m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$$
$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$$

(iii) [2 pts] You want to run UCS to find an optimal solution but your program has limited memory that allows only at most X states to be stored on the frontier. Let b be the branching factor for the problem and c be the cost of the optimal solution. Determine the correct upper bound on c such that UCS is guaranteed to find the optimal solution while staying within the space constraint.

$c \le$ $b^c \le X \Rightarrow c \le \log_b X$



Assume $P(C=c|B=b) = x$

$P(C=c|B=\neg b) = y$

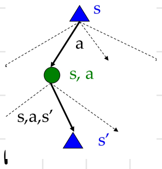
$p(B=b|C=c, B'=b)$

$x \cdot x = p(C=c, B=b \mid B'=b)$

$$\rightarrow \frac{x \cdot x}{p(C=c|B'=b)} = \frac{x \cdot x}{x \cdot x + (1-x) \cdot y}$$

$$g + h_2 = g + 2h_1 \le 2(g + h_1) \le 2C^*$$

MDPs!



Define:

$T(s,a,s')$: Prob

$R(s,a,s')$

Bellman Equation

$$V^*(s) = \max_a Q^*(s,a)$$

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

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

Particle Filtering

① Initialize Particles.

② Repeat

1. Observation update $w(x) = P(e|x)$

i. Calculate weights of particles

ii. Calculate total weight for each state

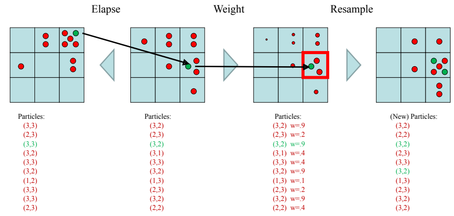
iii. If sum of all weight = 0, re-initialize

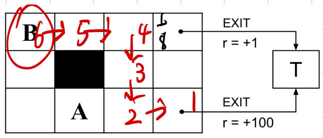
Else: normalize, resample particles

2. Time elapse update.

Recap: Particle Filtering

- Particles: track samples of states rather than an explicit distribution





Map of Reinforcement Learning

Known MDP: Offline Solution

Goal	Technique
Compute V^*, Q^*, π^*	Value / policy iteration
Evaluate a fixed policy π	Policy evaluation

Unknown MDP: Model-Based

Goal	Technique
Compute V^*, Q^*, π^*	VI/PI on approx. MDP
Evaluate a fixed policy π	PE on approx. MDP

Unknown MDP: Model-Free

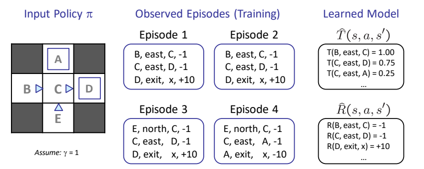
Goal	Technique
Compute V^*, Q^*, π^*	Q-learning
Evaluate a fixed policy π	Value Learning

Value iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

$V_0 = 0$!! 易无出

Example: Model-Based Learning



Policy Evaluation, Extraction, & Iteration

- Policy Evaluation calculates the V's for a fixed policy
  - $V^{\pi}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^{\pi}_k(s')]$
- Policy extraction determines optimal policy given optimal values V*(s)
  - $\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- Policy iteration lets us find the optimal policy *faster* than value iteration
  - Evaluation: $V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')\left[R(s,\pi_i(s),s') + \gamma V^{\pi_i}_k(s')\right]$
  - Improvement: $\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$

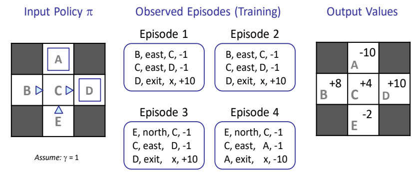
One iteration of

Value iter: $O(|S|^2|A|)$

policy: $O(|S|^3)$

→解能

Example: Direct Evaluation → model free!



model-free!

Direction Evaluation: calculate state value from episodes, not single transitions.

RL!

① TD learning × find opt policy

only learn value of states

TD Learning

1. Initialize all $V^{\pi}(s)$ to 0, determine $\pi(s)$ and α in (0,1]
2. Repeat
   a. Take sample $(s, \pi(s), s')$

   $$\boxed{\text{sample} = R(s,\pi(s),s') + \gamma V^{\pi}(s')}$$

   b. Incorporate sample into exponential moving average of $V^{\pi}(s)$

   $$\boxed{V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \cdot \text{sample}}$$

   (Slowly decrease α from 1 to 0)

Approximate Q-learning

Idea: Q-learning is not feasible if there are thousands of states

Fix: store each state as linear combinations of features

Every V(s) & Q(s, a) has a feature vector, and you want to tune the weight vector

$$V(s) = w_1 \cdot f_1(s) + w_2 \cdot f_2(s) + \ldots + w_n \cdot f_n(s) = \vec{w} \cdot \vec{f}(s)$$

$$Q(s,a) = w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + \ldots + w_n \cdot f_n(s,a) = \vec{w} \cdot \vec{f}(s,a)$$

Update rule:

$$\text{difference} = [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$$

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

Q learning

1. Initialize $Q(s,a) = 0$, α
2. Repeat

off-policy, √ optimal value but act sub opt

$$\boxed{\text{sample} = R(s,a,s') + \gamma \max_{a'} Q(s',a')}$$

$$\boxed{Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample}}$$

$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma r_{t+1} + \gamma^2 \max_{a'} Q(s_{t+2},a'))$

(Sample of the expected discounted reward using r_{t+1})

$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma((1-\alpha)Q(s_{t+1},a_{t+1}) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+2},a'))))$

(Nested Q-learning update)

$Q(s_t,a_t) = (1-\alpha)Q(s_t,a_t) + \alpha(r_t + \gamma \max((1-\alpha)Q(s_{t+1},a_{t+1}) + \alpha(r_{t+1} + \gamma \max_{a'} Q(s_{t+2},a')), \max_{a'} Q(s_{t+1},a')))$

(Max of normal Q-learning update and one step look-ahead update)

Exploration vs Exploitation

- How do we ensure that we've explored a sufficient amount?

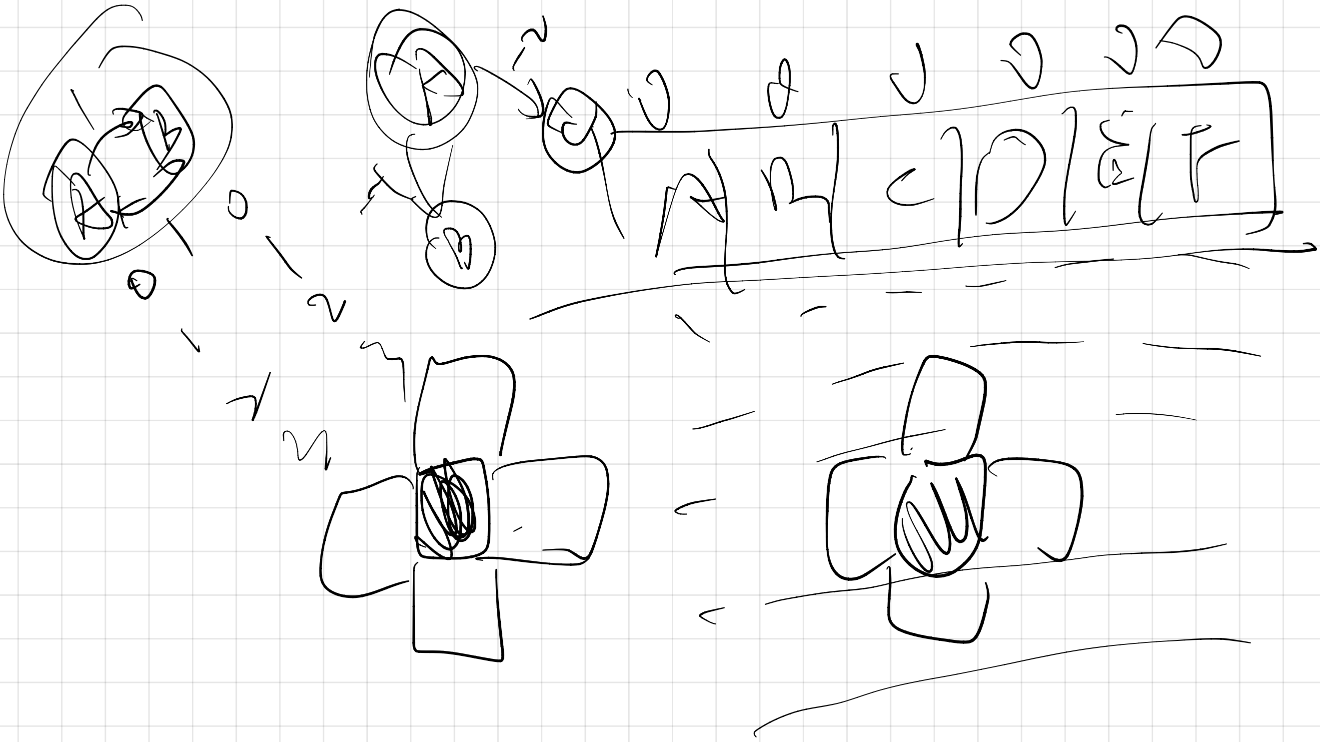
Strategies:

- ε-greedy strategy - every timestep,
  - "explore" w. p. ε (choose action randomly)
  - "exploit" w. p. 1 - ε (follow established policy)
- exploration function strategy - bias toward less explored regions
  - exploration function:

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

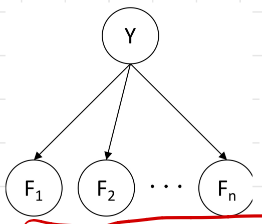
  - new update rule:

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



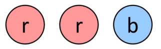
ML

Naive Bayes



$$P(Y|F_{0,0}\dots F_{15,15}) \propto P(Y)\prod_{i,j} P(F_{i,j}|Y)$$

MLE eg!



- **Model:**

X	red	blue
$P_\theta(x)$	θ	$1-\theta$

- **Data**: draw N balls. N_r come up red, N_b come up blue
  - Dataset: $D = \{x_1, \dots, x_n\}$
  - Ball draws are independent and identically distributed (i.i.d.):

$$P(D \mid \theta) = \prod_i P(x_i \mid \theta) = \prod_i P_\theta(x_i) = \theta^{N_r}\cdot(1-\theta)^{N_b}$$

Maximum likelihood estimation: find θ that maximizes $P(D \mid \theta)$

$$\theta = \underset{\theta}{\operatorname{argmax}}\, P(D \mid \theta) = \underset{\theta}{\operatorname{argmax}} \log P(D \mid \theta)$$

Deriving MLEs

- **Maximum likelihood estimation**: find θ that maximizes $P(D \mid \theta)$

$$\theta = \underset{\theta}{\operatorname{argmax}}\, P(D \mid \theta) = \underset{\theta}{\operatorname{argmax}} \log P(D \mid \theta)$$

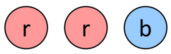
$$\frac{\partial}{\partial\theta}\log P(D \mid \theta) = \frac{\partial}{\partial\theta}[N_r\log(\theta) + N_b\log(1-\theta)]$$
$$= N_r\frac{\partial}{\partial\theta}\log(\theta) + N_b\frac{\partial}{\partial\theta}\log(1-\theta)$$
$$= N_r\frac{1}{\theta} - N_b\frac{1}{1-\theta}$$
$$= 0$$

Multiply by $\theta(1-\theta)$: $\quad N_r(1-\theta) - N_b\theta = 0$

$$N_r - \theta(N_r + N_b) = 0$$

$$\boxed{\hat{\theta} = \frac{N_r}{N_r + N_b}}$$

Smoothing



- Pretend you saw every outcome once more than you actually did

$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x[c(x)+1]} = \frac{c(x)+1}{N+|X|}$$

$$P_{ML}(X) = \left\langle\frac{2}{3}, \frac{1}{3}\right\rangle$$

$$P_{LAP}(X) = \left\langle\frac{3}{5}, \frac{2}{5}\right\rangle$$

Overfitting! increase strength k in Laplas smoothing.

Perception

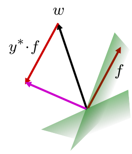
Binary Perceptron

- Start with weights = 0
- For each training instance:
  - Classify with current weights

$$y = \begin{cases} +1 & \text{if } w\cdot f(x) \ge 0 \\ -1 & \text{if } w\cdot f(x) < 0 \end{cases}$$

  - If correct (i.e., y=y*), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^*\cdot f$$



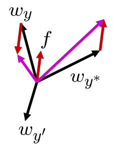
Multiclass Perceptron

- Start with all weights = 0
- Pick up training examples one by one
- Predict with current weights

$$y = \arg\max_y w_y \cdot f(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$



What it the **minimum** number of parameters needed to fully model a joint distribution $P(Y, F_1, F_2, \dots, F_n)$ over label Y and n features F_i? Assume binary class where each feature can possibly take on k distinct values.

$2k^n - 1$ $\quad$ $2k^n$ rows

Naive Bayes assumption

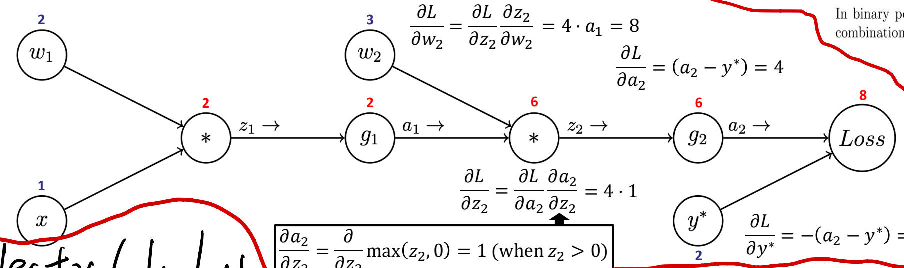
Example: Automatic Differentiation

- Build a computation graph and apply chain rule: $f(x) = g(h(x))$ $\quad f'(x) = h'(x)\cdot g'(h(x))$
- Example: neural network with quadratic loss: $L(a_2, y^*) = \frac{1}{2}(a_2 - y^*)^2$ and ReLU activations $g(z) = \max(0, z)$
- $a_2 = g_2(w_2 * g_1(w_1 * x))$

$$\boxed{\frac{\partial z_2}{\partial w_2} = \frac{\partial}{\partial w_2}(w_2\cdot a_1) = a_1}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial w_2} = 4\cdot a_1 = 8$$

$$\frac{\partial L}{\partial a_2} = (a_2 - y^*) = 4$$



$$\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_2} = 4\cdot 1$$

$$\boxed{\frac{\partial a_2}{\partial z_2} = \frac{\partial}{\partial z_2}\max(z_2, 0) = 1 \text{ (when } z_2 > 0)}$$

$$\frac{\partial L}{\partial y^*} = -(a_2 - y^*) = -4$$

You suspect that you are overfitting with your Naive Bayes with Laplace Smoothing. How would you adjust the strength k in Laplace Smoothing? ↑k

(d) While using Naive Bayes with Laplace Smoothing, increasing the strength k in Laplace Smoothing can:

- [x] Increase training error ?
- [x] Increase validation error ?
- [ ] Decrease training error
- [x] Decrease validation error

It is possible for the perceptron algorithm to never terminate on a dataset that is linearly separable in its feature space. F

If the perceptron algorithm terminates, then it is guaranteed to find a max-margin separating decision boundary. F

In binary perceptron where the initial weight vector is $\vec{0}$, the final weight vector can be written as a linear combination of the training data feature vectors. T

For binary class classification, logistic regression produces a linear decision boundary. T

In the binary classification case, logistic regression is exactly equivalent to a single-layer neural network with a sigmoid activation and the cross-entropy loss function. T

(j) You train a linear classifier on 1,000 training points and discover that the training accuracy is only 50%. Which of the following, if done in isolation, has a good chance of improving your training accuracy?

- [x] Add novel features
- [ ] Train on more data

(k) You now try training a neural network but you find that the training accuracy is still very low. Which of the following, if done in isolation, has a good chance of improving your training accuracy?

- [x] Add more hidden layers
- [x] Add more units to the hidden layers

Vector Calculus

1. Show $\frac{\partial}{\partial\vec{x}}(\vec{x}^T\vec{c}) = \vec{c}^T$

$\frac{\partial}{\partial x_i}(\vec{x}^T\vec{c}) = c_i$ ∴ ✓

2. Show $\frac{\partial}{\partial\vec{x}}||\vec{x}||_2^2 = 2\vec{x}^T$

$\frac{\partial ||x||^2}{\partial x_i} = 2x_i \Rightarrow 2\vec{x}^T$

3. Show $\frac{\partial}{\partial\vec{x}}(A\vec{x}) = A$

$f_i = \sum_k A_{ik}x_k$

$\frac{\partial f_i}{\partial x_j} = A_{ij}$

4. Show $\frac{\partial}{\partial\vec{x}}(\vec{x}^TA\vec{x}) = \vec{x}^T(A + A^T)$

$\Rightarrow (A+A^T)\vec{x}$

$= \vec{x}^T(A+A^T)$

$\frac{\partial f}{\partial x_i} = \sum_{k=1}^{n} A_{ik}x_k + \sum_{k=1}^{n} A_{ki}x_k$

$= \sum_k (A_{ik} + A_{ik}^T)x_k$

矩阵公式

$(A^T)^T = A$

$(A+B)^T = A^T + B^T$

$(AB)^T = B^TA^T$

$\Rightarrow (x^Tx)^T = x^Tx^{TT} = x^Tx$

$\Rightarrow x^TAy = (x^TAy)^T = y^TA^Tx$

$\vec{a}\cdot\vec{b} = \vec{a}^T\vec{b}$

$\nabla_{\vec{w}} f(\theta) = \frac{\partial f}{\partial w}$

$\vec{x}$: col

$\vec{x}^T$: row

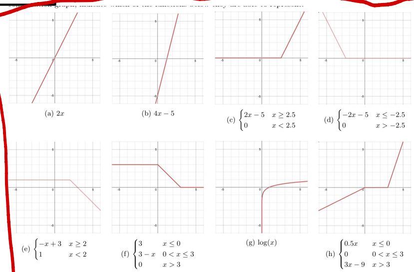
$\frac{\partial \text{scalar}}{\partial \text{vector}} \to \vec{x}^T$ (vector)

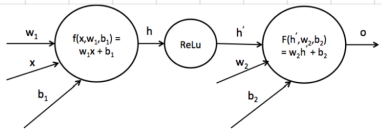
$\frac{\partial \text{vector}}{\partial \text{vector}} \to$ Matrix

$||x||_1 = \sum_{i=1}^m |x_i|$，即向量元素绝对值之和，x 到零点的曼哈顿距离

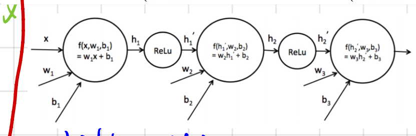
2-范数

$||x||_1 = \sqrt{\sum_{i=1}^m x_i^2}$，2-范数也称为Euclid范数（欧几里得范数，常用计算向量长度），即向量元素绝对值的平方和再开方，表示x到零点的欧式距离

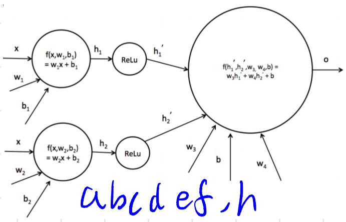




(c)(d)(e)

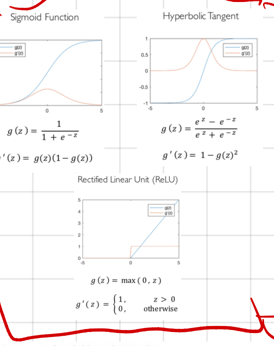


(c)(d)(e)(f)



abcdef, h

$w_i \leftarrow w_i - \alpha\frac{\partial Loss}{\partial w_i}$



梯度下降 momentum: 加过去的梯度

Act functions

(大数据集)

Batch GD → 方差小,稳,慢

Stochastic GD → 方差大,快速
(SGD) 振荡

Neural Network Properties.

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.
- Practical considerations:
  - Must have nonlinear activation function
  - Requires an arbitrarily large number of neurons:
    - Danger for overfitting (hence early stopping!)
    - No guarantee that we can do this on real-world compute
  - Often more efficient in practice to have more layers, less neurons

Logistic Regression

- Maximum likelihood estimation:

$$\max_w \; ll(w) = \max_w \sum_i \log P(y^{(i)}|x^{(i)}; w)$$

with:

$$P(y^{(i)} = +1|x^{(i)}; w) = \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$$

$$P(y^{(i)} = -1|x^{(i)}; w) = 1 - \frac{1}{1+e^{-w\cdot f(x^{(i)})}}$$

Aside: linear regression ≠ logistic regression!