Dimensionality reduction (cont.)

Machine Learning – 10701/15781

Carlos Guestrin
Carnegie Mellon University

April 25th, 2007

Lower dimensional projections

- Rather than picking a subset of the features, we can new features that are combinations of existing features

X_1, x_2, x_3

$X_2^2, x_1 x_3^3 \ldots$

Select some of these

projection:

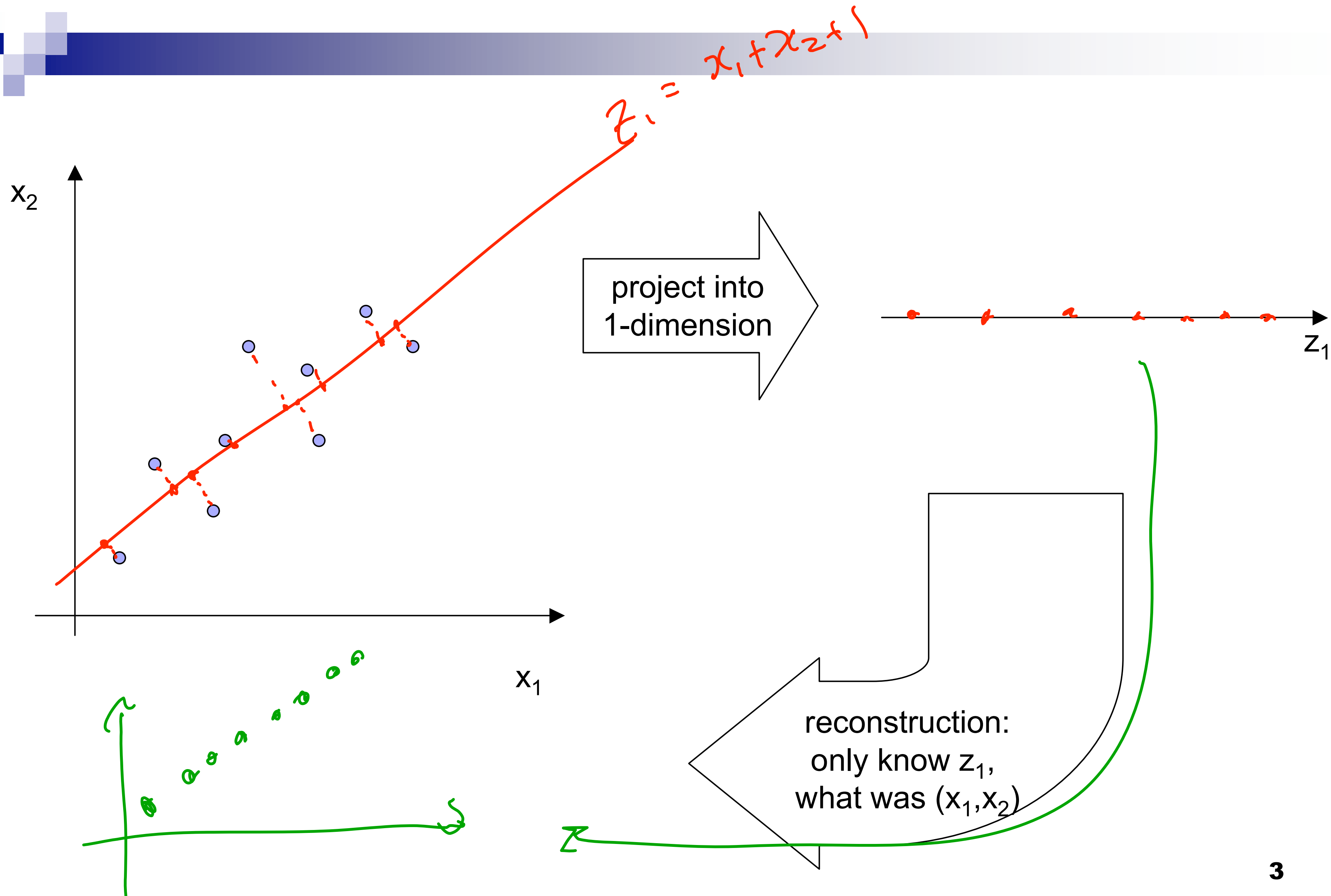
new feature, e.g.,

$$X_{new} = 0.5X_1 - 0.75x_2 + 0.92x_3 \ldots$$

new feature

- Let's see this in the unsupervised setting
  - just **X**, but no Y

Linear projection and reconstruction



Principal component analysis – basic idea

- Project n-dimensional data into k-dimensional space while preserving information:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

- Choose projection with minimum reconstruction error

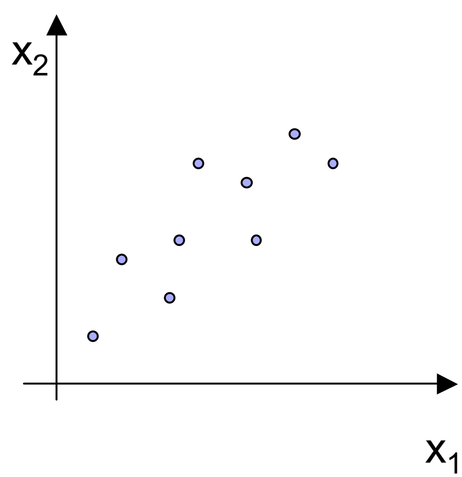
Linear projections, a review

- Project a point into a (lower dimensional) space:
  - **point**: $\mathbf{x} = (x_1,\ldots,x_n)$
  - **select a basis** – set of basis vectors – $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$
    - we consider orthonormal basis:
      - $\mathbf{u}_i \cdot \mathbf{u}_i=1$, and $\mathbf{u}_i \cdot \mathbf{u}_j=0$ for $i \neq j$
  - **select a center** – $\bar{\mathbf{x}}$, defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: $(z_1,\ldots,z_k)$, $z_i = (\mathbf{x}-\bar{\mathbf{x}}) \cdot \mathbf{u}_i$
    - minimum squared error

PCA finds projection that minimizes reconstruction error

- Given m data points: $\mathbf{x}^i = (x_1^i,\ldots,x_n^i)$, i=1…m
- Will represent each point as a projection:
  - $\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$ where: $\bar{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^i$ and $z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$
- PCA:
  - Given k·n, find $(\mathbf{u}_1,\ldots,\mathbf{u}_k)$ minimizing reconstruction error:

$$error_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$



Understanding the reconstruction error

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

$$z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$$

- Given k·n, find ($\mathbf{u}_1, \ldots, \mathbf{u}_k$) minimizing reconstruction error:

$$error_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

- Note that $\mathbf{x}^i$ can be represented exactly by n-dimensional projection:

$$\mathbf{x}^i = \bar{\mathbf{x}} + \sum_{j=1}^{n} z_j^i \mathbf{u}_j$$

- Rewriting error:

Reconstruction error and covariance matrix

$$error_k = \sum_{i=1}^{m} \sum_{j=k+1}^{n} [\mathbf{u}_j \cdot (\mathbf{x}^i - \bar{\mathbf{x}})]^2$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}^i - \bar{\mathbf{x}})(\mathbf{x}^i - \bar{\mathbf{x}})^T$$

Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis $(\mathbf{u}_1,\ldots,\mathbf{u}_n)$ minimizing:

$$error_k = \sum_{j=k+1}^{n} \mathbf{u}_j^T \Sigma \mathbf{u}_j$$

- Eigen vector:

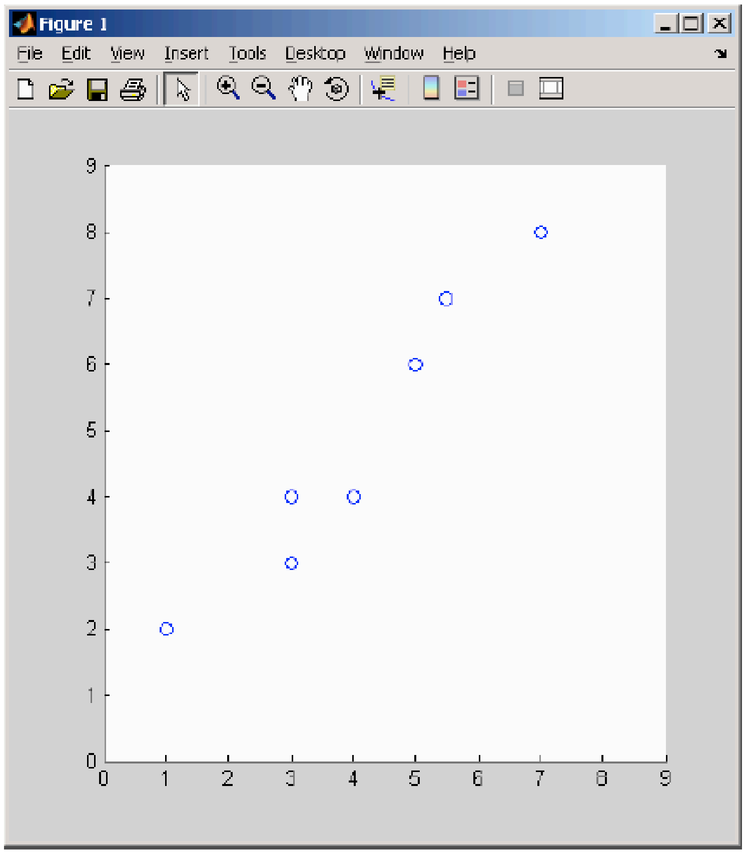
- Minimizing reconstruction error equivalent to picking $(\mathbf{u}_{k+1},\ldots,\mathbf{u}_n)$ to be eigen vectors with smallest eigen values

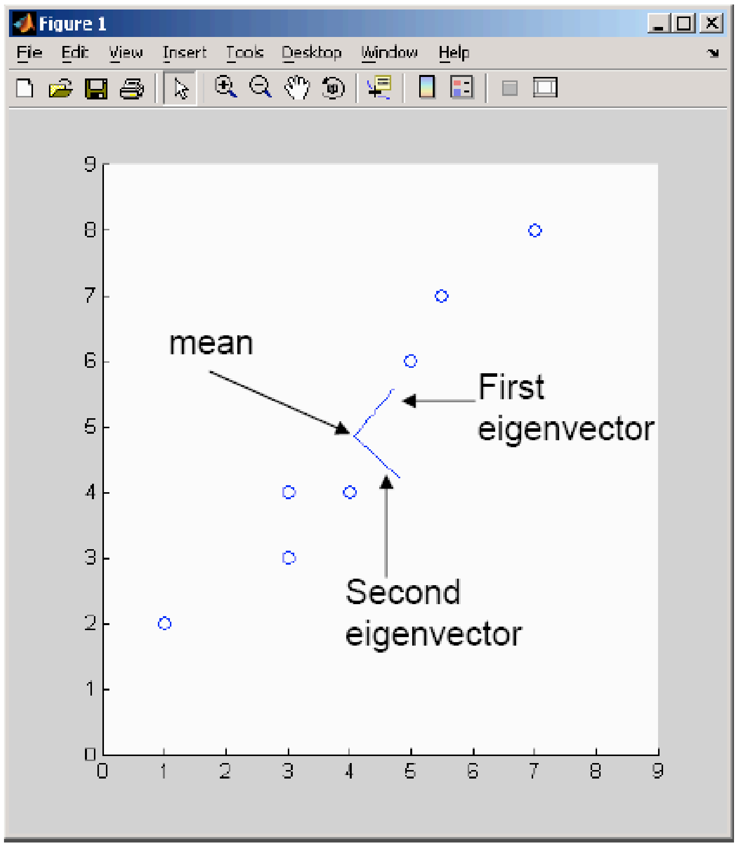
Basic PCA algoritm

- Start from m by n data matrix $\mathbf{X}$
- **Recenter**: subtract mean from each row of $\mathbf{X}$
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- **Compute covariance matrix**:
  - $\Sigma \leftarrow 1/m\ \mathbf{X_c}^{\mathbf{T}}\ \mathbf{X_c}$
- Find **eigen vectors and values** of Σ
- **Principal components:** k eigen vectors with highest eigen values

PCA example

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

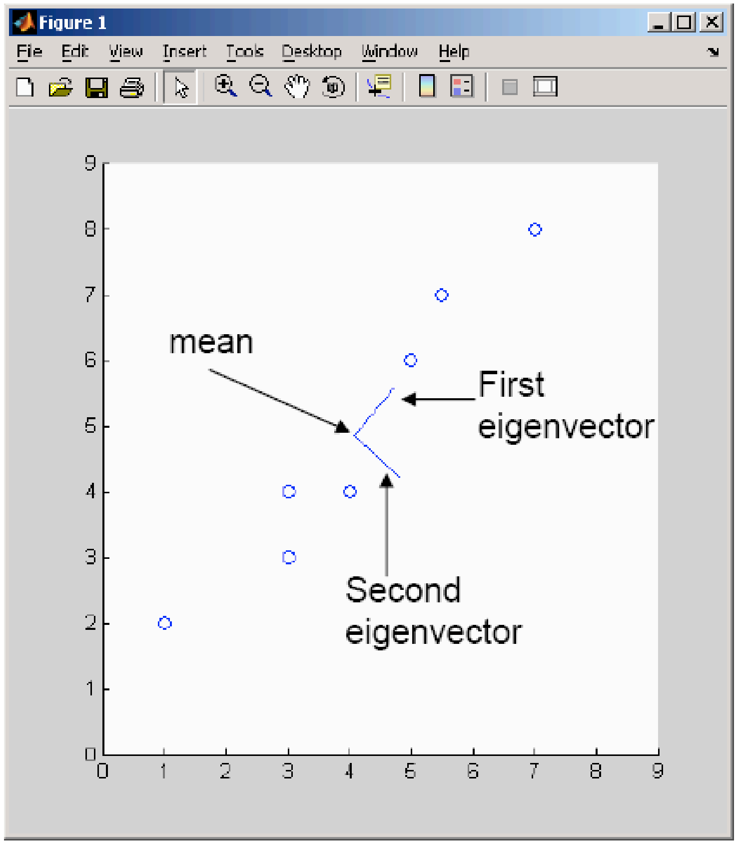


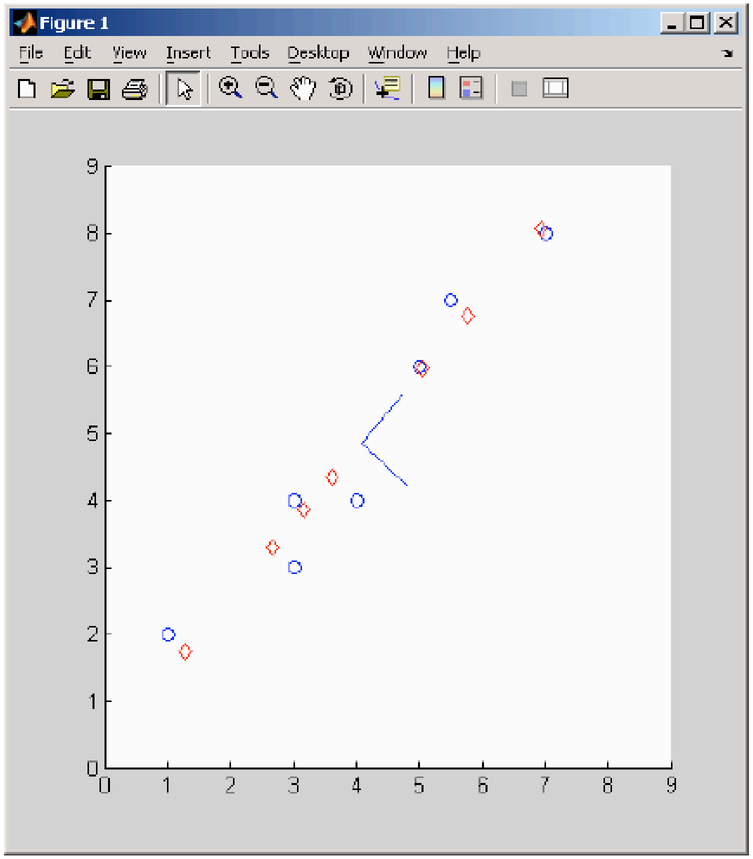


PCA example – reconstruction

$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^{k} z_j^i \mathbf{u}_j$$

only used first principal component

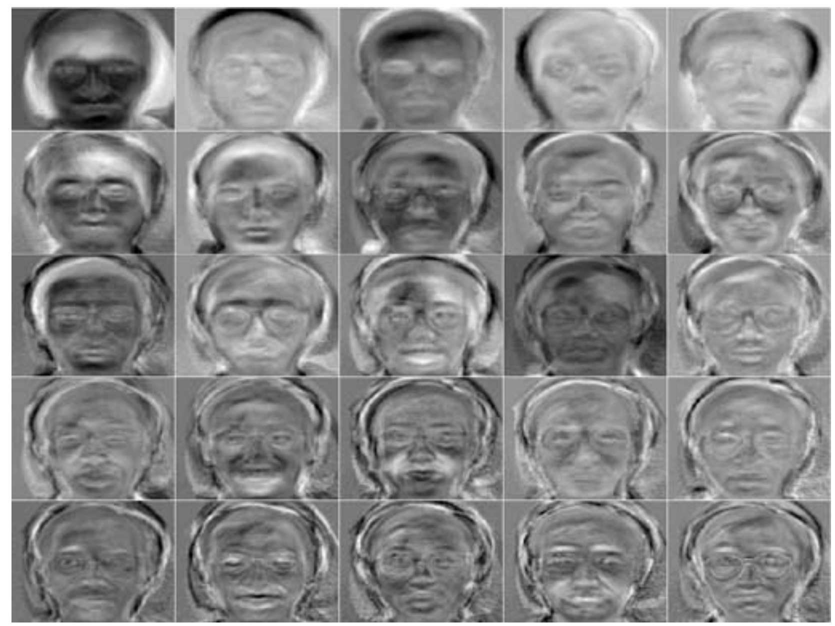




Eigenfaces [Turk, Pentland '91]

- Input images:

- Principal components:



Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:

Relationship to Gaussians

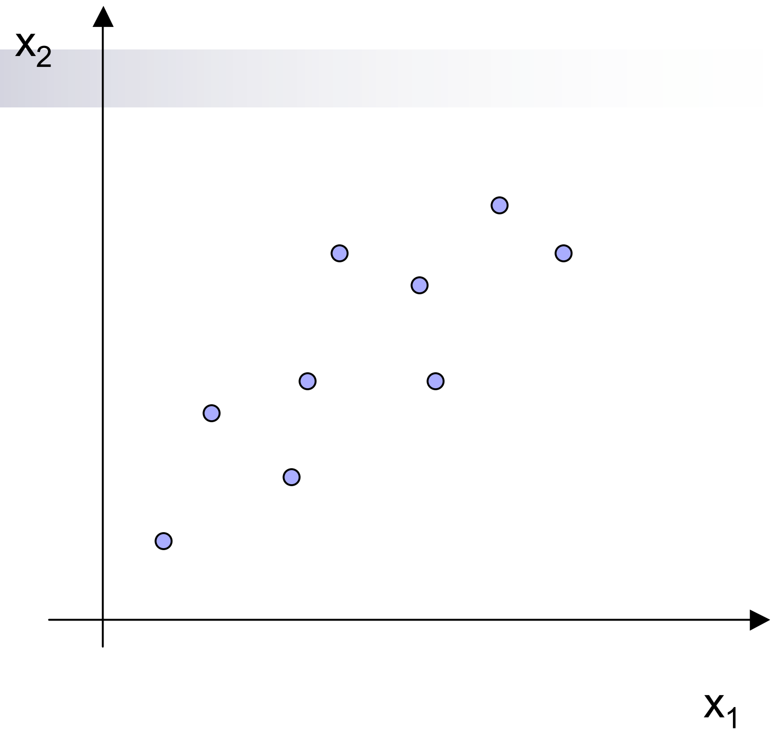
- PCA assumes data is Gaussian
  - $\mathbf{x} \sim N(\bar{\mathbf{x}};\Sigma)$
- Equivalent to weighted sum of simple Gaussians:

$$\mathbf{x} = \bar{\mathrm{x}} + \sum_{j=1}^{n} z_j \mathbf{u}_j; \quad z_j \sim N(0;\sigma_j^2)$$

- Selecting top k principal components equivalent to lower dimensional Gaussian approximation:

$$\mathbf{x} \approx \bar{\mathrm{x}} + \sum_{j=1}^{k} z_j \mathbf{u}_j + \varepsilon; \quad z_j \sim N(0;\sigma_j^2)$$

  - $\varepsilon \sim N(0;\sigma^2)$, where σ^2 is defined by error_k



Scaling up

- Covariance matrix can be really big!
  - Σ is n by n
  - 10000 features ! $|\Sigma|$
  - finding eigenvectors is very slow…
- Use singular value decomposition (SVD)
  - finds to k eigenvectors
  - great implementations available, e.g., Matlab svd

SVD

- Write $\mathbf{X} = \mathbf{W}\ \mathbf{S}\ \mathbf{V}^T$
  - $\mathbf{X} \leftarrow$ data matrix, one row per datapoint
  - $\mathbf{W} \leftarrow$ weight matrix, one row per datapoint – coordinate of $\mathbf{x}^i$ in eigenspace
  - $\mathbf{S} \leftarrow$ singular value matrix, diagonal matrix
    - in our setting each entry is eigenvalue λ_j
  - $\mathbf{V}^T \leftarrow$ singular vector matrix
    - in our setting each row is eigenvector $\mathbf{v}_j$

PCA using SVD algoritm

- Start from m by n data matrix $\mathbf{X}$
- **Recenter**: subtract mean from each row of $\mathbf{X}$
  - $\mathbf{X_c} \leftarrow \mathbf{X} - \overline{\mathbf{X}}$
- Call SVD algorithm on $\mathbf{X_c}$ – ask for k singular vectors
- **Principal components:** k singular vectors with highest singular values (rows of $\mathbf{V}^T$)
  - **Coefficients** become:

Using PCA for dimensionality reduction in classification

- Want to learn f:**Xa**Y
  - **X**=<X_1,…,X_n>
  - but some features are more important than others

- **Approach**: Use PCA on **X** to select a few important features

PCA for classification can lead to problems…

- Direction of maximum variation may be unrelated to "discriminative" directions:

- PCA often works very well, but sometimes must use more advanced methods
  - e.g., Fisher linear discriminant

What you need to know

- Dimensionality reduction
  - why and when it's important
- Simple feature selection
- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD
  - problems with PCA

Announcements

- Homework 5:
  - Extension: Due Friday at 10:30am
  - Hand in to Monica, Wean 4619
- Project:
  - Poster session: Friday May 4th 2-5pm, NSH Atrium
    - please arrive a 15mins early to set up
  - Paper: Thursday May 10th by 2pm
    - electronic submission by email to instructors list
    - maximum of 8 pages, NIPS format
    - no late days allowed
- FCEs!!!!
  - Please, please, please, please, please, please give us your feedback, it helps us improve the class! ☺
    - http://www.cmu.edu/fce

Markov Decision Processes (MDPs)

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

April 25th, 2006

Thus far this semester

- Regression:
- Classification:
- Density estimation:

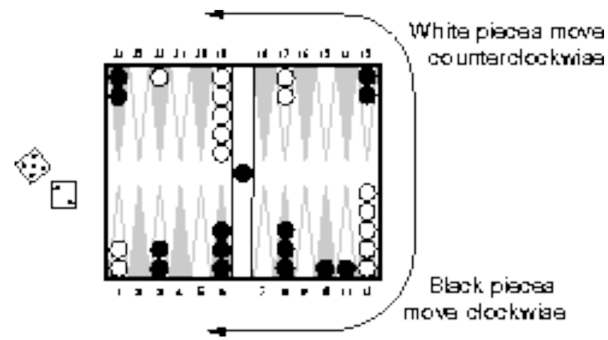
Learning to act

[Ng et al. '05]

- Reinforcement learning
- An agent
  - Makes sensor observations
  - Must select action
  - Receives rewards
    - positive for "good" states
    - negative for "bad" states

Learning to play backgammon
[Tesauro '95]

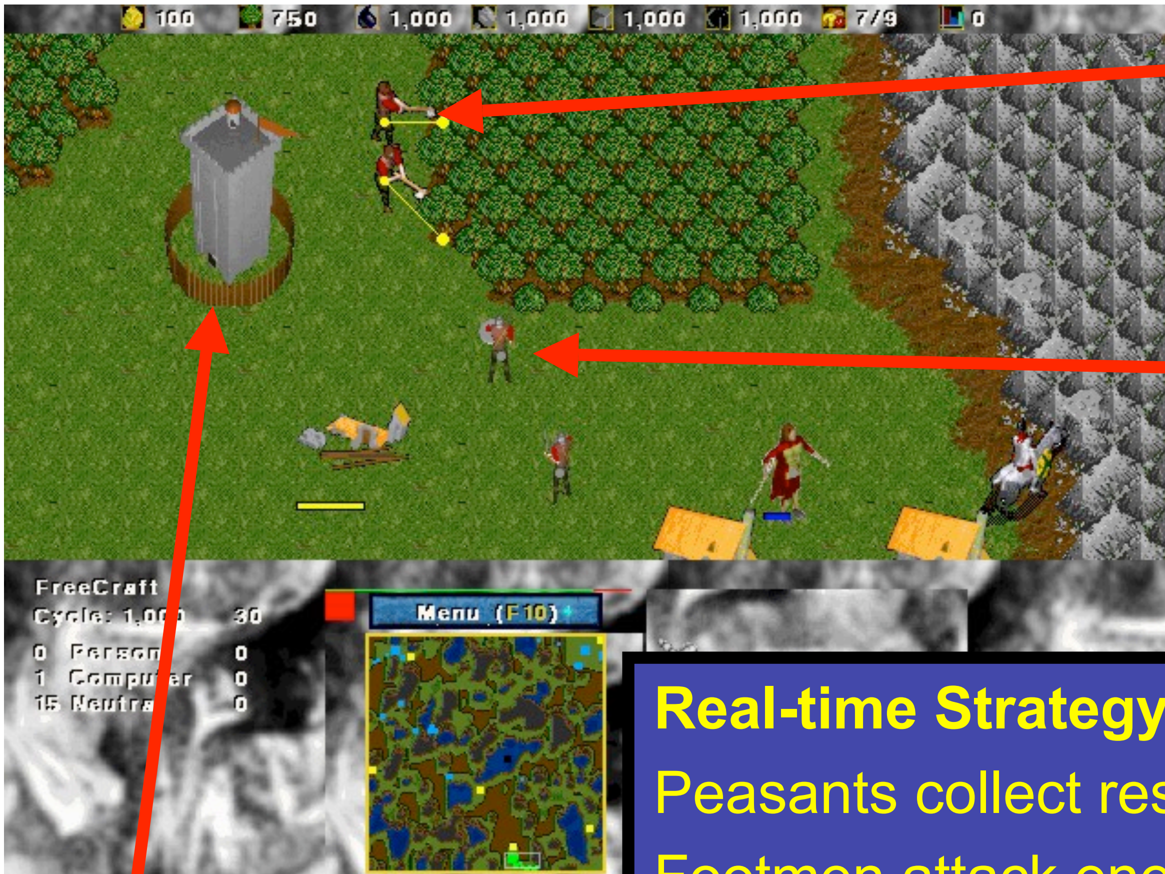
- Combines reinforcement learning with neural networks
- Played 300,000 games against itself
- Achieved grandmaster level!



Roadmap to learning about reinforcement learning

- When we learned about Bayes nets:
  - First talked about formal framework:
    - representation
    - inference
  - Then learning for BNs
- For reinforcement learning:
  - Formal framework
    - Markov decision processes
  - Then learning

FreeCraft



peasant

footman

building

Real-time Strategy Game

Peasants collect resources and build

Footmen attack enemies

Buildings train peasants and footmen

States and actions

- State space:
  - Joint state **x** of entire system
- Action space:
  - Joint action **a**= {a_1,…, a_n} for all agents

States change over time

- Like an HMM, state changes over time
- Next state depends on current state and action selected
  - e.g., action="build castle" likely to lead to a state where you have a castle
- Transition model:
  - Dynamics of the entire system P(**x**'|**x**,**a**)

Some states and actions are better than others

- Each state **x** is associated with a reward
  - positive reward for successful attack
  - negative for loss

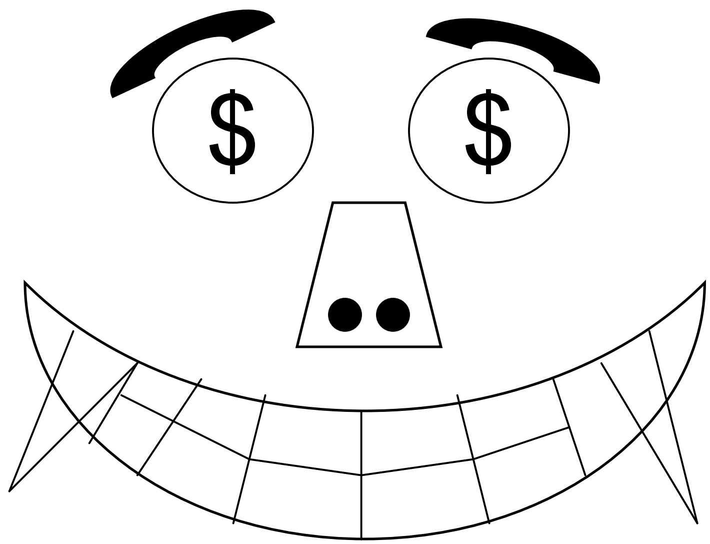
- Reward function:
  - Total reward R(**x**)

Discounted Rewards

An assistant professor gets paid, say, 20K per year.

How much, in total, will the A.P. earn in their life?

20 + 20 + 20 + 20 + 20 + … = Infinity



What's wrong with this argument?

Discounted Rewards

"A reward (payment) in the future is not worth quite as much as a reward now."

- Because of chance of obliteration
- Because of inflation

Example:

Being promised $10,000 next year is worth only 90% as much as receiving $10,000 right now.

Assuming payment n years in future is worth only $(0.9)^n$ of payment now, what is the AP's Future Discounted Sum of Rewards ?

Discount Factors

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor γ" is

(reward now) +

γ (reward in 1 time step) +

γ^2 (reward in 2 time steps) +

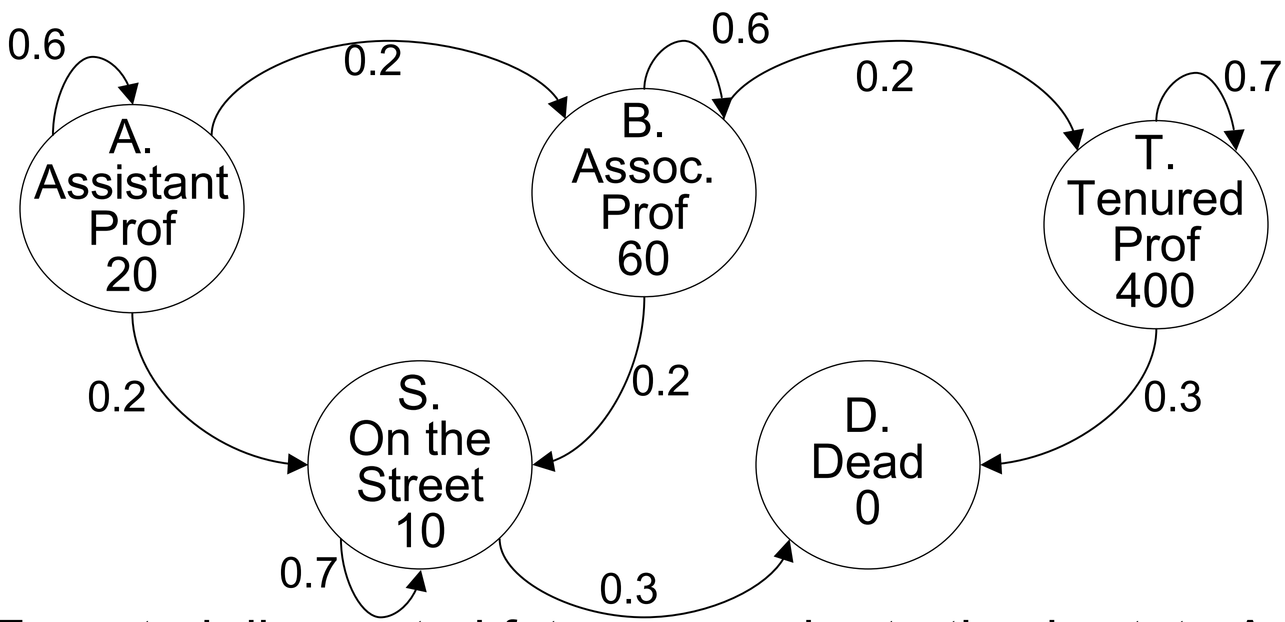
γ^3 (reward in 3 time steps) +

:

: (infinite sum)

The Academic Life

Assume Discount Factor $\gamma = 0.9$



Define:

V_A = Expected discounted future rewards starting in state A

V_B = Expected discounted future rewards starting in state B

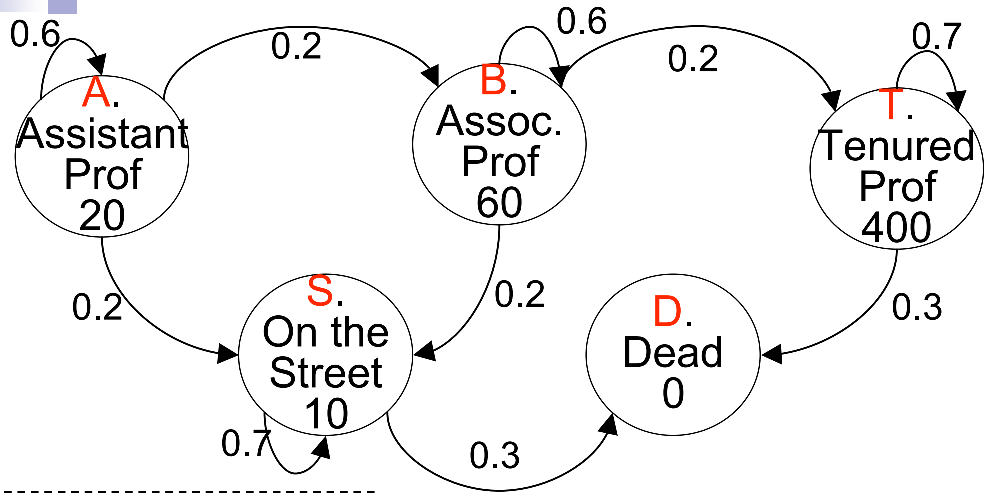
V_T = " " " " " " " T

V_S = " " " " " " " S

V_D = " " " " " " " D

How do we compute V_A, V_B, V_T, V_S, V_D ?

Computing the Future Rewards of an Academic



Assume Discount Factor $\gamma = 0.9$

Joint Decision Space

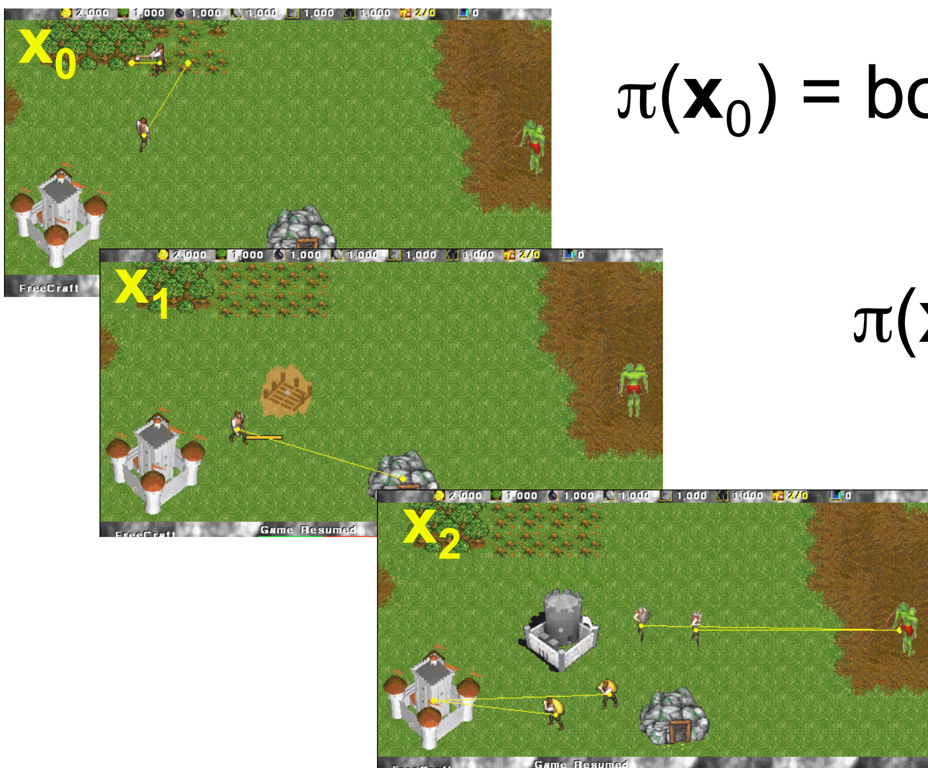
Markov Decision Process (MDP) Representation:

- State space:
  - Joint state **x** of entire system
- Action space:
  - Joint action **a**= $\{a_1, \ldots, a_n\}$ for all agents
- Reward function:
  - Total reward R(**x**,**a**)
    - sometimes reward can depend on action
- Transition model:
  - Dynamics of the entire system P(**x**'|**x**,**a**)

Policy

Policy: $\pi(\mathbf{x}) = \mathbf{a}$

At state **x**, action **a** for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

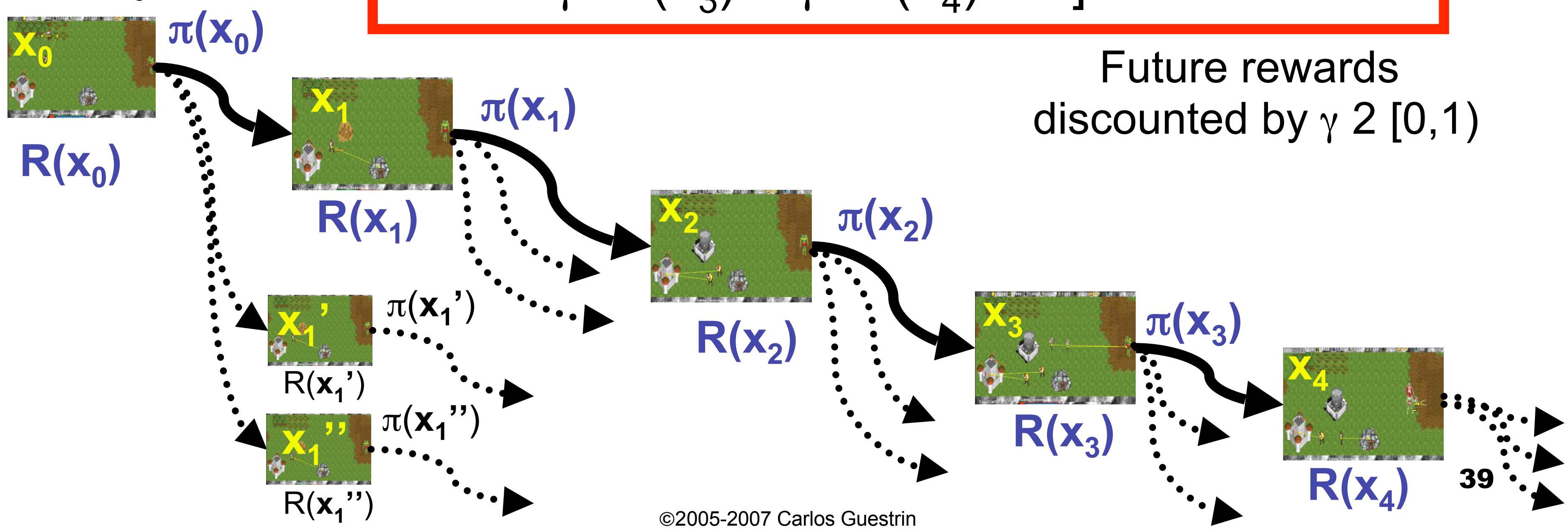
$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

Value of Policy

Value: $V_\pi(\mathbf{x})$ → Expected long-term reward starting from **x**

$$V_\pi(\mathbf{x_0}) = \mathbf{E}_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + L]$$

Start from $\mathbf{x_0}$



Future rewards discounted by γ 2 [0,1)

Computing the value of a policy

$$\boxed{\begin{aligned} V_\pi(\mathbf{x_0}) = \mathbf{E}_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \\ \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + L] \end{aligned}}$$

- Discounted value of a state:
  - value of starting from x_0 and continuing with policy π from then on

$$\begin{aligned} V_\pi(x_0) &= E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots] \\ &= E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)] \end{aligned}$$

- A recursion!

Computing the value of a policy 1 – the matrix inversion approach

$$V_\pi(x) \;=\; R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

- Solve by simple matrix inversion:

Computing the value of a policy 2 – iteratively

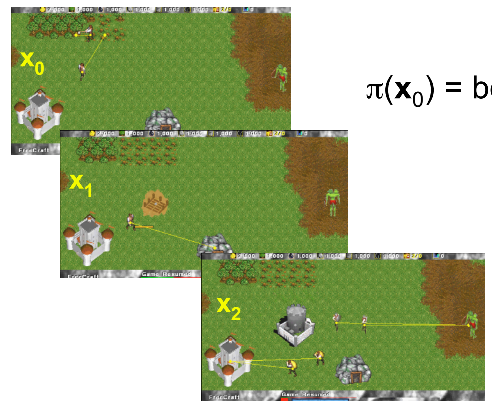
$$V_\pi(x) \;=\; R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

- If you have 1000,000 states, inverting a 1000,000x1000,000 matrix is hard!
- Can solve using a simple convergent iterative approach: (a.k.a. dynamic programming)
  - Start with some guess V_0
  - Iteratively say:
    - $V_{t+1} = R + \gamma\, P_\pi\, V_t$
  - Stop when $||V_{t+1}-V_t||_1 \cdot \varepsilon$
    - means that $||V_\pi-V_{t+1}||_1 \cdot \varepsilon/(1-\gamma)$

But we want to learn a **Policy**

- So far, told you how good a policy is…
- But how can we choose the best policy???
- Suppose there was only one time step:
  - world is about to end!!!
  - select action that maximizes reward!

Policy: $\pi(\mathbf{x}) = \mathbf{a}$ → At state $\mathbf{x}$, action $\mathbf{a}$ for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

Another recursion!

- Two time steps: address tradeoff
  - good reward now
  - better reward in the future

Unrolling the recursion

- Choose actions that lead to best value in the long run
  - Optimal value policy achieves optimal value V*

$$V^*(x_0) \;=\; \max_{a_0} R(x_0, a_0) + \gamma E_{a_0}[\max_{a_1} R(x_1) + \gamma^2 E_{a_1}[\max_{a_2} R(x_2) + \cdots]]$$

Bellman equation

- Evaluating policy π:

$$V_\pi(x) \;=\; R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

- Computing the optimal value V* - Bellman equation

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) V^*(\mathbf{x}')$$

Optimal Long-term Plan

Optimal value function $V^*(\mathbf{x})$ → Optimal Policy: $\pi^*(\mathbf{x})$

$$Q^*(\mathbf{x},\mathbf{a}) = R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

Optimal policy:

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{a}} Q^*(\mathbf{x},\mathbf{a})$$

Interesting fact – Unique value

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

- *Slightly surprising fact*: There is only one V^* that solves Bellman equation!
  - there may be many optimal policies that achieve V^*
- *Surprising fact*: optimal policies are good everywhere!!!

$$V_{\pi^*}(x) \geq V_\pi(x), \;\; \forall x, \;\; \forall \pi$$

Solving an MDP

Solve Bellman equation → Optimal value $V^*(\mathbf{x})$ → Optimal policy $\pi^*(\mathbf{x})$

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

Bellman equation is non-linear!!!

Many algorithms solve the Bellman equations:

- Policy iteration [Howard '60, Bellman '57]
- Value iteration [Bellman '57]
- Linear programming [Manne '60]
- …

Value iteration (a.k.a. dynamic programming) – the simplest of all

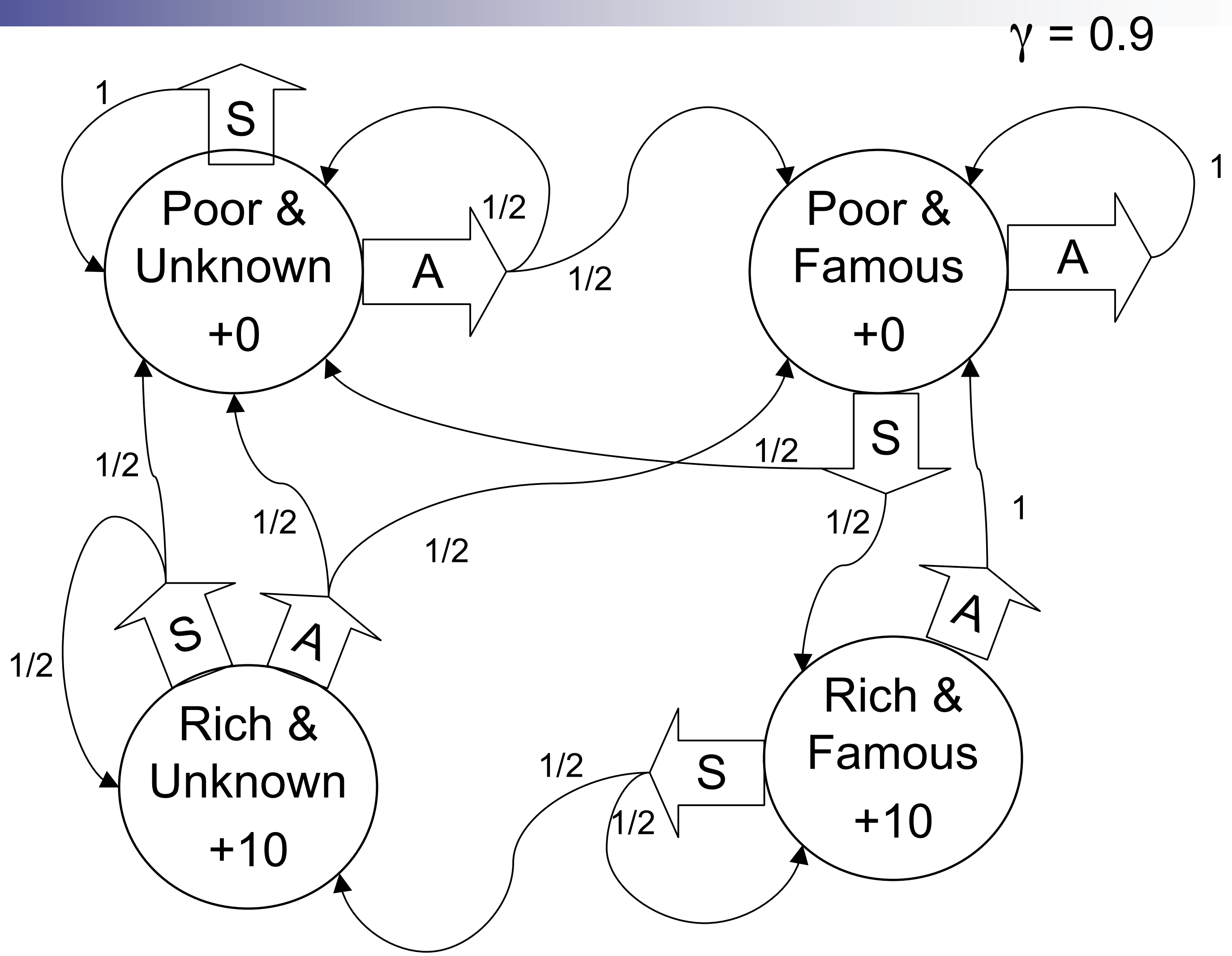
$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

- Start with some guess V_0
- Iteratively say:
  - $$V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V_t(\mathbf{x}')$$
- Stop when $||V_{t+1}-V_t||_1 \cdot \varepsilon$
  - means that $||V^*-V_{t+1}||_1 \cdot \varepsilon/(1-\gamma)$

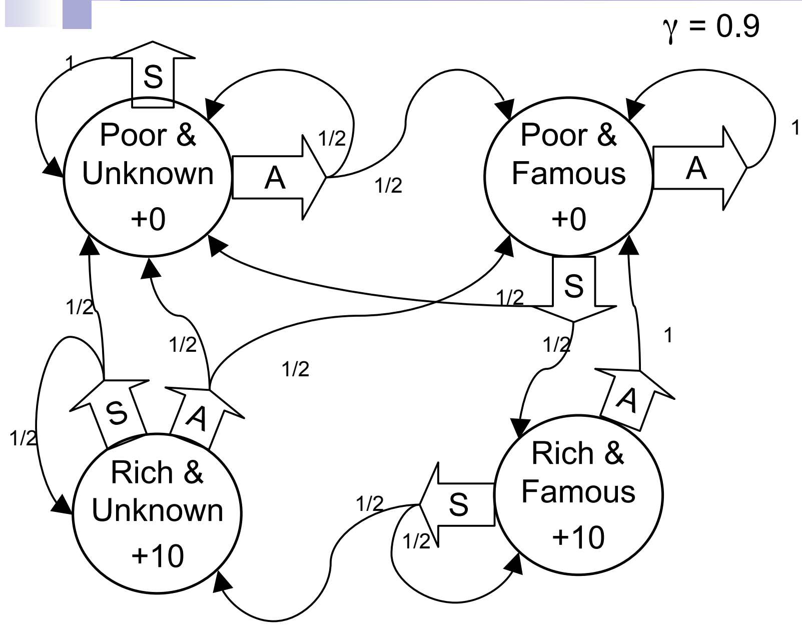
A simple example

You run a startup company.

In every state you must choose between Saving money or Advertising.



Let's compute $V_t(x)$ for our example

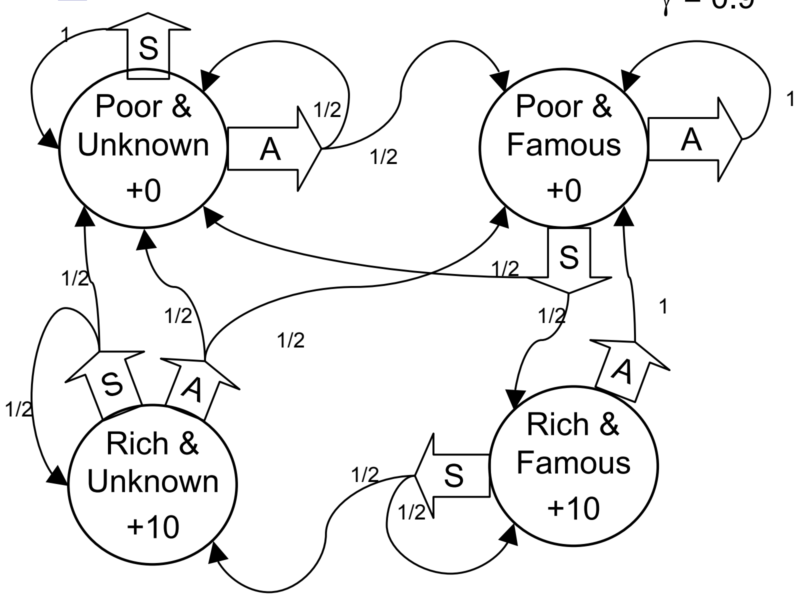


t	V_t(PU)	V_t(PF)	V_t(RU)	V_t(RF)
1				
2				
3				
4				
5				
6				

$$V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V_t(\mathbf{x}')$$

Let's compute $V_t(x)$ for our example

$\gamma = 0.9$



t	V_t(PU)	V_t(PF)	V_t(RU)	V_t(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	6.53	25.08	18.55
4	3.852	12.20	29.63	19.26
5	7.22	15.07	32.00	20.40
6	10.03	17.65	33.58	22.43

$$V_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V_t(\mathbf{x}')$$

Policy iteration – Another approach for computing π^*

- Start with some guess for a policy π_0
- Iteratively say:
  - evaluate policy:

$$V_t(\mathbf{x}) = R(\mathbf{x}, \mathbf{a} = \pi_t(\mathbf{x})) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a} = \pi_t(\mathbf{x})) V_t(\mathbf{x}')$$

  - improve policy:

$$\pi_{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x}, \mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, \mathbf{a}) V_t(\mathbf{x}')$$

- Stop when
  - policy stops changing
    - usually happens in about 10 iterations
  - or $||V_{t+1}-V_t||_1 \cdot \varepsilon$
    - means that $||V^*-V_{t+1}||_1 \cdot \varepsilon/(1-\gamma)$

Policy Iteration & Value Iteration: Which is best ???

It depends.

- Lots of actions? Choose Policy Iteration
- Already got a fair policy? Policy Iteration
- Few actions, acyclic? Value Iteration

Best of Both Worlds:

- Modified Policy Iteration [Puterman]
  - …a simple mix of value iteration and policy iteration

3rd Approach

Linear Programming

LP Solution to MDP

[Manne '60]

Value computed by linear programming:

$$\text{minimize}: \sum_{\mathbf{x}} V(\mathbf{x})$$

$$\text{subject to}: \begin{cases} V(\mathbf{x}) \geq R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V(\mathbf{x}') \\ \forall \mathbf{x}, \mathbf{a} \end{cases}$$

- One variable $V(\mathbf{x})$ for each state
- One constraint for each state **x** and action **a**
- **Polynomial time solution**

What you need to know

- What's a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing V_π
- Optimal value function and optimal policy
  - Bellman equation
- Solving Bellman equation
  - with value iteration, policy iteration and linear programming

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials