Lecture notes by Edward Loper

Course: CIS 620 (Advanced Topics in AI)
Professors: Michael Kearns and Lawrence Saul
Institution: University of Pennsylvania

http://www.cis.upenn.edu/~mkearns/teaching/cis620/cis620.html

Wednesday, January 9, 2002

1 Markov Decision Process

- State space S
- Actions a, b, ... from each state
- Transition probabilities P(·|s,a), P(s'|s,a)
- (local) Rewards R(s,a) ∈ [0,1]
- Discounted reward: $r_0 + \gamma\, r_1 + \gamma^2 r_2 + \ldots$ (c.f. finite horizon)
- Policy Π: states → actions
- Planning: given MDP, compute (near) optimal policy

We are assuming full observability, i.e., we always know what state we're currently in.

(Pset 1: available later today/tonight.)

1.1 Value Functions

- Π = Policy
- Define value function $V^{\Pi}(s)$: $V^{\Pi}(s)$ = expected discounted return starting from s and using Π.
- $V^{\Pi}(s) = R(s,\pi(s)) + \gamma(\sum_{s'} P(s'|s,\Pi(s))V^{\Pi}(s))$
- Now, define $V^*(s)$ to be the optimal exected discouted return from s. I.e., $V^*(s) = \max_{\Pi}\{V^{\Pi}(s)\}$

1.2 The planning problem

- $\exists\ \Pi^*$ s.t. $V^{\Pi^*}(s) = V^*(s)$ Argument: if there were a better choice for one time we arrive at V(s), then we should always do the same thing there..

Planning problem: find Π^* (= optimal plan)

- You know the transition probabilities
- You know the rewards
- You want to find the policy

Bellman optimality:

- $V^*(s) = \max\{\ R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')\ \}$
- Payoff has 2 components: immediate, and discounted.
- $\Pi^*(s) = \text{argmax}_a\{\ R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')\ \}$
  - Greedy(V^*)

1.3 Finding the optimal policy

Try to find a value that makes the bellman optimality equation true:

- $V^*(s) = \max_a\ \{\ R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s')\ \}$

Value Iteration

Iterate v, an estimate of $V^*(s)$:

- $v(s) \leftarrow \max_a\ \{\ R(s,a) + \gamma \sum_{s'} P(s'|s,a)v(s')\ \}$
- If v converges, we're done.

v does converge; why?

- Define error $\Delta_t = \max_s\{ |v_t(s) - V^*(s)| \}$
- $\Delta_{t+1} = \max_s\{ |v_{t+1}(s) - V^*(s)| \}$
- $\Delta_{t+1} = \max_s\{ |\max_a \{ R(s,a) + \gamma \sum_{s'} P(s'|s,a)v_t(s') \} - \max_a \{ R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s') \}| \}$

Note that:

- $|\max\{f(a)\} - \max\{g(a)\}| \leq |\max_a\{f(a) - g(a)\}|$

So:

- $\Delta_{t+1} \leq \max_s\{ |\max_a \{ R(s,a) + \gamma \sum_{s'} P(s'|s,a)v_t(s') - R(s,a) + \gamma \sum_{s'} P(s'|s,a)V^*(s') \}| \}$
- $\Delta_{t+1} \leq \max_s\{ |\max_a \{ \gamma \sum_{s'} P(s'|s,a)v_t(s') - \gamma \sum_{s'} P(s'|s,a)V^*(s') \}| \}$
- $\Delta_{t+1} \leq \max_s\{ |\max_a \{ \gamma \sum_{s'} P(s'|s,a)(v_t(s')-V^*(s')) \}| \}$

Δ_{t+1} is greatest where:

- $v_t(s')-V^*(s') = \Delta_t$

Since we're taking the expectation over a pdf, we can do:

- $\Delta_{t+1} \leq \max_{s,a,s'}\{ |v_t(s') - V^*(s')| \}$

So:

- $\Delta_{t+1} \leq \gamma\Delta_t$

So if we make $t \geq \log(1/\epsilon)/\log(1/\gamma)$, then we converge to within ϵ.

Our optimal policy π = greedy(v)

If we use a value function that's close to optimal, then is the resulting policy greedy(v) close to optimal?

n.b., v and V^π are not necessarily the same..

The finite horizon nature of the problem is really what allows us to put bounds on how close we will be.. our estimation error is limited..

Time: $O(tn^2)$ for t iterations, n states.

1.4 Policy Evaluation

Given Π, we want to compute $V^\Pi(s)$.

- $V^\Pi(s) = R(s,\Pi(s)) + \gamma \sum_{s'} P(s'|s,a)V^\Pi(s')$

This is basically just n linear equations, in n variables (variables are $v_s = V^\Pi(s)$)

$v_s = R(s,\Pi(s)) + \gamma \sum_{s'} P(s'|s,a)v_{s'}$

Solving n linear equations: $O(n^3)$ time

But it's exact (not approximate): we can exactly evaluate a given particular policy in n^3 time..

1.5 Notation..

- $V^\Pi(s)$ = expected return
- $Q^\Pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a)v^\Pi(s')$ I.e., if we ignore policy Π for one step, and do a instead; and from then on, we use policy Π, then what expected discounted return do we get?

1.6 Policy Iteration

- Iterate over a policy π_t
- $\pi_{t+1}(s) = \text{argmax}_a \{ Q^{\pi\ t}(s,a) \}$

- $\pi_{t+1}(s) = \text{argmax}_a \{ R(s,a) + \gamma \sum_{s'} P(s'|s,a) v^{\{\pi_t\}}(s') \}$

How does this differ from value iteration? First, there's no approximation. I.e., we compute Q values directly.

Convergence

- whenever $\text{argmax}_a\{Q^{\pi\ t}(s,a)\} \neq \pi_t(s)$, we change our policy
- whenever this happens, it's a good thing (i.e., increases our value function)
- $\exists$ finite number of policies
- we change whenever we're suboptimal
- i.e., policy iteration will always converge to the optimal solution in a finite limited time.

Notation...

- $P(s', \pi, t)$ = probability that we're in state s' after t iterations, according to policy π.
- In particular:
  - $P(s', \pi, 1) = P(s'|s,\pi(s))$
  - $P(s'|s,\pi,0) = \{1 \text{ iff } s'=s\}$

Policy iteration & value iteration are relatable in a precise way

2 Probability Theory

We want to know: if we've been to state s, and action a has led to some distribution of results.. then what does probability theory tell us about what we expect?

2.1 Characterizing a random variable

Moments

A random variable x has moments:

$$E[x^n] = \int p(x)x^n dx$$

We can write this very compactly by refering to the generating function. Generating function:

$$\begin{aligned}\{f\}(k) &\triangleq E[e^{kx}] \\ &= E[\sum((k^n x^n)/n!)] \\ &= \sum((k^n/n!)E[x^n])\end{aligned}$$

We can "generate" moments from the generating function by differentiating.

$$\partial/\partial\ x^n[\{f\}(x)]_{k=0} = E[x^n] = \int p(x)x^n dx$$

This can be signifigantly easier than differentiating

Example: Bernoulli distribution

$$x \in \{0,1\}$$

$$p_x = \begin{cases} 1-\mu & \text{if } x=0 \\ \mu & \text{if } x=1 \end{cases}$$

$$\begin{aligned}\{f\}(k) &= E[e^{kx}] \\ &= (1-\mu) + \mu\ e^k\end{aligned}$$

Example: Gaussian

$$p(x) = 1/\text{root}(s\pi\sigma^2) * e^{-(x-\mu)^2}/2\sigma^2$$

$$\begin{aligned}\mu &= E[x] \\ \sigma^2 &= E[x^2] - E[x]^2\end{aligned}$$

$$\{f\}(k) = E[e^{kx}]$$

2.2 KL Distance

A measure of distance between two probability distributions. (KL divergence?)

Continuous:

$$KL(p,q) \triangleq \int dx\ p(x)\ln(p(x)/q(x))$$

Discrete:

$$KL(p,q) \triangleq \sum p(x)\ln(p(x)/q(x))$$

KL divergence is always non-negative.

Using the following lower bound on ln:

$$\ln(z) \geq 1-1/z$$

we can show that KL divergence is non-negative:

$$\begin{aligned} KL(p,q) &\geq \sum p(x)(1-(q(x)/p(x)) \\ &= \sum (p(x)-q(x)) \\ &= \sum p(x) - \sum q(x) \\ &= 0 \end{aligned}$$

KL divergence is asymmetric:

$$KL(p,q) \neq KL(q,p)$$

Sefine step function:

$$\theta(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ if } z < 0 \end{cases}$$

Upper bound:

$$\theta(z) \leq e^{kz} \text{ for all } k>0$$

"Independantly identically distributed" = IID

Let $x_1, \ldots, x_n$ be IID with mean μ=E[x]

2.3 Central Limit Theorem (CLT)

- Large deviation theorem characterizes the tails of the distributions.
- What about the center of the distribution?

Let $x_1, x_2, \ldots, x_n$ be IID, with a mean μ=E[x] and a varaiance σ^2=E[x^2]-E[x]^2. Variance is bounded ($\sigma^2<\infty$). Let:

$$y \triangleq 1/\sqrt{n} \sum(x_i-\mu)$$

Then the distribution of this sum converges to a gaussian in the limit that n becomes very large:

$$\lim_{n\to\infty} p(y) = 1/\sqrt{s\pi\sigma^2)}\ e^{-y^2/2\sigma^2}$$

Note: convergence is rapid for the center of the distribution, but actually fairly slow for the tails.

2.4 Rollout

Value iteration and policy iteration are special cases of rollout:

$$
\begin{aligned}
V\{\hat{}\}_{t+1}(s) \leftarrow\ & R(s, \pi\{\hat{}\}_t(s)) + \\
& \gamma \sum P(s'|s, \pi\{\hat{}\}_t, 1)R(s', \pi\{\hat{}\}_t(s')) + \\
& \gamma^2 \sum P(s'|s, \pi\{\hat{}\}_t, 2)R(s', \pi\{\hat{}\}_t(s')) + \\
& \ldots + \\
& \gamma^k \sum P(s'|s, \pi\{\hat{}\}_t, k)R(s', \pi\{\hat{}\}_t(s')) +
\end{aligned}
$$

3 The learning problem

What if we only have the ability to sample the environment?

- not given the P(·|s,a) or R(s,a)

→ find them by sampling. how to sample?

M = generic name for a markov decision process (MVP)
M = ⟨ S, A, P(·|s,a) R(s,a)⟩

All of these assume full observability (we always know what state we're in) Also, we know S and A.

3.1 Generative models (aka simulators)

$$
(s,a) \rightarrow \begin{bmatrix} \\ G_M \\ \\ \end{bmatrix} \begin{matrix} \rightarrow S' \text{ distributed as } P(\cdot|s,a) \\ \\ \rightarrow R = R(s,a) \end{matrix}
$$

Strong model of information access: we can sample any place in the markov model. Queries don't interact (don't have to worry about the "risk" of taking an unreversable action).

Simple strategy: just sample each (s,a) enough to estimate the parameters.

Surprisingly, this is often available

- e.g., in a simulation, we can re-run the same situation repeatedly. (e.g., computer backgammon: we can always play one move from any board configuration, and see what happens).

Reasonable goal for the generative model: find the optimal policy

Policy evaluation with generative models

- One approach: sample extensively, then use planning approach
- Better approaches?

Given policy π, learn $V^\pi(s)$

$$
\begin{aligned}
V^\Pi(s) &= R(s,\pi(s)) + \gamma(\textstyle\sum_{s'} P(s'|s,\Pi(s'))V^\Pi(s)) \\
V^\Pi(s) &= R(s,\pi(s)) + \gamma(\textstyle\sum_{s'} P(s'|s,\Pi(s'),1)R(s,\pi(s)) \\
&\qquad + \gamma^2(\textstyle\sum_{s''} P(s''|s,\Pi(s''),2))V^\pi(s'')
\end{aligned}
$$

This isn't too useful for planning (we can just use the simple equation); but for learning, we have easier access to these equations.

Define τ to be an infinite sequence of ⟨ s,r⟩ values (a trajectory throught the markov model) produced by the policy

τ = $\langle s_0, r_0 \rangle \rightarrow \langle s_1, r_1 \rangle \rightarrow \langle s_2, r_2 \rangle \rightarrow \ldots$

Monte carlo algorithm: sample s/r...

TD(K) = Temporal difference

- K = horizon
- α = learning rate

Alternative algorithm to find $V^\pi\{\hat{}\}$ (estimate of V^π)

$$V^\pi\{\hat{}\}_{t+1}(s_0) \leftarrow (1-\alpha)V^\pi\{\hat{}\}_t(s_0) + \alpha\,[r_0 + \gamma\, r_1 + \ldots + \gamma^{k-1} r_{k-1} + \gamma^k\, V^\pi\{\hat{}\}_t(s_k)$$

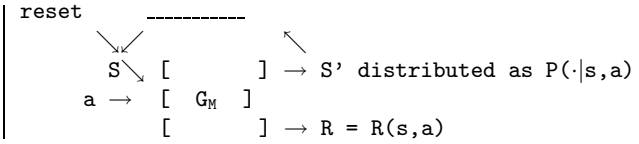
Usually, we just set α=1/t

- this transforms things into a running average

"bais-variance" tradeoff for k:

- if k is large, we get error from increased randomness (sampling larger groups of random variables)
- if k is small, $V^\pi\{\hat{}\}$ gives us error from bias

3.2 Episodic learning (reset-to-start)



Designated start state.

We can always reset to the start state; but we can't reset to just any state..

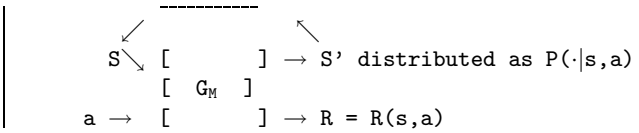
Some parts of the state space may be arbitrarily difficult to get to. (We may not care about those areas; they don't affect optimal actions from other states)

(This can be generalized to a distribution of start states)

Now we have to worry about consequences, but we don't have an exploration/exploitation tradeoff.

3.3 Full learning (No-reset model)

No reset at all.



Exploration/exploitation tradeoff

What is reasonable to expect from the learning algorithm?

4 TD(k) and Policy Evaluation

4.1 Policy Evaluation

- Policy evaluation-planning → linear equations in V^π, so we can just solve for it.
- Phased TD(k) algorithm → in the case of planning, we might as well stick with k=1; but for run-time learning, we may want to use phased TD(k) with k>1.. (assume generative model: we can reset to any state)

$$V\{\hat{}\}^\pi_{t+1}(s) \leftarrow 1/n \sum_{i=1}^{n} [r_0^i + \gamma\, r_1^i + \ldots + \gamma^{k-1} r_{k-1}^i + \gamma^k V\{\hat{}\}^p i_t(s^i_k)]$$

Here, the "i"s are superscripts in r_0^i, etc.

Bias-variance tradeoff:

- if we trust V more, use smaller k: minimize error from variance of markov process.
- if we trust V less, use larger k: minimize error from bias of V

As long as k<∞, we'll never actually converge..

Q: why not just use k=∞, do it once, and ignore V? Because we might have a good a priori estimate of V (e.g., if we're doing policy iteration, then presumably V doesn't change that much between iterations).

$$V^\pi(s) = E[r_0 + \gamma\, r_1 + \ldots + \gamma^k V^\pi(s_k)]$$
$$V^\pi(s) = E[r_0] + \gamma\, E[r_1] + \ldots + \gamma^k E[V^\pi(s_k)]$$

The samples we are considering are r_t^i:

$$r_0^1 + \gamma\, r_1^1 + \ldots + \gamma^k V^\pi(s_k^1)$$
$$r_0^2 + \gamma\, r_1^2 + \ldots + \gamma^k V^\pi(s_k^2)$$

Basically, we're averaging each of these columns. One potential problem: the colums don't represent independant random variables: if we reached an ulikely set of states at t=1, then we probably reached an unlikely set of states at t=2.

We will show uniform convergance: despite the dependancies, we'll get a good estimate.

We'll consider a single column.. In particular, the set of rewards r^i we get on a given time step in trial i of walking the mdp.

Lemma 1

Let r^1, r^2, r^3, ..., $r^n \in [0,1]$ iid random variables drawn according to some distribution. $E[r^i] = \mu$. Then:

$$P[|1/n \sum [r^i] - \mu| \geq \epsilon] \leq e^{-\epsilon^2 n/3}$$

(this is a "large deviation bound")

I.e., the probability that the sampled mean is off from the true mean by more than ϵ is $e^{-\epsilon^2 n/3}$.

So we want $n = C/\epsilon^2$.

Lemma 2

But the thing we really want to bound is:

$$P[|1/n \sum [r_j^i] - E[r_j]| \geq \epsilon \text{ for any } j]$$

How do we do this? (Esp since they're not independant). Well, to be conservative, we know the "union bound":

$$P(a \vee b) \leq P(a)+P(b)$$

So now we've bounded the probability of each.. so we can immediately say:

$$P[|1/n \sum[r_j{}^i] - E[r_j]| \geq \epsilon \text{ for any } j] \leq k e^{-\epsilon^2 n/3}$$
$$\leq e^{\log(k)-\epsilon^2 n/3}$$

Entropy vs energy interpretation: interpret "ϵ^2n/3" as energy term, "log(k)" as entropy term..

Introduce a confidence parameter ∂ (upper bound on "failure probability"):

$$\partial \geq k e^{-\epsilon^2 n/3}$$
$$\epsilon = \text{sqrt}(3\log(k/\partial)/n)$$

So with probability $\geq$ (1-∂) over phase t data, all the reward averages are simultaneously within this ϵ of their expectation.

Error bound on policy evaluation

Define:

$$\Delta_t \triangleq \max_s\{|V\{\hat{}\}^\pi{}_t(s) - V^\pi(s)|\}$$

Then with probability $\geq$ 1-∂:

$$\partial_{t+1} \geq ((1-\gamma^k)/(1-\text{gamma})) * \epsilon + \gamma^k \Delta_t$$
$$\partial_{t+1} \geq ((1-\gamma^k)/(1-\text{gamma})) * \text{sqrt}(3*\log(k/e)/n) + \gamma^k \Delta_t$$

This comes from:

$$(1-\gamma^k)/(1-\text{gamma}) = 1 + \gamma + \gamma^2 + \ldots + \gamma^{k-1}$$

The "variance" part is:

$$((1-\gamma^k)/(1-\text{gamma})) * \text{sqrt}(3*\log(k/e)/n)$$

The "bias" part is:

$$\gamma^k \Delta_t$$

When k is very large:

$$\Delta_{t+1} \leq \text{sqrt}(3*\log(k/\partial)/n)/(1-\partial)$$

When k=1:

$$\Delta_{t+1} \leq \text{sqrt}(3*\log(1/\partial)/n) + \gamma\Delta_t$$

We can then use the union bound over iteration phases, to limit our overall error. Then we can solve Δ_t..

$$\Delta_t \leq \epsilon(1-\gamma^{(kt)})/(1\gamma) + \gamma^{kt}\Delta_0$$
$$\Delta_t \leq \epsilon(1-\gamma^{(kt)})/(1\gamma) + \gamma^{kt}/(1-\gamma)$$

As t->∞:

$$\Delta_t \leq \epsilon(1/1-\gamma)$$
$$\Delta_t \leq (1/1-\gamma)*\text{sqrt}(3*\log(k/\partial)/n)$$

So our asymptotic error is nonzero. The residual asymptotic error is:

$$\Delta_t \leq (1/1-\gamma)*\text{sqrt}(3*\log(k/\partial)/n)$$

So smaller k gives smaller residual error.

But the convergence is really controlled by the γ^{kt} term, so the larger k is, the faster we'll converge.

So decrease k as we go?

The algorithm "TD(λ)" has a continuous paramater λ (instead of k) that basically allows you to do a weighted average the resut of using k=1, k=2, ... The extremes (0,1) correspond to k=1 and k=∞.

5 Learning Optimal Policies

5.1 Q-Learning

Value iteration:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$
$$= R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_b \{Q^*(s',b)\}$$

Use this formulation to produce a learning algorithm, q-learning: if we experience the transition [s{–}a/r→ s'] (s to s' with action a and reward r):

$$\hat{Q}_{t+1}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha[r + \gamma \max_b Q(s',b)]$$

How much data do we need for this estimate to converge to Q^*? (as a funciton of the number of states in the environment)

Monday, February 4, 2002

6 Overview

Where we're going..

- We need compact representations to encode a complex world
  - mdp isn't compact enough – exponential blowup of independant inputs.
  - how can we discover structure in the world?
  - clusters (e.g., all states such that f(s) is true)
  - embedding (compression?)
  - factorization: what is independant?
  - use probablistic models to learn structure
  - bayes nets
  - aka probablistic graphical models
  - provide a unified framework for encoding structure
    - introduce examples over the next few weeks; then show the general framework that they fit in.

this week:

- predictions based on conditional probabilities
- conditional probabilities (CPs) for (few) discrete random variables can be tabulated
- consider continuous random variables (we can't tabulate). conditional probabilities must be paramaterized. natural parametric forms?
- simplest examples: linear & logistic regression

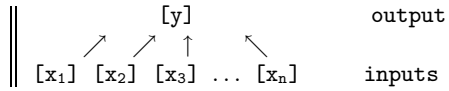
outline for this week:

- linear programming
- matlab
- logistic regression
- gradient ascent
- newton's method

7 Linear Regression

Predictions based on continuous variables.

Goal: predict a continuous output $y \in \mathbb{R}$ from one or more inputs $X=(x_1,x_2,\ldots,x_n) \in \mathbb{R}^d$.



Model this input/output by a noisy linear map. I.e., our model of predicting y is:

- $y = \sum_i w_i x_i$ + noise

Abbreviate sum as inner product:

- $y = W \cdot X$

7.1 Noise

Model noise by a conditional gaussian distribution.

- $P(y|X) = 1/\text{sqrt}(2\pi)\ e^{-1/2(y-W\cdot X)^2}$
- $E(y|X) = W\cdot X$

Can we use this model to learn to predict the future from the past?

If examples are iid from the joint distribution p(X,y), then what is the probability of the data we saw under this model?

- $P(y_1,y_2,\ldots,y_n|X_1,X_2,\ldots,X_n)$

Since we're assuming examples are iid, we can reduce this to:

- $P(y_1|X_1)P(y_2|X_2)\ldots P(y_n|X_n)$

Learning problem: estimate the parameters of the distribution (=W)

Maximize the likelihood:

$$\text{argmax}[W]\ P(y_1,y_2,\ldots,y_n|X_1,X_2,\ldots,X_n)$$

to make it easier, maximize the log likelihood:

$$\begin{aligned}&\text{argmax}[W]\ \textstyle\sum \log P(y_1|X_1)\\ &= \text{argmax}[W]\ -\textstyle\sum[1/2\ \log(2\pi) + 1/2(y-W\cdot X_n)^2]\\ &= \text{argmax}[W]\ -\textstyle\sum 1/2(y-W\cdot X_n)^2\\ &= \text{argmax}[W]\ -\textstyle\sum(y-W\cdot X_n)^2\end{aligned}$$

So it's the same as minimizing the sum of squared error

7.2 How do we maximize a function of several variables?

To find a maximum, use the gradient, and set gradient=0. L(w) = likelihood

$$\begin{aligned}&L(w) = -\textstyle\sum[1/2\ \log(2\pi) + 1/2(y-W\cdot X_n)^2]\\ &\partial\ L/\partial\ w_\alpha = -\textstyle\sum(y-W\cdot X_n)(-x_{\alpha\ n})\end{aligned}$$

So for all α, we should have:

$$\partial\ L/\partial\ w_\alpha = 0$$

This yields a tractable set of linear equations, that we can explicitly solve.

$$\begin{aligned}&\partial\ L/\partial\ w_\alpha = \textstyle\sum(yx_{\alpha\ n}-W\cdot X_n x_{\alpha\ n}) = 0\\ &\textstyle\sum yx_{\alpha\ n} - \sum W\cdot X_n x_{\alpha\ n} = 0\\ &\textstyle\sum yx_{\alpha\ n} = \sum W\cdot X_n x_{\alpha\ n}\\ &\textstyle\sum yx_{\alpha\ n} = \sum_n(\sum_\beta\ w_\beta\ x_{\beta\ n})x_{\alpha\ n}\\ &\textstyle\sum yx_{\alpha\ n} = \sum_\beta(\sum_n\ x_{\alpha\ n}x_{\beta\ n})w_\beta\end{aligned}$$

d equations, d unknowns. use matrix inversion to solve.

outer product

define outer product of d-dimensional vectors:

$$[UV^T]_{\alpha\beta} = u_\alpha\ v_\beta$$

outer product is a dxd square matrix, where elemnt (i,j) is $u_i v_j$.

getting back to our gradient ascent...

Vector notation for solution:

$$\begin{aligned}&[\textstyle\sum_n\ x_n x_n^T]W = \sum_m\ y_m x_m\\ &W = [\textstyle\sum_n\ x_n x_n^T]^{-1}(\sum_m\ y_m x_m)\end{aligned}$$

Set:

$A = [\sum_n x_n x_n^T]^{-1}$
$B = \sum_m y_m x_m$

When do we have trouble inverting the matrix?

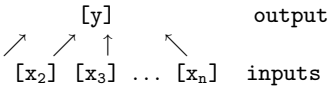
- if we have more inputs than examples
- if the examples all lie in a smaller dimentional space

Options when we can't invert:

- minimum norm: $\min\{|w|^2\}$ such that $\partial L/\partial W = 0$
- weight decay: $\max\{L(W)\text{-}C|W^2|$

8 Logistic Regression

How can we predict a binary output $y \in \{0,1\}$ from $X=(x_1,x_2,\ldots,x_n) \in \mathbb{R}^d$?

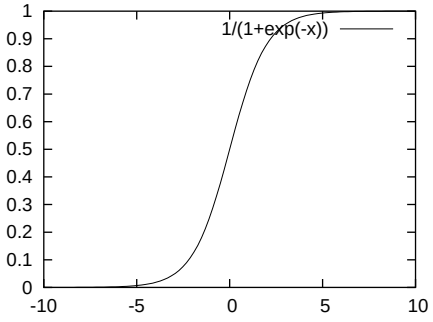


Model input-output by a conditional Bernoulli distribution:

$P(y=1|X) = \sigma(W \cdot X)$

Where we define $\sigma(\cdot)$ as:

$\sigma(z) = 1/(1+e^{-z})$



Properties of $\sigma(\cdot)$:

- $\sigma(z) \to 1$ as $z \to \infty$
- $\sigma(z) \to$-1 as z→-

∞

- $\sigma(-z) = 1\text{-}\sigma(z)$
- $\partial/\partial z\ \sigma(z) = \sigma(z)\sigma(-z)$

8.1 Learning from examples

Assume iid examples $y_n \in \{0,1\}$

$P(y_1, y_2, \ldots, y_n|X_1,X_2,\ldots,X_n) = \prod p(y_n|X_n)$

Estimate parameters W by maximizing the log likelihood

$\text{argmax}[W] \sum \log P(y_1|X_1)$
$= \text{argmax}[W] \sum[\ \log y_n \log[\sigma(W \cdot X_n)] + (1-y_n)\log[1-\sigma(W \cdot X_n)]\ ]$
$= \text{argmax}[W] \sum[\ \log y_n \log[\sigma(W \cdot X_n)] + (1-y_n)\log[\sigma(-W \cdot X_n)]\ ]$

This is a nonlinear function, whose maximum can't be computed in closed form. So we can't just set the gradient to zero and solve. Use iterative algorithms.

Wednesday, February 6, 2002

Consider: continuous state sequence. problem: predict our next state.

Gradient ascent:

$$W \leftarrow W + \eta \partial L / \partial W$$

$$W \leftarrow W - 1/(\partial^2 L / \partial W \partial W^T) \partial L / \partial W$$

9 pset 1

pset1 outside moore 554. μ=30/50

10 Review

2 types of optimization problems:

- linear
- nonlinear
- convex: has more structure than generic nonlinear; but more complex than linear.

Types of solutions:

- exact (clsoed form)
- iterative

Types of convergence:

- finite (polynomial) time
- asymptotic
- monotonic: each iteration improves the answer

11 Convex Sets and Functions

11.1 Introduce convex sets & functions in 1d.

Convex Set

- A set $\Omega \in \mathbb{R}$ is convex if $\forall$ x,y$\in\Omega$:
  - (1-t)x + ty $\in \Omega$ for t$\in$[0,1]
  - i.e., a set is convex if we can pick 2 points, and any point between them is in the set.

Convex Function

- For any line we can draw between 2 points on the function, the function lies below that line.
- A function f(x) is convex on Ω if for all points x,y$\in\Omega$:
  - f((1-t)x + ty) $\leq$ (1-t)f(x) + tf(y) for t$\in$[0,1]
  - Note:
  - (1-t)f(x) + tf(y) for t$\in$[0,1] is the line between 2 points.
  - f((1-t)x + ty) is the function value between 2 points.

Some convex functions:

- x^2
- e^x

f(x) is concave if (-f(x)) is convex. Some concave functions:

- log(x)
- sqrt(x)

Local Minima are Global minima

Theorem: If f(x) is convex on Ω, then:

- Any local minimum is a global minumum.
- (of course, any global minimum is a local minimum.)

Proof by contradiction:

- let x^* be a local minimum of f
- assume $\exists$ y s.t. $f(y)<f(x^*)$
- let $t \in [0,1]$
- Then by convexity: $f((1-t)x^*-ty) \leq (1-t)f(x^*)+tf(y)$
- Then since we're taking a linear combination of $f(x^*)$ and something smaller: $f((1-t)x^*-ty) \leq f(x^*)$
- But then x^* can't be a local minimum (consider limit as $t \rightarrow 0$).

Convex Functions and Tangents

Theorem: if f(x) is once differentiable, then it is convex on Ω iff:

- $f(y) \geq f(x) + f'(x)(y-x)$

Intuition: $f(x)+f'(x)(y-x)$ is the tangent. So what we're saying is that the tangent of any point lies below or at the function at every point..

Proof ($\Rightarrow$):

- for $t \in [0,1]$: $f((1-t)x+ty) \leq (1-t)f(x)+tf(y)$
- so $f(1-t)x+ty)-f(x) \leq t[f(y)-f(x)]$
- so $(1/t)[f(1-t)x+ty)-f(x)] \leq \leq f(y)-f(x)$
- lhs as $t \rightarrow 0$ is the definition of derivative. so take lim $t \rightarrow 0$
- $f'(x)(y-x) \leq f(y)-f(x)$

Proof ($\Leftarrow$):

- Let $x_1,x_2 \in \Omega$. Let $x=(1-t)x_1+tx_2$. So $x \in [x_1,x_2]$.
- By assumption: (Eq. A) $f(x_1) \geq f(x) + (x_1-x)f'(x)$
- By assumption: (Eq. B) $f(x_2) \geq f(x) + (x_2-x)f'(x)$
- Take linear combination of (Eq. A) and (Eq. B):
  - $(1-t)f(x_1) + tf(x_2) \geq [(1-t)+t]f(x) + [(1-t)(x_1-x)+t(x_2-x)]f'(x)$
  - $(1-t)f(x_1) + tf(x_2) \geq f(x) + [(1-t)x_1+tx_2 - (1-t)x-tx]f'(x)$
  - By our definition of x:
  - $(1-t)f(x_1) + tf(x_2) \geq f(x)$
  - $(1-t)f(x_1) + tf(x_2) \geq f((1-t)x_1+tx_2)$

Convexity By Nonnegative Second Derivative

Theorem: if f(x) is twice differentiable, then it is convex on Ω iff:

- $f''(x) \geq 0 \ \forall \ x \in \Omega$

Proof intuition:

- Use a taylor expansion
- Take the following to be true, on faith: $\forall$ x,y $\exists$ $z \in [x,y]$ s.t.
  - $f(y) = f(x) + f'(x)(y-x) + (1/2)f''(z)(y-x)^2$
  - Then $f''(z) \geq 0$ implies convexity..
  - $f(y) \geq f(x) + f'(x)(y-x)$

Examples:

- $d^2/dx^2\ e^x = e^x \geq 0$
- $d^2/dx^2\ x^2 = 2 > 0$
- d^2/dx^2 -log x= $1/x^2 \geq 0$
- d^2/dx^2 -ln $\sigma(z) = -\sigma(z)\sigma(z) \geq 0$

Jensen's inequaltiy

Theorem: let f(x) be a convex function of a random variable $x \in \mathbb{R}$.

- $E[f(x)] = \int dx\ p(x)f(x)$

or:

- $E[f(x)] = \sum_i\ p(x_i)f(x_i)$

Note: x can be defined over discrete set, even though f must be defined on a continuous set.

Then:

- $E[f(x)] \geq f(E[x])$

Proof (assuming f(x) is once differentiable):

- $E[f(x)] \geq E[f(z)+(x-z)f'(z)]\ \forall\ z$
- $E[f(x)] \geq f(z) + (E[x]-z)f'(z)$
- Take $z = E[x]$ $(z \in \Omega)$
- $E[f(x)] \geq f(E[x])$

Examples:

- $E[x^2] \geq E[x]^2$
- $E[e^{xk}] \geq e^{kE[x]^2}$
- $E[\log(x)] \leq \log E[x]$

11.2 Convex Dualtiy

Motivation: if f(x) is convex, and we choose some slope λ for a line, then where will it intercept f(x) such that it touches but is below.. I.e., find the "intercept function" -g(λ) such that:

- $f(x) \geq \lambda\ x\text{-}g(\lambda)$

Ideally, we'd like a tight bound.. :)

A convex function f(x) can be represented in terms of a "conjugate" (or "dual") function g(λ):

- $f(x) = \max_\lambda\{\lambda\ x\text{-}g(\lambda)\}$

The dual function (which is also convex) is given by:

- $g(\lambda) = \max_x\{\lambda\ x\text{-}f(x)\}$

These are "Legendre transformations"

Example:

- $f(x) = e^x$
- $g(\lambda) = \lambda \log\lambda - \lambda$

Which means that:

- $e^x \geq \lambda\ x + \lambda$ - $\lambda \log\lambda$

Example:

- f(x) = log(x)
- g(x) = 1 - log(1/λ) = 1+log(λ)

Which means that:

- f(x) $\leq \lambda$ x - 1 - logλ

11.3 Higher Dimentions

Convex Sets

- If you pick any 2 points, then everything between them must be included.

Conditions

d=1	d>1
x$\in\mathbb{R}$	x$\in\mathbb{R}^d$
f'(x)	df/dX or ∇ f(x)
f''(x)≥ 0	(see below)
duals	f(x) $\geq \Lambda$ X - g(Λ) X$\in\mathbb{R}^d$

below:

- For all V $\in \mathbb{R}^d$:
  - $V^T\ \partial^2 f/dXdX^T\ V \rightarrow 0$

12 Review... Monotonic Convergence

12.1 Auxilliary function

Given a (log) likelihood function L (concave) that we wish to maximize, define an auxilliary function Q(w,w_t) continuous in w and w_t s.t.: - L(w) = Q(w,w) - L(w) $\geq$ Q(w,w_t)

Pick Q such that it's easy to maximize. Then by maximizing Q around w_t, we can find a new w that's better than the old one.

w_{t+1} = argmax$_w$ Q(w,w_t)

12.2 Logistic Regression

L(w) = L_+(w) - L_-(w)
L_+(w) = $\sum_n$(w$\cdot$ x_n) y_n
L_-(w) = $\sum_n$log[1+exp(w$\cdot$ x_n)]

12.3 Multiplicitive Updates:

$$\exp \longleftarrow \exp(w_\alpha) \begin{bmatrix} [(dL_+/dw_\alpha)] \\ [---------] \\ [(dL_-/dw_\alpha)] \end{bmatrix} \eta$$

Where:

η = max$_n$ $\sum_\alpha$ $x_{\alpha\ n}$

(Assumes $x_{\alpha\ n} \geq 0$)

i.e., the learning rate is set by looking at the "densest" image. E.g., for an image (where brighter pixels have higher values), the brightest image sets the larning rate.

This update rule converges monotonically.

12.4 Auxilliary Function

So what auxilliary function do we want to use?

$$Q(w,w_t) = L(w_t) + \sum_{\alpha\ n}\{y_n(w_\alpha - w_{\alpha\ t}) - (1/S)\sigma_n^{(t)}[\exp(s(w-w_{\alpha\ t}))-1]$$

Where:

$\sigma_n^{(t)} \equiv 1/(1+\exp(-w_t \cdot x_n))$

∂/∂ w Q(w,w_t) = 0
$\sum_n$ $x_{\alpha\ n}$ [$y_n - \sigma_n$(t) exp(s(w$_\alpha$-w$_{\alpha\ t}$))]
exp(s(w$_\alpha$-w$_{\alpha\ t}$))($\sum_n \sigma_n^{(t)} x_{\alpha\ n}$ = $\sum_n x_{\alpha\ n} y_n$

We're only computing the gradient (not the hessian), so it's cheaper than other methods (in high dimensionality).

12.5 Constraints

- algorithm is tailored to sparse nonnegative input
  - nonnegativity is required: otherwise derivitives give complex numbers, etc.

- sparsity is desirable: the densest vector limits our learning rate.
- but: for negative data, we can just adjust..

13 State Aggregation

Conditional Probabilities for large discrete variables must be paramaterized. I.e., P(s'|s) is just too big to write out in a tabular form.

Aggregate states into clusters, and work on the cluster level.

- Assume we have a discrete markov process
  - markov process has a countable state space S.
  - markov process has probablistic dynamics P(s'|s)
  - What if |S| is very large?
  - Difficult to elicit P(s'|s) from experts
  - Difficult to learn from examples (many parameters)
  - Why build a model at all?
  - Transfer: if we slightly change the task, models are very useful.

Transform original state space S into an aggregate state space.

- Imagine that states s$\in$ S can be mapped to clusters c$\in$ C and vice versa:
  - P(c|s) = probability that state s is mapped to cluster c
  - P(s'|c) = prob that a state s in cluster c is followed by state s'.

So we could use an alternate notation like:

- $P(c_t|s_t)$ = prob that state s_t is mapped to cluster c_t
- $P(s_{t+1}|c_t)$ = prob that state s_t is followed by s_{t+1}, given that state s_t maps to cluster c_t.

(but we won't :))

State transitions s$\rightarrow$ c$\rightarrow$ s' are mediated by cluster assignments. Thus:

- $P(s'|s) = \sum_c P(c|s)P(s'|c)$

So we're clustering, but allowing probablistic fuzziness.

We are using a "matrix factorization" to provide a compact representation:

- P(s'|s) is a |S|$\times$|S| matrix (call it A)
- P(c|s) is a |S|$\times$|C| matrix (call it B)
- P(s'|c) is a |C|$\times$|S| matrix (call it C)

so we're modelling A as:

- A = B $\times$ C

We're adding some complexity cost: instead of table lookup, we have matrix multiplicaiton (an inner product for a single entry).

Represent this model as a graph:

s $\rightarrow$ c $\rightarrow$ s'

(Each link characterized by a matrix, but we explictly do *not* have a link from S to S')

This decomposition also gives us transitions between clusters..

- $P(c_{t+1}|c_t) = \sum_s P(c_{t+1}|s)P(s|c_t)$

Inferring cluster labels:

$$P(c|s,s') = P(s',c|s)/P(s'|s)$$

$$= \frac{P(s'|c)P(c|s)}{\sum_{c'} P(s'|c')P(c'|s)}$$

So given P(c|s) and P(s'|c), we can compute:

- P(s'|s) (inner product)
- P(c|s,s') (bayes rule)

13.1 Learning

so how do we learn?

Learning from "complete" data

Suppose we have fully labeled trajectories of the form:

- $s_1 \rightarrow c_1 \rightarrow s_2 \rightarrow c_2 \rightarrow s_3 \rightarrow \cdots \rightarrow s_T$

What's the probability of this trajectory under our model?

$$P(\{s_t, c_t\}_{t=1 \ldots T}) = \prod_t P(s_{t+1}|c_t)P(c_t|s_t)$$

How many times does $P(s_{t+1}=s|c_t=c)$ appear? Define counts:

- $N(s \rightarrow c)$ = \# of times state s is followed by cluster c
- $N(c \rightarrow s)$ = \# of times cluster c is followed by state s

Use these counts to rewrite $P(\{s_t,c_t\})$:

$$P(\{s_t, c_t\}) = \prod_s \prod_c P(s|c)^{N(c \rightarrow s)} P(c|s)^{N(s \rightarrow c)}$$

We want to maximize the log likelihood of the data (L_C=log likelihood for complete data):

$$L_C = \sum_s \sum_c \log(P(s|c))N(c \rightarrow s) + \log(P(c|s))N(s \rightarrow c)$$

Then we just do learning by maximizing this log likelihood. In particular, we want to maximize L_C subject to the constraints:

$$\sum_c P(c|s) = 1$$
$$\sum_s P(s|c) = 1$$

Solution to the complete data form:

Maximum likelihood estimates:

$$P(c|s) = N(s \rightarrow c)/\sum_{c'} N(s \rightarrow c')$$
$$P(s'|c) = N(c \rightarrow s')/\sum_{s''} N(c \rightarrow s'')$$

Learning from "incomplete" data

Suppose we just have partially labeled trajectories of the form:

- $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \rightarrow s_T$

I.e., we have a hidden cluster space.. (this seems very similar to HMMs?)

What's the probability of this trajectory under our model?

$$P(\{s_t\}_{t=1 \ldots T}) = \prod_t P(s_{t+1}|s_t)$$
$$= \prod_t \sum_{c_t} P(s_{t+1}|c_t)P(c_t|s_t)$$

14 Learning from Incomplete Data

We have "partially" labeled trajectories

- $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \cdots \rightarrow s_T$

What's the probability of this trajectory under our model?

$$P(\{s_t\}_{t=1\ldots T}) = \prod_t P(s_{t+1}|s_t) = \prod_t \sum_{c_t} P(s_{t+1}|c_t)P(c_t|s_t)$$

Let N(s→ s') count the number of transitions from s to s' in the data.

$$P(s_1,s_2,\ldots,s_T) = \prod_{s,s'} \left[\sum_c P(s'|c)P(c|s)\right]^{N(s\rightarrow s')}$$

Log likelihood (H for "hidden"):

$$L_H(s_1,s_2,\ldots,s_T) = \sum_{s,s'} N(s\rightarrow s')\log\sum_c [P(s'|c)P(c|s)]$$

Then we just do learning by maximizing this log likelihood. In particular, we want to maximize L_H subject to the constraints:

$$\sum_c P(c|s) = 1$$
$$\sum_s P(s|c) = 1$$

We can't solve this in closed form. So use an iterative solution. Use the EM algorithm

14.1 EM algorithm

Intuitive motivation:

- in the complete-data situation, to maximize the likelihood in the observed case, we normalized observed counts N(s→ c) and N(c→ s')
- to maximize log likelihood for the hidden-variable setting:
  - compute inferred counts of clusters (E-step):
    - $\text{Nhat}_k(s\rightarrow c) = \sum_{s'} N(s\rightarrow s')P_k(c|s,s')$
    - $\text{Nhat}_k(c\rightarrow s') = \sum_s N(s\rightarrow s')P_k(c|s,s')$
    - where P_k is given by Bayes' rule
    - use Nhat_k to find MLE for P_{k+1}
    - $P_{k+1}(c|s) = \text{Nhat}_k(s\rightarrow c)/\sum_{c'}\text{Nhat}_k(s\rightarrow c')$
    - $P_{k+1}(s|c) = \text{Nhat}_k(c\rightarrow s)/\sum_{s'}\text{Nhat}_k(c\rightarrow s')$

These updates give monotonic convergence..

These are multiplicitive updates.

$$\frac{\partial L_h}{\partial P(c|s)} = \alpha_{cs}$$

$$\frac{\partial L_h}{\partial P(s'|c)} = \beta_{s'c}$$

The m-step updates can also be written as:

$$P_{k+1}(c|s) = P_k(c|s)\frac{\alpha_{cs}}{\sum_c \alpha_{cs}P_k(c'|s)}$$

$$P_{k+1}(s|c) = P_k(s|c)\frac{\beta_{sc}}{\sum_c \beta_{s'c}P_k(s'|c)}$$

14.2 Proof of Monotonic Convergence

(I got bored)

15 Vector Quantization

Assign an integer label to each vector (basically, we're doing automatic clustering). Choose k centroids. Assign each data point to the nearest centriod; re-estimate the centroids; repeat. For nicely clustered data, this can give useful clusters.

K-means algorithm for data $\{X_n\}$ (each X_n is a vector)

1. Choose μ_i at frandom for i=1,2,...,k (each μ_i is a vector)
2. Assign y_n=argmin$_i$ $|x_i$-$x_n|^2$
3. Set μ_i to the mean of $\{x_n\}y_n$=i
4. Go to step 2 (until convergence)

Questions:

- what type of data is algorithm suited to?
- what exactly is being optimized?
- why does it converge?
- what if there are missing features?

15.1 Clustering

Goal:

- divide data into clusters
- assign new data to clusters

15.2 Mixture Models

Vector quantization is a member of a probablistic set of models called "mixture models."

Goal

- model the distribution p(x) from which iid training examples are drawn. iid examples are $\{x_n\}$.
- Generative model (intuition)
  1. Roll a k-sided (non-fair) die, with probabilities ρ_i (i=1..k), $\sum_i \rho_i = 1$
  2. Based on the outcome of the die, sample x from one of k "component distributions."
  3. Identify each component distribution with a cluster.

The overall pdf is a weighted sum of the individual distributions. If each component distribution is relatively tight, and sparse, then we can find them..

Hidden variable

- In addition to x (observed variable), introduce a discrete hidden (unobserved, latent) variable $z \in \mathbb{Z}$.
- z's pdf is given by ρ_z (die probabilities).

Define a joint pdf P(x,z) = P(x|z)P(z)

So P(x) = sum$_z$ P(x|z)P(z)

"mixture model" terminology: we're "mixing" k different distributions.

Graphical model

$z \rightarrow x$

Component distributions

x is continuous, so we need a parametric form. A useful choise is the multivariate gaussian.

- basically, a guassian among multiple dimensions.

$$P(x|z=i) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_i|}} \exp\{-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)$$

Each component (cluster) i has its own mean and covariance:

μ_i = expected value of x (k-vector)
= E[x|z=i]
Σ_i = covariance matrix (k-by-k matrix)
= E[$(x-\mu_i)\cdot(x-\mu_i)$|z=i]

note! Σ is covariance, not sum ($=\sum$).

Covarinace matrix Σ_i must be positive definite (basically the equivalant to saying that a real number must be positive). Other than that, it can have any value.

$P(x|z=i) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_i|}} \exp\{-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)$
$|\Sigma_i|$ = determinant
Σ_i^{-1} = matrix inverse..

Benefits:

- modelling assumptions are explicit
- know what data will be well-suited to the algorithm
- inferences of many different types are supported by the model.
  - e.g., clustering/vector quantization.
  - restoration of "missing" features
  - novelty/outlier detection
  - learning – viewed as maximum likelihood estimation

15.3 Learning from labeled examples

Suppose we have labeled examples $\{x_n, z_n\}$ where $z_n \in \{1 \ldots k\}$

Define indicator variables:

$$Z_{in} = \begin{cases} 1 \text{ if } z_n = i \\ 0 \text{ otherwise} \end{cases}$$

The probability of iid takes the following form:

$P(\{x_n,z_n\}) = \prod_{n=1..N} P(x_n|z_n)P(z_n)$
$= \prod_n \prod_i \{P(z_n=i)P(x_n|i)\}^{\{Z_{in}\}}$
$= \prod_n \prod_i \{\rho_i P(x_n|i)\}^{\{Z_{in}\}}$

(Fill in $P(x_n|z=i)$ from big ugly equation above :))

Work with log likelihood (for fully-observed case) L_O:

$L_O = \sum_{in} Z_{in}\log(\rho_i) - Z_{in}\log(P(x_n|i))$
$L_O = \sum_{in} Z_{in}[\log(\rho_i) - \frac{1}{2}(x_n-\mu_i)^T\Sigma_i^{-1}(x_n-\mu_i)$
$-d\log(2\pi)/2 - \log(|\Sigma_i|)/2]$

Learning is maximizing L_O with respect to $\{\rho, \mu, \Sigma\}$ subject to the constraints:

- $\sum_i \rho_i = 1$
- Σ_i covariances are positive definite

Solution

- Let $N_i = \sum_n Z_{in}$ counts the number of examples labeled by the ith cluster.
- $\rho_i = N_i/N$
- $\mu_i = 1/N_i \sum_n Z_{in} x_n$
- $\Sigma_i = 1/N_i \sum_n Z_{in}(x_n\text{-}\mu_i)(x_n\text{-}\mu_i)^T$

15.4 Learning from unlabeled examples

Suppose we have unlabeled examples x_n.

$$P(x_1,\ldots,x_n) = \prod_n \sum_z P(x_n|z)P(z)$$

Work with log likelihood (for hidden-variable case) L_H:

$$L_H = \sum_n \log [\sum_z P(x_n|z)P(z)]$$
$$L_H = \sum_n \log [\sum_i P(x_n|z=i)\rho_i]$$

(Again, substitute in $P(x_n|z=i)$ from above)

Learning is maximizing L_H with respect to $\{\rho, \mu, \Sigma\}$ subject to the constraints:

- $\sum_i \rho_i = 1$
- Σ_i covariances are positive definite

Can't solve in closed form.

Solution: use EM

16 EM

Intuitive motivation.. 2 steps:

1. E-step:
   - to maximize L_O, we just used counts N_i of the data in each cluster.
   - But we don't have direct access to these counts, since the training samples are unlabeled.
   - We can make guesses, though: try to find a value for N_i that will make L_H higher than our current estimate, even if it doesn't give the right value.
   - Use Bayes rule:
     - $P(z|x_n) = P(x_n|z)P(z)/p(x_n)$
     - $P(z|x_n) = P(x_n|z)P(z)/(\sum_z P(x_n|z)P(z))$
     - $P(z=i|x_n) = P(x_n|z=i)\rho_i/(\sum_i P(x_n|z=i)\rho_i)$
     - Define a surrogate for the real labels:
     - $\gamma_{in} \equiv P(z=1|x_n)$
     - Fill this in with our definitions for $P(x_n|z=i)$
     - Use the surrogate in place of the indicator variable Z_{in}.
     - Use γ to calculate Nhat.
     - M-step
     - Use Nhat and the equations from the fully-observed case to update the parameters:
     - $\rho_i = \text{Nhat}_i/\text{Nhat}$
     - $\mu_i = 1/\text{Nhat}_i \sum_n \gamma_{in} x_n$
     - $\Sigma_i = 1/\text{Nhat}_i \sum_n \gamma_{in}(x_n\text{-}\mu_i)(x_n\text{-}\mu_i)^T$

16.1 Proof of Monotonic Convergence

Let:

H = hidden variable
V = visible variable
θ = current parameters
θ' = updated parameters

For the mixture model with gaussians:

- H = {z}
- V = {x}
- θ = {ρ,μ,Σ}

But we'll prove convergence for the general case.

$L(\theta') = \sum_n \log P(V_n|\theta')$
$L(\theta') = \sum_n \log P(H_n,V_n|\theta')/P(H_n|V_n,\theta')$

This is true for any values of H_n. In particular, we can construct $L(\theta')$ as a weighted sum of values that equal $L(\theta')$. I.e., it is always true that:

$\sum_x Zp(x) = Z$

if Z doesn't depend on x. In our case, Z=$L(\theta')$. So we can use a weighted sum over the PDF for H_n.

$L(\theta') = \sum_n \sum_{Hn} P(H_n|V_n,\theta)[\log P(H_n,V_n|\theta')/P(H_n|V_n,\theta')]$
$L(\theta') = \sum_n \sum_{Hn} P(H_n|V_n,\theta)[\log P(H_n,V_n|\theta')/P(H_n|V_n,\theta) +$
$\log P(P(H_n|V_n,\theta)/P(H_n|V_n,\theta'))]$

We can interpret:

$\log P(P(H_n|V_n,\theta)/P(H_n|V_n,\theta'))$

as the KL distance between distributions paramaterized by θ and θ'.

KL-distance is always positive: use it to generate an inequality.

$L(\theta') \geq \sum_n \sum_{Hn} P(H_n|V_n,\theta) \log[P(H_n,V_n|\theta')/P(H_n|V_n,\theta)]$
$Q(\theta',\theta) \equiv \sum_n \sum_{Hn} P(H_n|V_n,\theta) \log[P(H_n,V_n|\theta')/P(H_n|V_n,\theta)]$

- $L(\theta') = Q(\theta',\theta')$
- $L(\theta') \geq Q(\theta',\theta)$

It follows that $L(\theta)$ improves monotonically under:

- $\theta' = \text{argmax}_{\theta'} Q(\theta',\theta)$

17 Mixture Model

we need to solve for parameters

- $\rho_i = P(z{=}i)$
- μ_i, Σ_i = multivariate gaussian parameters

H = hidden variable
V = visible variable
θ = current parameters
θ' = updated parameters

Find auxilliary function Q of the log likelihood:

$$Q(\theta',\theta) \equiv \sum_n \sum_{H_n} P(H_n|V_n,\theta) \log[P(H_n,V_n|\theta')/P(H_n|V_n,\theta)]$$

Use iterative scaling:

$$\theta' = \text{argmax}_{\theta''} Q(\theta'', \theta)$$

17.1 EM

E step

Compute the term in Q(θ',θ) that depends on θ'

$$Q(\theta',\theta) \equiv \sum_n \sum_{H_n} P(H_n|V_n,\theta) \log[P(H_n,V_n|\theta')] + \ldots$$

(We don't care about the second term, since it doesn't depend on θ'. So, it won't affect our argmax!)

We can rewrite this as a posterior expectation of the log joint distribution:

$$Q(\theta',\theta) = \sum_n E[\log P(H_n,V_n|\theta') \mid V_n,\theta]$$

(Thus the name E-step for "expectation")

For the mixture model:

$$\prod_n P(x_n,z_n) = \prod_n \prod_i \{\rho_i P(x_n|i)\}^{\{Z_{in}\}}$$

where:

$$Z_{in} = \begin{cases} 1 \text{ if } z_n = i \\ 0 \text{ otherwise} \end{cases}$$

Note that in this context, Z_{in} are random variables (since the labels z_n are random variables). But they have very simple expected values:

$$E[Z_{in}] = P(z_n{=}i)$$
$$E[Z_{in}|x_n] = P(z_n{=}i|x_n) = \gamma_{in}$$

$$\begin{aligned} &\sum_n E[\log P(H_n,V_n|\theta') \mid V_n,\theta] \\ &= E[\sum_n z_{in} (\log \rho_i' - 1/2(x_n-\mu_i')^T\Sigma_i'(x_n-\mu_i') - \\ &\qquad 1/2\log(2\pi)^d |\Sigma_i'| \\ &\quad |x_n,\theta] \\ &= \sum_{ni} \gamma_{in} (\log \rho_i' - 1/2(x_n-\mu_i')^T\Sigma_i'(x_n-\mu_i') - \\ &\qquad 1/2\log(2\pi)^d |\Sigma_i'| \\ &\quad |x_n,\theta] \end{aligned}$$

$$\gamma_{in} = P(z_n{=}i) = P(x_n,z_n{=}i)/P(x_n)$$

(compute with θ, not θ')

18 Union Model

Consider a binary event z ∈ {0,1} that is triggered by one or more precursor events y_i ∈ {0,1} (n.b.: strong notion of causality):

$$z = \bigcup_i y_i$$

This is one possible model of "sensor fusion". Sensors are noisy.

Examples

- Security: an alarm is sounded when you detect one or more intrusions. But there might be false alarms, wind, etc.
- Vision: an object is spotted if it appears in any subregion of the viewing area.
- Speech: to detect if someone is speaking, break signal into freqs, and see if we detect speech at any freq.

to:, cc:, attchmnt:, subject:

$[z\ [y_1\ x_1\ x_2\ \ldots\ x_k]\ [y_2\ x_1\ x_2\ \ldots\ x_k]\ \ldots\ [y_n\ x_1\ x_2\ \ldots\ x_k]\ ]$

18.1 Sensor model

$P(y_i=1|x\text{-}i) = \sigma(w_i \cdot x_i)$

If any sensor fires (y=1), then the output (z) is 1.

$P(z=1|y) = 1-\prod_i(1-y_i)$

Take probability that sensors didn't fire; take product of that to get probability that no sensors fired; one minus that is the probability that any sensor fired.. $P(z=1|\{x_i\}) = 1-\prod_i\sigma(-w_i \cdot x_i)$

18.2 Interference

Given that we know something about z, what inferences can we make about the y's and x's? (x's fixed)

If z didn't fire, then we know that all y_i's didn't fire.

$E[y_i|z=0] = P(y_i=1|z=0) = 0 \text{ for all } i$

If we learn that z fired, then it can only increases our confidence that each of the y_i's fired:

$$\begin{aligned} E[y_i|z=1,\{x_j\}] &= P(y_i=1|z=1,\{x\}) \\ &\geq P(y_i=1|\{x\}) \\ &= E[y_i|x_i] \end{aligned}$$

Proof:

$P(y_i=1|z=1,\{x\}) = P(y_i=1,\ z=1|\{x\})/P(z=1|\{x\})$

Because of our model, z is independant of the x's, given the y's.

$P(y_i=1|z=1,\{x\}) = P(y_i=1|\{x\})P(z=1|y_i=1)/P(z=1|\{x\})$

We know that $P(z=1|y_i=1)=1$ so:

$$P(y_i=1|z=1,\{x\}) = P(y_i=1|\{x\})/P(z=1|\{x\})$$
$$P(y_i=1|z=1,\{x\}) = \sigma(w_i \cdot x_i)/[1-\prod_i\sigma(-w_i \cdot x_i)]$$

Denominator is always less than one:

$P(y_i=1|z=1,\{x\}) \geq \sigma(w_i \cdot x_i) = P(y_i=1|\{x\})$

18.3 Learning

Can we use a global learning algorithm, without combinatorial explosion of training time? Use EM..

Incomplete Data scenario

Assume we have a set of partially labelled examples $\{\langle x_{in}, z_{in}\rangle\}$ (n indexes over examples). E.g., for visual id, we are told which images contain the target, but not where in the image.

[Footnote: If the x_{in} are actually different instances of the same object, then this type of scenario is called "multiple instance learning." In reality, this is just a special case of sensor fusion..?]

Maximize log likelihood:

$L = \sum_n \log P(z_n|x_{1n}, x_{2n}, \ldots, x_{kn})$

Break into positive and negative examples:

$$L = \sum_n z_n \log P(z_n=1|x_{1n}, x_{2n}, \ldots, x_{kn}) + (1-z_n) \log P(z_n=0|x_{1n}, x_{2n}, \ldots, x_{kn})$$
$$L = z_n \log[1-\prod_i \sigma(-w_i \cdot x_i)] + (1-z_n) \log [\prod_i \sigma(-w_i \cdot x_i)]$$
$$L = z_n \log[1-\prod_i \sigma(-w_i \cdot x_i)] + (1-z_n) \sum_i \log \sigma(-w_i \cdot x_i)$$

Highly nonlinear: use an iterative algorithm.

18.4 EM

- EM is an iterative algorithm for optimizing the (global) log likelihood.
- But, at each iteration, it decouples different sensors.

E step

$$\theta = \{w_i\}$$
$$\theta' = \{w_i'\}$$
$$\sum_n E[\log P(y,z|\{x_{in}\},\theta') \mid z_n,\theta]$$

$$\gamma_{in} = E[y_i|z_n,\{x_{jn}\},\theta] = P(y_i=1|z_n,\{x_{in}\},\theta) = z_n \, \sigma(w_i \cdot x_i)/(1-\prod_i \sigma(-w_i \cdot x_i))$$

19 Markov Models

- Random variable $s_t \in \{1,2,\ldots,n\}$ = state at time t.
- Markov assumption about dynamics ("k-th order markov model"):
  - $P(s_t|s_1, s_2, \ldots, s_{t-1}) = P(s_t|s_{t-k}, \ldots, s_{t-1})$

Strengths..

- simple to compute P(sequence)
- simple to estimate model parameters from data

Weaknesses

- to model kth order correlations, we need to store n^k parameters
- in the real world, we don't have direct access to state variable
  - "partially observable worlds"

20 Hidden Markov Models

$$P(s,o) = P(s_1)\prod P(s_t|s_{t-1})\prod P(o_t|s_t)$$

Graphical model:

$$[S_1 \; O_1 \; [S_2 \; O_2 \; [S_3 \; O_3 \; [\ldots \; [S_t \; O_t]]]]]$$

Example: speech recognizer for the word "cat"

- state = phoneme
- observation = cat
- states = [silence, c, a, t, silence]

20.1 Parameters

Initial state probabilities:

π_i = P(s_1=i)

Transition probabilities:

a_{ij} = P(s_{t+1}=j|s_t=i)

Emission probabilities:

b_{ik} = b_i(k) = P(o_t=k|s_t=i)

20.2 Computing the likelihood

Use decoder lattuces and forward and backward and viterbi etc.. [I've done this enough times that I don't want to write notes for it again.. See manning & schutze for a good explanation.]

21 Bayes Nets

Two-component model. Qualitative component is a DAG: it defines independence properties. Quantitative component is a "conditional probability table" which lists the probability of a variable for each combination of values of its *parents* in the DAG.

Bayes net = G + CPT. G is a DAG. CPT is a conditional probability table.

Generally, the DAG represents causality. More formally, it represents info about conditional independence..

21.1 Semantics

Always:

$P(X_1,\ldots,X_n) = \prod_i P(X_i|x_1\ldots X_{i-1})$

Special: (*)

$P(X_1,\ldots,X_n) = \prod_i P(X_i|pa(X_i))$

Where pa(X_i) are the parents of X_i in G.

Definition:

- $\forall$ G $\forall$ P, G "represents" P if P obeys factorization (*) given by G.
- "General" independence conditions specified by G?

21.2 Example:

B = burglary A = arlarm E = earthquake J = john M = mary

Graph:

B→ A; E→ A
A→ J; A→ M

CPT:

B	E	P(A=1)
0	0	...
0	1	...
1	0	...
1	1	...

21.3 "Explaining away" and nonmonotonicity

There's some marginal probability of burglary:

P(B=1) small

Now suppose that we observe the event that the alarm went off

P(B=1|A=1) large

Now suppose that we additionally observe the event that there was an earthquake:

P(B=1|A=1,E=1) small

This last step, where P(B=1|...) gets small again, is known as "explaining away." We have a probabalistic explanation for the alarm, and as a result, P(B=1|...) gets small again.

This is a useful property – gracefully incorperates the fact that our beliefs can be revised on the basis of increasing evidence.

21.4 Causality

choose direction of DAG to match causality. This tends to give us more efficient representations. But we can actually choose any ordering. All that the bayes net represents is info about correlations, not about causality. When interpreting a bayes net graph, we have to be careful not to assume that it really asserts causality..

21.5 Conditional independences

- node value is conditionally dependant, given parents, of all other nodes.

Let X and Y be variables. S is a set of observed values.

S = {J=1, M=0}

When can we assert that X and Y are independant, given S? If there's no (undirected) path from X→ Y that doesn't go through S.

Blocking

```
X{–}···→ Z →···{–}Y
```

A path p btwn X and Y is "blocked" by S if there is a single node Z∈ S on the path and edges along path touching Z go in the *same* direction. Here, we've observed an "intermediate cause".

```
X{–}···← Z →···{–}Y
```

A path p btwn X and Y is "blocked" by S if there is a single node Z∈ S on the path and edges along path touching Z go away from Z. Here, we've observed a "common cause".

```
X{–}···→ Z ←···{–}Y
       /\
      /  \
     /no S\
    /_____\
```

A path p btwn X and Y is "blocked" by S if there is a single node Z∉ S on the path, and edges along path touching Z go towards Z, and no descendants of Z are in S.

Say X and Y are d-separated by S in G if every path from X to Y is blocked by S.

If X and Y are d-separated by S in G then X and Y are conditionally independant. If X and Y are not d-separated by S in G then there exists some P represented by G s.t. X and Y are conditionally dependant.

22 Inference in Bayes Nets

Input:

```
Bayes net <G, CPTs>
Set S of observations (e.g., {J=1,M=0})
Variable X
```

Desired output:

```
P(X|S)
```

Goal:

- tractable: running time polynomial time of <G, CPT>

- inference

Monday, March 25, 2002

Hm. Assignment 6 is due today. Huh. Ok. :) I guess I'll do that tonight..

PROJECTS!! :)

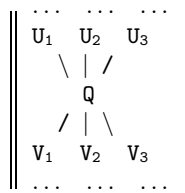
22.1 Polytrees

Definition: if a DAG is a polytree, then it contains no *undirected* cycles.

Today: efficient algorithm for inference in polytree Bayes nets

Definitions

Node Q, our query node, has parents and children.



Where U's parent's & children may be observed; and V's parents and children may be observed; and U's and V's may be observed. Since it's a polytree, U's are not connected to V's (except via Q). Furthermore, U's are not connected to each other and V's are not connected to each other.

Definition:

- S = set of evidence nodes
- $\forall$ X,Y: S(X,Y) $\subseteq$ S = set of evidence nodes reachable from X avoiding Y.

For example, the evidence in U_1's blob is S(U_1,Q). The evidence in V_i's blob is S(V_i,Q)..

Definitions:

- $S^+ = \bigcup_i$ S(U_i,Q)
- $S^- = \bigcup_i$ S(V_i,Q)

Algorithm

Use message passing to distribute information.

```
For every X→ Y:
   1. X sends to Y: P(X=x,S(X,Y)) ∀ x
   2. Y sends to X: P(S(Y,X)|X=x) ∀ x
```

X sends Y a message for each value of X, which gives the joint probability of the setting of x and the probability of all evidence connected to X (but not connected to Y).

Y sends X a message for each value of X, which gives the conditoinal probability of the evidence connected to Y (but not X), given the setting of X to the given value..

We still need to convince ourselves:

1. base case: leaf nodes can find message values
2. recursive case: non-leaf nodes can find message values
3. we can compute P(Q|S) given messages to Q.

Do #3 first (find P(Q=q|S)). First, reduce the conditional to a joint probability:

$P(Q=q|S) = P(Q=q,S)/P(S) = \alpha\ P(Q=q,S)$

Then find the joint probability:

$P(Q=q,S) = P(Q=q,S^+)P(S^-|Q=q,\ S^+)$

Use d-separation (on Q, since for the right probability, Q is on the right hand side of the conditional – it's evidence (in this circumstance, at least)) (this is d-sep case 1):

$P(S^-|Q=q,\ S^+) = P(S^-|Q=q)$

So:

$P(Q=q,S) = P(Q=q,S^+)P(S^-|Q=q)$

So we can find P(Q=q|S) if we can find:

$P(Q=q,S^+)$
$P(S^-|Q=q)$

For the first one:

$P(Q=q,S^+) = \sum_{ubar} P(Ubar=ubar,S^+)P(Q=q|Ubar=ubar,S^+)$

Because it's a bayes net, we can reduce this to:

$P(Q=q,S^+) = \sum_{ubar} P(Ubar=ubar,S^+)P(Q=q|Ubar=ubar)$

We can find P(Q=q|Ubar=ubar) in the CPTs.

By d-separation condition 3:

$P(Ubar=ubar,S^+) = \prod_i\ P(U_i=u_i,S(U_i,X))$

So:

$P(Q=q,S^+) = \sum_{ubar}(\prod_i\ P(U_i=u_i,S(U_i,X)))P(Q=q|Ubar=ubar)$

Get the left part from messages; and the right part from CPTs.

Note: new tutorial materials are now on the web page.

Review:

- Bayes net = DAG+CPTs
- Represent *any* $P(X_1,\ldots,X_n)$
- Order of decomposition matters!
- d-separation: if X,Y d-sep by S $\Rightarrow$ P(X,Y|S)=P(X|S)P(Y|S)
- Problem of inference: compute P(X|S)
- Polytree algorithm: linear run time

22.2 Non-Polytree

What happens if the underlying DAG is not a polytree?

Approach 1: make it a tree.

```
season -> sprinkler -> wet
season -> rain -> wet
wet -> slippery
```

Node merging:

- Combine sprinkler & rain into a single 4-value variable.
- Everything that happens within the node is done exhaustively
- Size of merged node table is exponential
- Works well if you have a few small loops; not good with big loopy networks.
- Exponential in the biggest merged node's size

Cut-set conditioning:

- Create 2 new graphs, by fixing the value of a node in the cycle (e.g., season) to each possible value.
- Essentially "removes" the node from the graph – eliminates cycle.
- Run polytree on new graphs (which are polytrees).
- Exponential in the cut-set size (number of nodes we "removed")

Approach 2: Get as far from a tree as you can

- Polytrees: each node has a "small" number of "strong" influences
- Alternative: large numbers of weak influences

Consider for simplicity a 2-layer bayes net. Variables:

$u_1\ u_2\ \ldots\ u_n = U$

$x_1\ x_2\ \ldots\ x_m = X$

We have full connectivity from each u_i to each x_j.

We can think of U as "hidden causes" and X as "observable effects".

By d-separation on U's:

$P(X|U) = \prod_j P(x_j|U)$

Marginal independance of the u's:

$P(U) = \prod_i P(u_i)$

$P(U|X) \neq \prod_i P(U_i|X)$

Parametric representations..

- Give a weight vector to each node
- Use weight vectors to give a compact representation of the CPTS: $P(x_i|U)$. Otherwise, these would be exponential.
- Assume $P(X=1|U) = \sigma(\theta \cdot U)$

Think of the weights as being roughly uniform (e.g., within 2 orders of magnitude of each other)

- limits the influence of individual inputs.

This is a standard definition of a neural network!

Possible definitions for σ-esqe funcitons include:

- sigmoid $\sigma = (1+\exp(-z))^{-1}$
- noisy-or $\oplus = 1\text{-}e^{-z}$

(Assume each input bias $p_i=1/2$, just for convenience – we can handle the general case)

$P(x_i=1,\ldots,x_m=1) = (1/2)^n \sum_U \prod_i \oplus(U)$

$\sigma(z) = 1\text{-}e^{-z}$

We don't like 1-exponentials, so upper bound it with an exponential, depending on what part of the 1-exp we're looking at:

$\log(1-e^{-z}) \leq \lambda\, z - g(\lambda)$
$1-e^{-z} \leq e^{\lambda z} - g(\lambda)$
$g(\lambda) = (\lambda+1)\log(\lambda+1) - \lambda \log\lambda$

23 Game Theory

23.1 Where does this fit?

Previous topics included:

- reinforcement learning: learning how to act in an uncertain environment
- unsupervised learning & bayesian networks: probablistic modeling and inference

Both were passive environments ("nature").

Game theory is a reasonable starting point to look at problems of interaction.

23.2 Single-stage matrix games

- Players i=1, ..., n (start with n=2)
- Each player has actions $a_1 \ldots a_k$ (start with k=2, actions={0,1})
- Payoff matrices (1 per player): matrix M_i for player i that is indexed by the actions of the 2 players.

$M_i(x_1, x_2) \quad x_1, x_2 \in \{0,1\}$

Specifies the payoff to player i if player 1 plays x_1 and player 2 plays x_2.

- Moves are simultaneous

Example: Prisoner's dilema (confess/deny)

	M_1			M_2	
	x_1=c	x_1=d		x_1=c	x_1=d
x_2=c	-5	-10	x_2=c	-5	0
x_2=d	0	-1	x_2=d	-10	-1

Can also write as:

	x_1=c	x_1=d
x_2=c	(-5,-5)	(0,-10)
x_2=d	(-10,0)	(-1,-1)

A "stable solution" is one where neither player has any incentive to change their action.

Stable solution = (confess, confess)

Example: Penny matching (heads/tails)

	x_1=h	x_1=t
x_2=h	(1,0)	(0,1)
x_2=t	(0,1)	(1,0)

Player 1 wins if the moves agree; Player 2 wins if the moves disagree. This is a "zero-sum" game (well, actually it's "constant-sum" game, but for strategic purposes, they're the same.

No stable entry. But there is a stable strategy:

- both players flip a fair coin.

Mixed Strategies

Define $p_i \triangleq \Pr[x_i=1]$ be a randomized strategy for player i

Define $M_i(p_1,p_2) = E(M_i(x_1,x_2))$ where x_1 is drawn from p_1, x_2 from p_2.

(p_1,p_2) is a "mixed strategy" for the game ("mixed"=probabilistic).

```
M1(p1,p2) = (1-p1)(1-p2)M1(0,0) *
            p1p2M1(0,1) *
            (1-p1)(1-p2)M1(1,0) *
            p1p2M1(1,1)
```

If we fix p2, then $M_1(p_1,p_2)$ is a linear function of p_1. So if p_2 is fixed, there are 3 possible ways the function can look: positive slope, negative slope, zero slope:

- if positive slope: play fixed action 1
- if negative slope: play fixed action 0
- if zero slope: play anything

Nash Equilibrium

Definition: a mixed strategy (p_1,p_2) is a "Nash Equilibrium" for the matrix game (M_1,M_2) if two symmetric conditions hold:

- $M_1(p_1,p_2) \geq M_1(p_1',p_2)\ \forall\ p_1' \in [0,1]$
- $M_2(p_1,p_2) \geq M_2(p_1,p_2')\ \forall\ p_2' \in [0,1]$

So neither player has a unilateral incentive to deviate.

Example: Correlated equilibria (go/stop)

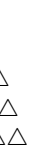
```
      x1=g       x1=s
x2=g (-10,-10) (0,-1)
x2=s (-1,0)    (-1,-1)
```

23.3 Nash's Theorem

For any matrix game (M_1,M_2) there exists a (possibly mixed) Nash equilibrium. Holds for any n.

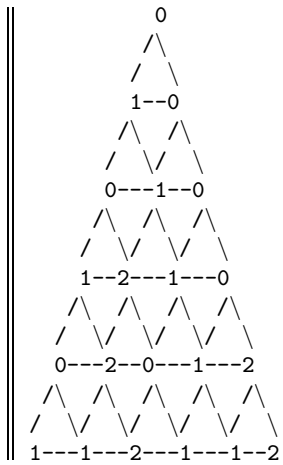
Sperner Labeling

- Consider an equilateral triangle
- Triangulate into smaller equilateral triangles
  - segment with 5 horiz sections, then segment each section with smaller equilateral triangles



A sperner labeling:

- Puts a 0, 1, or 2 label at ever vertex of the segmented triangle
- exterior corners are labeled 0, 1, and 2.
- Vertices on exterior sides can not have the label from the opposite corner.
- Internal vertices get any label



Sperner's Lemma: any sperner labeling results in at least one small triangle with veritces labeled (0,1,2).

Intuition:

- Think of each triangle as a room
- Vertex is any (0,1) edge.
- Since we always move through (0,1) edges, we must be in either a (0,1,0), a (0,1,1), or a (0,1,2) triangle. If it's (0,1,2), we're done. Otherwise, go through the other "door". Keep going through "doors" until we get to a (0,1,2) triangle.

Proof:

- First, add edges on (0,1) side of big triangle.
  - (ensures that there is exactly 1 exterior "door" = (0,1) edge)
  - Enter the triangle containing an exterior (0,1) edge
  - If not in a (0,1,2) triangle, but entered a triangle via a (0,1) edge, then enter the other adjancent triangle across a (0,1) edge.
- We will never re-visit any triangle.
  - Consider the first triangle that we revisit
  - We must have entered from either the place that we entered or the place that we left. But then we must have revisited that triangle. So this wasn't the first triangle we revisited. But that's a contradiction.
  - We can't "leave" the castle, since there is only one exterior (0,1) edge.

Brower's Fixed Point Theorem: For any convex region and a function that maps points from that region to that region, then it has a fixed point.

Brouwer's Fixed Point Theorem

- Let S be any (n-dimensional) simplex (e.g., S=equilateral triangle in $\mathbb{R}$)
- Let ϕ: S$\rightarrow$ S be any continuous mapping
- Then $\exists$ x s.t. ϕ(x)=x

Proof sketch (for special case of the equilateral triangle):

- Let T_1, T_2, ..., T_n, ... be a nested sequence of successively finer triangulations.
- Define labeleings L_1, L_2, ..., L_n, ... for each triangulation T_n.
- Triangle labeling:
  - Let x be an interior edge that we're trying to label, which is a subtriangulation of T.
  - Draw a ray from x through ϕ(x).
  - Each edge of the outer triangle determines a single label

- Label x with the label for the (outer) edge that the ray passes through
- Break ties (at vertices) in favor of lowest label

Hm, this might not be quite right. But the ideas is that we're supposed to be able to define a labeling that depends on ϕ(x) that's a sperner labeling.

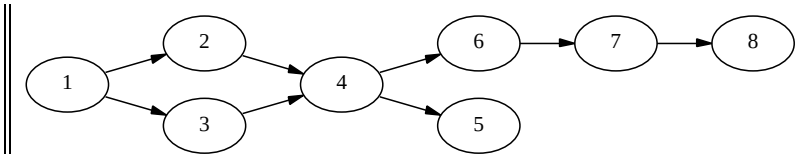
- So in every labeling L_n under ϕ, $\exists$ a small (0,1,2) triangle.
- Let x_i be the center of a (0,1,2) triangle in L_i
- (from real analysis) the infinite sequence x_1, x_2, ..., x_n, ... has an infinite convergant subsequence with limit x.

Claim: $\phi(x) = x$.

- intuition: x is the center of an arbitrarily small 0-1-2 triangle.
- in a 0-1-2 triangle, ϕ maps the corners (which are close to each other) in very different directions: the angles spanned by the corner vectors are $\geq \pi/3$.
- continuity says that if we pick a small enough region R around x, then the size of the corresponding ϕ(R) must be small
- so R must overlap ϕ(R).
- Make R arbitrarily small, and x=ϕ(x).

24 Graphical Games

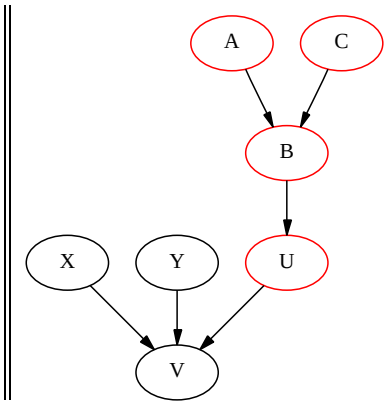
Players 1, . . ., n with joint action xbar and mixed strategy pbar.



Represent using local payoffs: Size is exponential in the max degree, not in n.

24.1 Tree

Assume that G is a tree. Then we can define an efficient algorithm..



$\text{Up}_G(U)$ = set of all nodes "upstream" from U, including U

G^U = graph induced by $\text{Up}_G(U)$

i.e., just the local game matrices in all nodes above U. But we need to change the game matrix for U: it expects an input from V. Fix a value for it (i.e., marginalize V's input out of U's matrix).

So now we've created a new smaller graphical game. It has a nash equilibrium (by Nash's theorem). Call this an "upstream nash equilibrium from U".