Knowledge representation and reasoning

- First-order logic: syntax and semantics
- Soundness and completeness of proof procedures
- Converting first-order formulas into clausal form
- Unification and MGU
- Resolution proof: forward chaining and refutation
- Answer extraction
- Knowledge graph

Alphabet

Logical symbols (fixed meaning and use):

- Punctuation: (,),,,.
- Connectives and quantifiers: $=, \neg, \wedge, \vee, \forall, \exists$
- Variables: $x, x_1, x_2, \ldots, x', x'', \ldots, y, \ldots, z, \ldots$

Non-logical symbols (domain-dependent meaning and use):

- Predicate (谓词) symbols
  - arity: number of arguments
  - arity 0 predicates: propositional symbols
- Function symbols
  - arity 0 functions: constant symbols

Terms (项)

- Every variable is a term
- If $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term

Formulas

- If $t_1, \ldots, t_n$ are terms and P is a predicate symbol of arity n, then $P(t_1, \ldots, t_n)$ is an atomic formula
- If t_1 and t_2 are terms, then $(t_1 = t_2)$ is an atomic formula
- If α and β are formulas, and v is a variable, then $\neg\alpha, (\alpha \wedge \beta), (\alpha \vee \beta), \exists v.\alpha, \forall v.\alpha$ are formulas

Interpretations

An interpretation (解释) is a pair $M = \langle D, I \rangle$

- D is the domain, can be any non-empty set
- I is a mapping from the set of predicate and function symbols
- If P is a predicate symbol of arity n, $I(P)$ is an n-ary relation over D, i.e., $I(P) \subseteq D^n$
  - If p is a 0-ary predicate symbol, *i.e.*, a propositional symbol, $I(p) \in \{true, false\}$
- If f is a function symbol of arity n, $I(f)$ is an n-ary function over D, *i.e.*, $I(f) : D^n \to D$
  - If c is a 0-ary function symbol, *i.e.*, a constant symbol, $I(c) \in D$

Denotation (指称) of terms

- Terms denote elements of the domain
- A variable assignment μ is a mapping from the set of variables to the domain D
- $\|v\|_{M,\mu} = \mu(v)$
- $\|f(t_1, \ldots, t_n)\|_{M,\mu} = I(f)(\|t_1\|_{M,\mu}, \ldots, \|t_n\|_{M,\mu})$

Satisfaction: atomic formulas

$M, \mu \models \alpha$ is read "M, μ satisfies (满足) α"

- $M, \mu \models P(t_1, \ldots, t_n)$ iff $\langle \|t_1\|_{M,\mu}, \ldots, \|t_n\|_{M,\mu} \rangle \in I(P)$
- $M, \mu \models (t_1 = t_2)$ iff $\|t_1\|_{M,\mu} = \|t_2\|_{M,\mu}$

Satisfaction: propositional connectives

- $M, \mu \models \neg\alpha$ iff $M, \mu \not\models \alpha$
- $M, \mu \models (\alpha \wedge \beta)$ iff $M, \mu \models \alpha$ and $M, \mu \models \beta$
- $M, \mu \models (\alpha \vee \beta)$ iff $M, \mu \models \alpha$ or $M, \mu \models \beta$

Satisfaction: quantifiers

$\mu[v \mapsto d]$ denotes a variable assignment just like μ, except that it maps v to d

- $M, \mu \models \exists v.\alpha$ iff for some $d \in D$, $M, \mu[v \mapsto d] \models \alpha$
- $M, \mu \models \forall v.\alpha$ iff for all $d \in D$, $M, \mu[v \mapsto d] \models \alpha$

Let α be a sentence. Then whether $M, \mu \models \alpha$ is independent of μ.

Thus we simply write $M \models \alpha$

Logical entailment (蕴涵)

- $S \models \alpha$ iff for every M, if $M \models S$ then $M \models \alpha$
- $S \models \alpha$ is read: S entails α or α is a logical consequence (推论) of S
- A special case: $\emptyset \models \alpha$, simply written $\models \alpha$, read "α is valid" (有效的)

Inference (推理) procedure

- We want a mechanical procedure to check if $KB \models \alpha$
- Called an inference procedure
- Sound (合理的) if whenever it says yes, then $KB \models \alpha$
- Complete (完备的) if whenever $KB \models \alpha$, then it says yes

Conversion to Clausal Form： 8 steps

1. Eliminate Implications.
2. Move Negations inwards (and simplify $\neg\neg$).
3. Standardize Variables.
4. Skolemize.
5. Convert to Prenex (前缀) Form.
6. Distribute disjunctions over conjunctions.
7. Flatten nested conjunctions and disjunctions.
8. Convert to Clauses.

Unifiers

- A unifier (合一项) of two formulas f and g is a substitution σ that makes f and g syntactically identical.
- Note that not all formulas can be unified – substitutions only affect variables.
- *e.g.*, $P(f(x), a)$ and $P(y, f(w))$ cannot be unified, as there is no way of making $a = f(w)$ with a substitution.

Computing MGUs

Given two atomic formulas f and g

1. $\sigma = \{\}$; $S = \{f, g\}$
2. If S contains an identical pair of formulas, stop and return σ as the MGU of f and g.
3. Else find the disagreement set $D = \{e_1, e_2\}$ of S
4. If $e_1 = V$ a variable, and $e_2 = t$ a term not containing V (or vice-versa) then let $\sigma = \sigma\{V = t\}$; $S = S\{V = t\}$; Goto 2
5. Else stop, f and g cannot be unified.

Note: to update σ, we must compose σ with $\{V = t\}$.
A common error is to just add $V = t$ to σ.

First-order Resolution

From the two clauses $\{\rho_1\} \cup c_1$ and $\{\neg\rho_2\} \cup c_2$, where there exists a MGU σ for ρ_1 and ρ_2, infer the clause $(c_1 \cup c_2)\sigma$

Theorem. $S \vdash ()$ iff S is unsatisfiable

Answer extraction

- We can also answer wh- questions
- Replace query $\exists x P(x)$ by $\exists x[P(x) \wedge \neg answer(x)]$
- Instead of deriving (), derive any clause containing just the answer predicate

什么是知识图谱

- 知识图谱是一张有向图，图中的节点表示实体或概念，而图中的边则由属性或关系构成
- 知识图谱可以看作是三元组(triple)的集合
  - (实体1-关系-实体2)：中国-首都-北京
  - (实体-属性-属性值)：北京-人口-2069万

Search

- Problem solving by search: formalization
- Uninformed search: Breadth-First, Uniform-Cost, Depth-First, Depth-Limited, and Iterative- Deepening
- Heuristic search: Greedy best-first, A*
- Properties of search: completeness, optimality, time and space complexity
- Path/cycle checking
- Game tree search: MiniMax, alpha-beta pruning
- Simulated annealing and genetic algorithms

The formalism

To formulate a problem as a search problem we need the following components:

1. Formulate a state space over which to search. The state space necessarily involves abstracting the real problem.
2. Formulate actions that allow one to move between different states. The actions are abstractions of actions you could actually perform.
3. Identify the initial state that best represents your current state
4. Identify the goal or desired condition one wants to achieve.

A solution to the problem is a sequence of actions that can transform the initial state into a state where the goal condition holds.

Tree search

- Frontier is the set of states we haven't yet explored/expanded, and want to explore
- Initial call has Frontier = the set of initial state

```
TreeSearch(Frontier, Sucessors, Goal? )
    If Frontier is empty return failure
    Curr = select state from Frontier
    If (Goal?(Curr)) return Curr.
    Frontier' = (Frontier - {Curr}) U Successors(Curr)
    return TreeSearch(Frontier', Successors, Goal?)
```

Summary of uninformed search

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

Note. The table is from the textbook where for BFS, goal test is done when a node is generated, so both time and space complexity is $O(b^d)$ instead of $O(b^{d+1})$.

A* search: Summary

- Define an evaluation function $f(n) = g(n) + h(n)$
- We use $f(n)$ to order the nodes on the frontier.
- $h(n)$ is admissible if for all nodes n, $h(n) \leq h^*(n)$
- $h(n)$ is consistent/monotone if for any nodes n_1 and n_2, $h(n_1) \leq c(n1 \rightarrow n2) + h(n_2)$
- Consistency implies admissibility
- Admissibility implies optimality
- Exponential time and space complexity

Path / cycle checking

- Path checking: when we expand n to obtain child c, ensures that the state c is not equal to the state reached by any ancestor of c along this path
- Cycle checking: keep track of all states previously expanded during the search; when we expand n to obtain child c, ensure that c is not equal to any previously expanded state
- For uniform-cost search, cycle checking preserves optimality
- For A* with monotone heuristics, cycle checking preserves optimality

Summary: Two-Player Zero-Sum Game

- Two players A (Max) and B (Min)
- Set of states S
- An initial state $I \in S$
- Terminal positions $T \subseteq S$
- Successor function
- Utility (效益) function $V : T \to \mathbf{R}$.

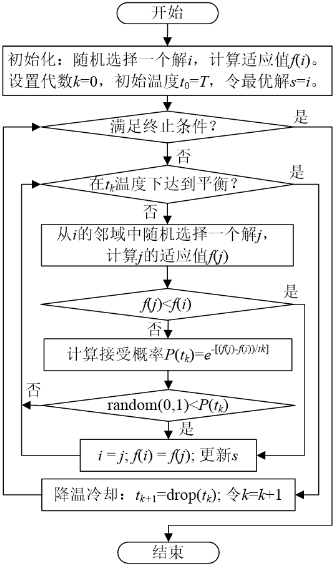
The MiniMax Strategy

- Assume that the other player will always play their best move
- you always play a move that will minimize the payoff that could be gained by the other player.

Alpha beta pruning

- Mark Max nodes with the change of alpha values, and Min nodes with the change of beta values
- Do alpha cut on a Max node whenever the current value $\geq$ the value of an ancestor Min node
- Do beta cut on a Min node whenever the current value $\leq$ the value of an ancestor Max node

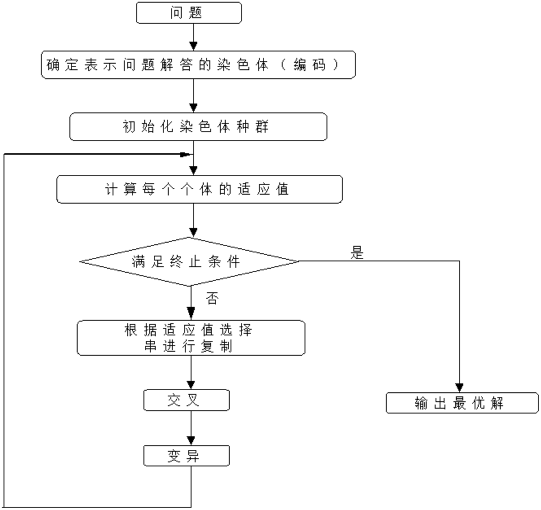
模拟退火基本流程



//功能：模拟退火算法伪代码
//说明：本例以求问题最小值为目标
//参数：T为初始温度; L为内层循环次数

procedure SA
  //Initialization
    Randomly generate a solution X_0, and calculate its fitness value $f(X_0)$;
  $X_{best}=X_0$; $k=0$; $t_k=T$;
  while not stop
    //The search loop under the temperature t_k
    for i=1 to L //The loop times
      Generate a new solution X_{new} based on the current solution X_k, and calculate its fitness value $f(X_{new})$.
      if $f(X_{new}) < f(X_k)$
        $X_k = X_{new}$;
        if $f(X_k) < f(X_{best})$ $X_{best}=X_k$;
        continues;
      end if
      Calculate $P(t_k)=e^{-[(f(Xnew)-f(Xk))/tk]}$;
      if random(0,1) $< P$
        $X_k = X_{new}$;
      end if
    end for
    //Drop down the temperature
    t_{k+1}=drop(t_k); $k=k+1$;
  end while
  print X_{best}
end procedure

遗传算法的一般步骤



问题
确定表示问题解答的染色体（编码）
初始化染色体种群
计算每个个体的适应值
满足终止条件
是
否
根据适应值选择
串进行复制
交叉
变异
输出最优解

常见问题

- 旅行商问题（Traveling Salesman Problem，TSP）：给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路
- 背包问题(Knapsack problem): 给定一组物品，每种物品都有自己的重量和价值，在限定的总重量内，选择物品使得物品的总价值最高。

Planning

- Closed world assumption
- STRIPS representation of actions
- STRIPS planning
- Relaxed plan heuristics

Closed World Assumption (CWA)

- The knowledge base used to represent a state of the world is a list of positive ground atomic facts. (Like a database.)
- Closed World Assumption (CWA) is the assumption that
  - the constants mentioned in KB are all the domain objects.
  - if a ground atomic fact is not in our list of "known" facts, its negation must be true.
- This gives complete information about the state of the system.

STRIPS Actions

- STRIPS represents an action using 3 lists.
  - A list of action preconditions.
  - A list of action add effects.
  - A list of action delete effects.
- These lists contain variables, so that we can represent a whole class of actions with one specification.
- Each ground instantiation of the variables yields a specific action.

Classical planning

- Given
  - a CW-KB representing the initial state,
  - A set of STRIPS operators mapping a state to a new state
  - a goal condition
- Determine a sequence of actions that transforms the initial CW-KB to a CW-KB satisfying the goal

Relaxed problem

- We make the assumption that
  - the precondition of each action is a set of positive facts, and
  - the goal is a set of positive facts.
- Recall that we can obtain heuristics by solving a relaxed version of 8-puzzle in which we relax one of the restrictions:
  - move to adjacent field only
  - move to blank field only
- The idea here is similar: consider what happens if we ignore the delete lists of actions.
- This yields a "relaxed problem" that can produce a useful heuristic estimate.

Reasoning under uncertainty

- Bayesian networks: graphs + tables, inference
- Variable elimination algorithm
- Use D-separation to determine independence

Bayesian Networks

A BN over variables $\{X_1, X_2, \ldots, X_n\}$ consists of:

- a DAG (directed acyclic graph) whose nodes are the variables
- a set of CPTs (conditional probability tables) $Pr(X_i|Par(X_i))$ for each X_i

Inference in Bayes Nets

Given

1) a **Bayes net**

$$\Pr(X_1, X_2,\ldots, X_n) = \Pr(X_n \mid Par(X_n)) * \Pr(X_{n-1} \mid Par(X_{n-1}) * \cdots * \Pr(X_1 \mid Par(X_1))$$

2) some **Evidence**, E

E = {a set of values for some of the variables}

We want to

- **compute the new probability <u>distribution</u>**

$$\Pr(X_k \mid E)$$

That is, we want to figure out

$$\Pr(X_k = d \mid E) \text{ for all } d \in Dom[X_k]$$

The VE Algorithm

Given a Bayes Net with CPTs F, query variable Q, evidence variables **E** (observed to have values e), and remaining variables **Z**. Compute Pr(Q|**E**)

1. Replace each factor $f \in F$ that mentions a variable(s) in **E** with its restriction $f_{\mathbf{E}=e}$ (this might yield a "constant" factor)
2. For each Z_j – in the order given – eliminate $Z_j \in \mathbf{Z}$ as follows:
    1. Let $f_1, f_2, \ldots, f_k$ be the factors in F that include Z_j
    2. Compute new factor $g_j = \sum_{Z_j} f_1 \times f_2 \times \ldots \times f_k$
    3. Remove the factors f_i from F and add new factor g_j to F
3. The remaining factors refer only to the query variable Q. Take their product and normalize to produce Pr(Q|**E**).

D-separation

- A set of variables E d-separates X and Y if it blocks every undirected path in the BN between X and Y.
- If evidence E d-separates X and Y, then X and Y are conditionally independent given evidence E
- So what is blocking?

Let P be an **undirected path** from X to Y in a BN.

Let **E** be a set of variables.

We say ***E blocks path P*** iff there is some node Z on the path such that:

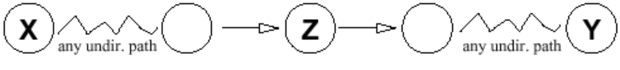
Case 1: one arc on P ***goes into*** Z and one ***goes out*** of Z, and Z∈**E**; or

Case 2: both arcs on P leave Z, and Z∈**E**; or

Case 3: both arcs on P enter Z and *neither Z, nor any of its descendants*, are in **E**.

Blocking: Graphical View

(1)

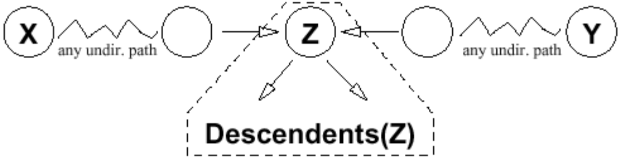


If Z in evidence, the path between X and Y blocked

(2)

If Z in evidence, the path between X and Y blocked

(3)



If Z is ***not*** in evidence and ***no*** descendent of Z is in evidence, then the path between X and Y is blocked

Machine learning

- Decision-tree learning
- Naive Bayes and Bayes network parameter learning
- K-means and EM
- Chain rule for computing partial derivatives
- Linear and logistic regression
- Backpropagation
- Reinforcement learning: Value iteration, Q-learning and SARSA
- CNN

Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: choose "most significant" attribute as root of (sub)tree

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

  if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
  else
    $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
    tree ← a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
      subtree ← DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
      add a branch to *tree* with label $(A = v_k)$ and subtree *subtree*
    return *tree*

Plurality-value(examples) returns the majority classification of the examples

Entropy

- The entropy of a random variable V with values v_k, each with probability $P(v_k)$:

$$H(V) = -\sum_k P(v_k) \log_2 P(V_k)$$

- The entropy of a Boolean random variable that is true with probability q:

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

- If a training set contains p positive examples and n negative examples, then the entropy of the goal attribute on the whole set is

$$H(Goal) = B(\frac{p}{p+n})$$

Information gain

- An attribute A with d distinct values divides the training set E into subsets $E_1, \ldots, E_d$.
- Each subset E_k has p_k positive examples and n_k negative examples,
- So the expected entropy remaining after testing attribute A is

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} B(\frac{p_k}{p_k + n_k}).$$

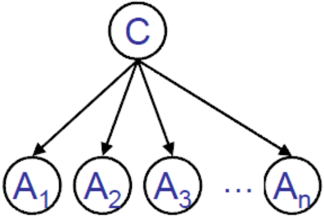
- The information gain (IG) from the attribute test on A is the expected reduction in entropy:

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

- Choose the attribute with the largest IG

Naive Bayes models

- Want to predict a class C based on attributes $A_1, \ldots, A_n$
- Parameters:
  - $\theta = P(C = true)$
  - $\theta_{i1} = P(A_i = true | C = true)$
  - $\theta_{i2} = P(A_i = true | C = false)$
- Assumption: A_i's are independent given C



Naive Bayes learning

- Notation: $p = \#(c), n = \#(-c), p_i^+ = \#(c, a_i)$, $n_i^+ = \#(c, -a_i), p_i^- = \#(-c, a_i), n_i^- = \#(-c, -a_i)$
- $P(d|h) = \theta^p (1-\theta)^n \Pi_i \theta_{i1}^{p_i^+} \theta_{i2}^{p_i^-} (1-\theta_{i1})^{n_i^+} (1-\theta_{i2})^{n_i^-}$
- $\theta = p/(p+n)$, $\theta_{i1} = p_i^+/(p_i^+ + n_i^+)$, $\theta_{i2} = p_i^-/(p_i^- + n_i^-)$,
- $P(C|a_1, \ldots, a_n) = \alpha P(C) \Pi_i P(a_i|C)$
- Choose the most likely class

Bayesian network parameter learning (ML)

- Parameters $\theta_{V,pa(V)=\mathbf{v}}$:
  - CPTs: $\theta_{V,pa(V)=\mathbf{v}} = P(V|pa(V)=\mathbf{v})$
- Data **d**:
  - d_1 : $\langle V_1=v_{1,1}, V_2=v_{2,1}, \ldots, V_n = v_{n,1}\rangle$
  - d_2 : $\langle V_1=v_{1,2}, V_2=v_{2,2}, \ldots, V_n = v_{n,2}\rangle$
  - ...
- Maximum likelihood:
  - Set $\theta_{V,pa(V)=\mathbf{v}}$ to the relative frequencies of the values of V given the values **v** of the parents of V

$$\theta_{V,pa(V)=\mathbf{v}} = \#(V,pa(V)=\mathbf{v}) \,/\, \#(pa(V)=\mathbf{v})$$

k均值算法

用于硬聚类

1. 选择k个重心（centroid）。
2. 寻找最近的重心并且更新聚类分配。将每个数据点都分配给离它最近的重心的聚类。距离的度量通常是欧式距离。
3. 将重心移动到它们的聚类的中心。每个聚类的重心的新位置是通过计算该聚类中所有数据点的平均位置得到的。
4. 重复第2和3步，直到每次迭代时重心的位置不再显著变化（即直到该算法收敛）。

高斯混合模型

- 高斯混合模型的参数是:
  - $w_i = P(C = i)$ (每个成分的权重),
  - μ_i (每个成分的均值),
  - Σ_i (每个成分的协方差).

EM算法

随机初始化模型参数，重复以下两步直到收敛:

1. E步
   - 计算数据$\mathbf{x}_j$是由成分i生成的概率

     $$p_{ij} = P(C = i|\mathbf{x}_j) = \alpha P(\mathbf{x}_j|C = i)P(C = i),$$

     其中$P(\mathbf{x}_j|C = i)$是第i个高斯分布，$P(C = i) = w_i$
   - 令$n_i = \sum_j p_{ij}$，即当前分配到成分i的数据点的期望数量
2. M步：计算新的均值，协方差，和权重

$$\boldsymbol{\mu}_i \leftarrow \sum_j p_{ij}\mathbf{x}_j/n_i$$

$$\boldsymbol{\Sigma}_i \leftarrow \sum_j p_{ij}(\mathbf{x}_j - \boldsymbol{\mu}_i)(\mathbf{x}_j - \boldsymbol{\mu}_i)^\top/n_i$$

$$w_i \leftarrow n_i/N$$

其中 N 是数据点的总数量.

Linear regression

- $h_w(x) = w \cdot x = \sum_i w_i x_i$
- Squared error loss: $Loss(w) = (y - h_w(x))^2$
- Chain rule: $\partial g(f(x))/\partial x = g'(f(x))\partial f(x)/\partial x$
- $\partial Loss(w)/\partial w_i = -2(y - h_w(x))x_i$
- $w_i \leftarrow w_i + \alpha(y - h_w(x))x_i$

Logistic regression

- A logistic function is the sigmoid of a linear function
- Logistic regression: regression with logistic functions
- $g(x) = 1/(1 + e^{-x})$
- $h_w(x) = g(w \cdot x)$
- $g' = g(1 - g)$
- $Loss(w) = (y - h_w(x))^2$
- $\partial Loss(w)/\partial w_i = -2(y - h_w(x))g'(w \cdot x)x_i$
  $= -2(y - h_w(x))h_w(x)(1 - h_w(x))x_i$
- $w_i \leftarrow w_i + \alpha(y - h_w(x))h_w(x)(1 - h_w(x))x_i$

The algorithm

initialize w arbitrarily
repeat
  for each e in examples do
    $p \leftarrow g(w \cdot x(e))$
    $\delta \leftarrow y(e) - p$
    for each i do
      $w_i \leftarrow w_i + \alpha\delta p(1-p)x_i$
until some stopping criterion is satisfied
return w

Forward and backward phases for backpropagation

Forward phase:

- Propagate inputs forward to compute the output of each unit
- Output a_j at unit j: $a_j = g(in_j)$ where $in_j = \sum_i w_{ij} a_i$

Backward phase:

- Propagate errors backward
- For an output unit j:
  $\Delta_j = g'(in_j)(y_j - a_j) = a_j(1 - a_j)(y_j - a_j)$
- For an hidden unit i:
  $\Delta_i = g'(in_i) \sum_j w_{ij} \Delta_j = a_i(1 - a_i) \sum_j w_{ij} \Delta_j$

Weight updating: $w_{ij} \leftarrow w_{ij} + \alpha a_i \Delta_j$

Markov Decision Processes

An MDP consists of:

- set S of states.
- set A of actions.
- $P(s'|s,a)$ specifies the probability of transitioning to state s' given that the agent is in state s and does action a.
- $R(s,a,s')$ is the expected reward received when the agent is in state s, does action a and ends up in state s'.
- $0 \leq \gamma \leq 1$ is discount factor.

Formulation of reinforcement planning

- A policy $\pi : S \times A \to [0, 1]$, where $\pi(s, a)$ is the probability of taking action a in state s
- Value function $V : S \to \mathbb{R}$, where $V_\pi(s) = \mathbb{E}_\pi[G|S = s]$, meaning the expected discount award of following π in state s
- Given an environment MDP (S, A, P, R, γ), where P and R are not known to the agent, find a policy π that maximizes $V_\pi(s_0)$

Q-learning

initialize $Q[S, A]$ arbitrarily
observe current state s
repeat forever:
    select and carry out an action a
    observe reward r and state s'
    $Q[s, a] \leftarrow Q[s, a] + \alpha\,(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
    $s \leftarrow s'$

SARSA (state-action-reward-state-action)

initialize $Q[S, A]$ arbitrarily
observe current state s
select action a using a policy based on Q
repeat forever:
    carry out action a
    observe reward r and state s'
    select action a' using a policy based on Q
    $Q[s, a] \leftarrow Q[s, a] + \alpha \left(r + \gamma Q[s', a'] - Q[s, a]\right)$
    $s \leftarrow s'$
    $a \leftarrow a'$

Q值更新公式的汇总

- 值迭代: 已知 $P(s'|s,a)$ 和 $R(s,a,s')$

$$Q[s,a] \leftarrow \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma \max_{a'} Q[s',a'])$$

- Q-learning: 使用经验 $\langle s,a,r,s' \rangle$

$$Q[s,a] \leftarrow Q[s,a] + \alpha \left(r + \gamma \max_{a'} Q[s',a'] - Q[s,a] \right)$$

- Sarsa: 使用经验 $\langle s,a,r,s',a' \rangle$

$$Q[s,a] \leftarrow Q[s,a] + \alpha \left(r + \gamma Q[s',a'] - Q[s,a] \right)$$

CNN: basic ideas

- Local connectivity: connect each neuron to only a local region of the previous layer
- Parameter sharing: weights shared across multiple local regions
- Downsampling to make images smaller

Convolutional Layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- In the output volume, the d-th depth slice is the result of performing a convolution of the d-th filter over the input volume, and then offset by d-th bias.

Pooling layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D2 = D1$
- For Pooling layers, it is not common to pad the input using zero-padding.

There are only two commonly seen variations of the max pooling layer found in practice: A pooling layer with F=3,S=2 (also called overlapping pooling), and more commonly F=2,S=2.