

# PCA

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

November 28<sup>th</sup>, 2007

©2005-2007 Carlos Guestrin

1

## Lower dimensional projections

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

e.g., feature selection:  
use  $x_1, x_7, x_{11}$

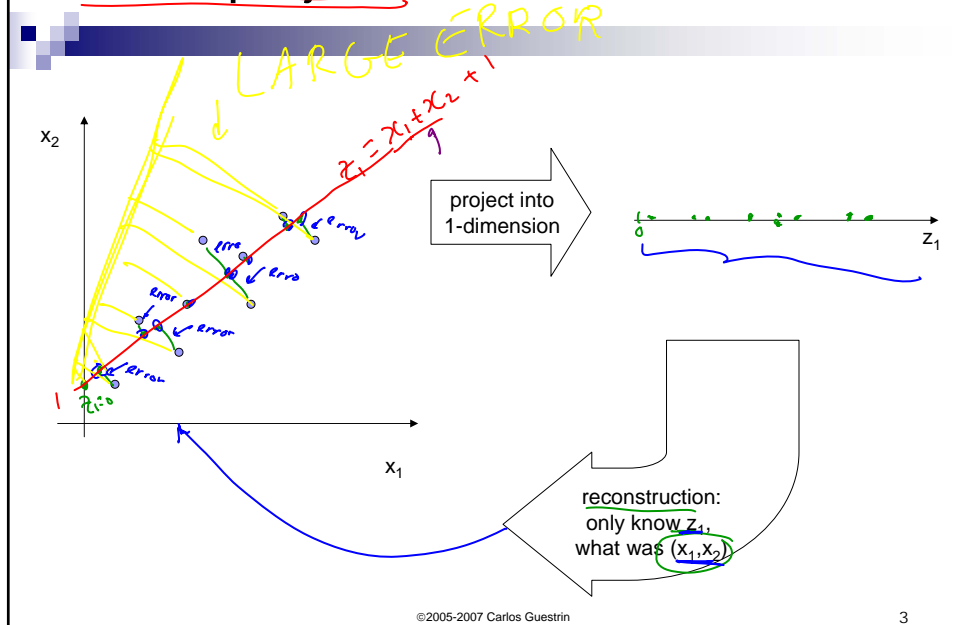
low. dim. proj.  
 $\tilde{x} = 0.1x_1 + 0.7x_2 - 0.35x_3 \dots$

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

©2005-2007 Carlos Guestrin

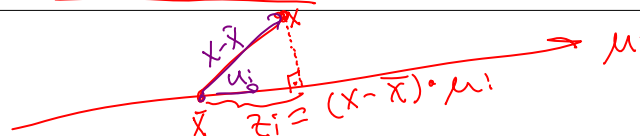
2

# Linear projection and reconstruction



# Linear projections, a review

- u · v ← dot product*
- Project a point into a (lower dimensional) space:
    - point:  $\mathbf{x} = (x_1, \dots, x_n)$   $\mathbf{x} \in \mathbb{R}^n$
    - select a basis – set of basis vectors –  $(\mathbf{u}_1, \dots, \mathbf{u}_k)$ 
      - 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:  $(z_1, \dots, z_k)$ ,  $z_i = (\mathbf{x} - \bar{\mathbf{x}}) \cdot \mathbf{u}_i$ 
      - minimum squared error



# PCA finds projection that minimizes reconstruction error

*k* ← num of basis vectors

- Given  $m$  data points:  $\mathbf{x}^i = (x_1^i, \dots, x_n^i)$ ,  $i=1 \dots m$
- Will represent each point as a projection:

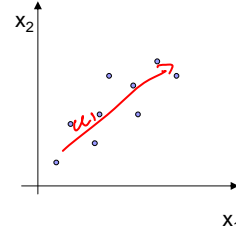
$$\hat{\mathbf{x}}^i = \bar{\mathbf{x}} + \sum_{j=1}^k z_j^i \mathbf{u}_j \quad \text{where: } \bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^m \mathbf{x}^i \quad \text{and} \quad z_j^i = (\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$$

*data point i* (pointing to  $\hat{\mathbf{x}}^i$ ), *centr.* (pointing to  $\bar{\mathbf{x}}$ ), *coords* (pointing to  $z_j^i$ ), *basis* (pointing to  $\mathbf{u}_j$ ), *mean* (pointing to  $\frac{1}{m} \sum_{i=1}^m \mathbf{x}^i$ ), *dot. previous slide* (pointing to  $(\mathbf{x}^i - \bar{\mathbf{x}}) \cdot \mathbf{u}_j$ )

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

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

*squared error*



# Understanding the reconstruction error

$(a+b)^2 = a^2 + ab + ba + b^2$   
*n basis orthonormal*

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

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

*equality* (pointing to the equation), *use same basis* (pointing to the equation)

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

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

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

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

$$(\mathbf{x}^i - \hat{\mathbf{x}}^i)^T (\mathbf{x}^i - \hat{\mathbf{x}}^i)$$

*$\mathbb{R}^n$*

- Rewriting error:  $\text{error}_k = \sum_{i=1}^m (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$

$$\begin{aligned} \text{error}_k &= \sum_{i=1}^m \left[ \bar{\mathbf{x}} + \sum_{j=1}^n z_j^i \mathbf{u}_j - \left( \bar{\mathbf{x}} + \sum_{j=1}^k z_j^i \mathbf{u}_j \right) \right]^2 \\ &= \sum_{i=1}^m \left( \sum_{j=k+1}^n z_j^i \mathbf{u}_j \right)^2 = \sum_{i=1}^m \left[ \underbrace{\left( \sum_{j=k+1}^n z_j^i \mathbf{u}_j \right)^T \left( \sum_{j=k+1}^n z_j^i \mathbf{u}_j \right)}_1 + \sum_{\substack{j=k+1 \\ l=k+1 \\ l \neq j}}^n \underbrace{z_j^i z_l^i \mathbf{u}_j^T \mathbf{u}_l}_{0} \right] \\ &= \sum_{i=1}^m \sum_{j=k+1}^n (z_j^i)^2 \leftarrow \text{sum of squared coeffs of dims we throw away} \end{aligned}$$

*ortho normal basis* (pointing to the 1 in the derivation)

# Reconstruction error and covariance matrix

$$\begin{aligned} (a \cdot b)^2 &= a^T b b^T a \\ a \cdot b &= a^T b = b^T a \end{aligned}$$

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

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

$$= \sum_{j=k+1}^n \sum_{i=1}^m u_j^T (x^i - \bar{x}) (x^i - \bar{x})^T u_j$$

$$= \sum_{j=k+1}^n u_j^T \left[ \sum_{i=1}^m (x^i - \bar{x}) (x^i - \bar{x})^T \right] u_j$$

$$= m \sum_{j=k+1}^n u_j^T \Sigma u_j$$

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

covariance matrix

# Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis  $(u_1, \dots, u_n)$  minimizing:

$$error_k = m \sum_{j=k+1}^n u_j^T \Sigma u_j$$

- Eigen vector:

$$error_k = m \sum_{j=k+1}^n \lambda_j$$

$\lambda_j$  is jth smallest eigen value

eigen vectors

$$Av = \lambda v$$

$$v^T Av = \lambda v^T v$$

$$v^T Av = \lambda$$

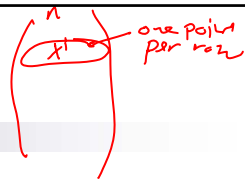
- Minimizing reconstruction error equivalent to picking  $(u_{k+1}, \dots, u_n)$  to be eigen vectors with smallest eigen values

"smallest" eigen vectors

$u_1, \dots, u_k \leftarrow$   
keep (use)

k eigen vectors of  $\Sigma$  with highest eigen value

# Basic PCA algorithm

$x = m$  

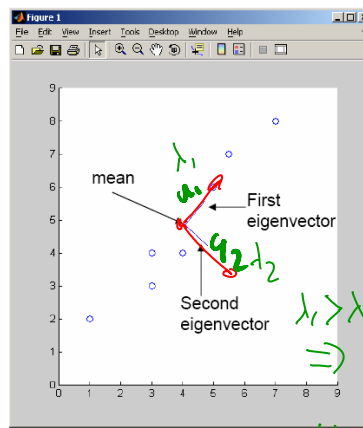
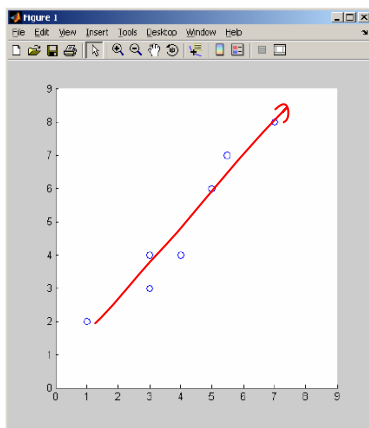
- Start from  $m$  by  $n$  data matrix  $X$
- Recenter: subtract mean from each row of  $X$ 
  - $X_c \leftarrow X - \bar{X}$
- Compute covariance matrix:
  - $\hat{\Sigma} \leftarrow 1/m X_c^T X_c$
- Find eigen vectors and values of  $\Sigma$
- Principal components:  $k$  eigen vectors with highest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

*eigs (Matlab)*

# PCA example

*So keep  $u_1$  if you keep  $u_1$ , error  $\lambda_2$   
if you keep  $u_2$ , error  $\lambda_1$   
because  $\lambda_1 > \lambda_2$*

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

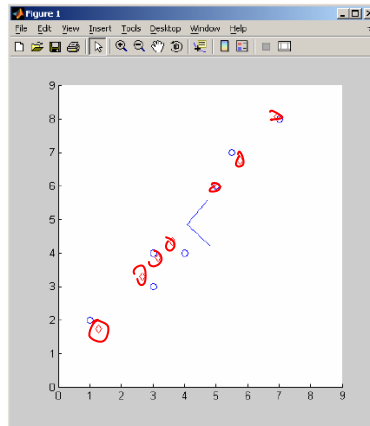
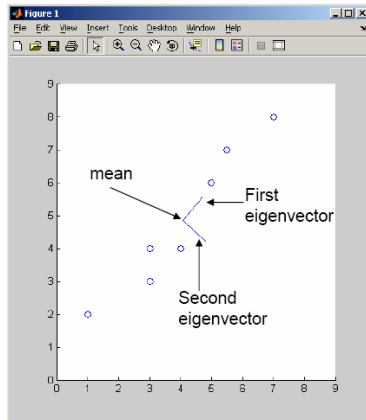


*$\lambda_1 > \lambda_2$   
 $\Rightarrow$  keep  $u_1$   
throw away  $u_2$*

# PCA example – reconstruction

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

only used first principal component



# Eigenfaces [Turk, Pentland '91]

■ Input images:



■ Principal components:



$$X = \begin{pmatrix} \text{Face} \end{pmatrix}$$

## Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:



## Scaling up

- Covariance matrix can be really big!
  - $\Sigma$  is n by n
  - 10000 features  $|\Sigma| \leftarrow 10000 \times 10000 = 100 \text{ million entries}$
  - finding eigenvectors is very slow...  
*stability*
- Use singular value decomposition (SVD)
  - finds to k eigenvectors
  - great implementations available, e.g., Matlab svd

# SVD

- Write  $X = W S V^T$

- $X$  ← data matrix, one row per datapoint
- $W$  ← weight matrix, one row per datapoint – coordinate of  $x^i$  in eigenspace
- $S$  ← singular value matrix, diagonal matrix
  - in our setting each entry is eigenvalue  $\lambda_j$
- $V^T$  ← singular vector matrix
  - in our setting each row is eigenvector  $v_j$

$$\begin{matrix} n \\ m \end{matrix} \begin{pmatrix} X \end{pmatrix} = \begin{matrix} m \\ k \end{matrix} \begin{pmatrix} W \end{pmatrix} \begin{pmatrix} \sigma_1 & & \\ & \dots & \\ & & \sigma_k \end{pmatrix} \begin{pmatrix} V^T \\ \vdots \\ v_i \\ \vdots \end{pmatrix}$$

$v_i$   
 singular  
 (Principle)  
 Component

# PCA using SVD algorithm

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

Coefficients become:

$$z_j^i = (x^i - \bar{x}) \cdot v_j$$

or read them from  $S$  and  $W$  matrices



$$X = \begin{pmatrix} \vdots \\ \vdots \\ \vdots \end{pmatrix} \\
 \bar{X} = \begin{pmatrix} \bar{x}_1 \\ \vdots \\ \bar{x}_n \end{pmatrix}$$

average of each column



# Markov Decision Processes (MDPs)

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

November 28<sup>th</sup>, 2007

19

## Thus far this semester

■ Regression:  $f: \mathcal{X} \rightarrow \mathbb{R}$  <sup>data</sup>  $\langle x, t(x) \rangle$

■ Classification:  $f: \mathcal{X} \rightarrow \{1, \dots, k\}$

■ Density estimation:  $f: \mathcal{X} \rightarrow [0, 1]$   
 $\int_{\mathcal{X}} f(x) dx = 1$

# Learning to act



[Ng et al. '05]

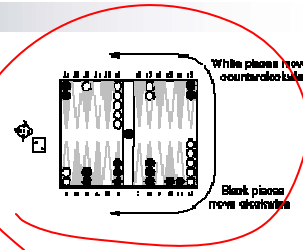
$\langle x, a \rangle$  ← data not like this

- 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]

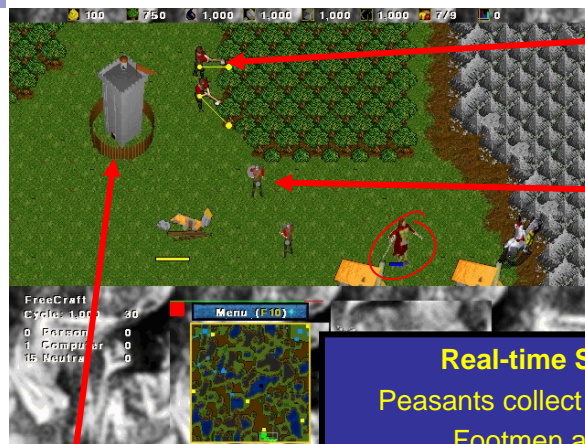
- Combines reinforcement learning with neural networks
- Played 300,000 games against itself
- Achieved grandmaster level!



# 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  $\mathbf{x}$  of entire system

- Action space:

- Joint action  $\mathbf{a} = \{a_1, \dots, a_n\}$  for all agents



# States change over time

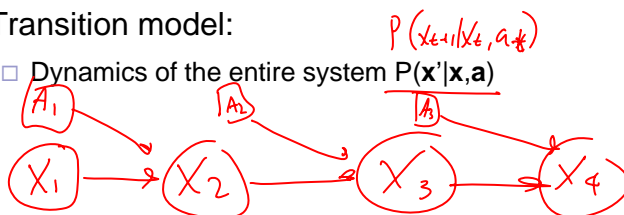
- Like an HMM, state changes over time

- 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  $P(\mathbf{x}'|\mathbf{x}, \mathbf{a})$



## Some states and actions are better than others

- Each state  $\mathbf{x}$  is associated with a reward

- positive reward for successful attack
- negative for loss



- Reward function:

- Total reward  $R(\mathbf{x})$

*can also be a function of action*  
 $R(\mathbf{x}, \mathbf{a}) \rightarrow \mathbb{R}$

## Markov Decision Process (MDP) Representation

- State space:

- Joint state  $\mathbf{x}$  of entire system

- Action space:

- Joint action  $\mathbf{a} = \{a_1, \dots, a_n\}$  for all agents

- Reward function:

- Total reward  $R(\mathbf{x}, \mathbf{a})$ 
  - sometimes reward can depend on action



- Transition model:

- Dynamics of the entire system  $P(\mathbf{x}'|\mathbf{x}, \mathbf{a})$

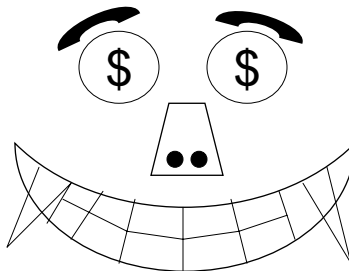
# Discounted Rewards

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

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

$$20 + 20 + 20 + 20 + 20 + \dots = \text{Infinity}$$

*inflation  
death*



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.

$\gamma =$

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** ?

$$20 + \gamma 20 + \gamma^2 20 + \gamma^3 20 + \dots$$
$$= \frac{20}{1-\gamma}$$

# Discount Factors

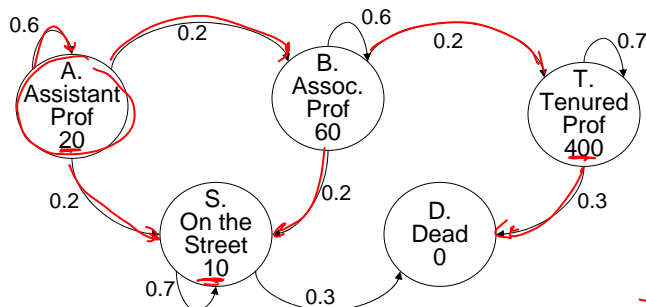
People in economics and probabilistic decision-making do this all the time.

The “Discounted sum of future rewards” using discount factor  $\gamma$  is

$$\begin{aligned}
 & \text{(reward now) +} \\
 & \gamma \text{ (reward in 1 time step) +} \\
 & \gamma^2 \text{ (reward in 2 time steps) +} \\
 & \gamma^3 \text{ (reward in 3 time steps) +} \\
 & \quad \vdots \\
 & \quad \vdots \text{ (infinite sum)}
 \end{aligned}$$

# The Academic Life

Assume Discount Factor  $\gamma = 0.9$



$$\begin{aligned}
 & V_D = 0 \\
 & V_T = 400 \\
 & \quad + 0.7 \gamma V_T \\
 & \quad + 0.3 V_D \quad \text{if } \gamma = 0 \\
 & V_T = \frac{400}{1 - 0.7\gamma}
 \end{aligned}$$

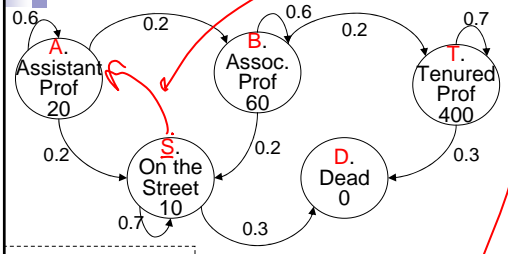
Define:

- $V_A$  = 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

*True solution if as long as  $\gamma < 1$  exist & unique*



Assume Discount Factor  $\gamma = 0.9$

*Set of linear equations (matrix inversion)*

$$V_D = 0$$

$$V_T = 400 + 0.7\gamma V_T + 0.3\gamma V_D$$

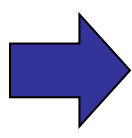
$$V_S = 10 + 0.7\gamma V_S + 0.3\gamma V_D$$

$$V_B = 60 + 0.6\gamma V_B + 0.2\gamma V_T + 0.2\gamma V_S$$

$$V_A = 20 + 0.6\gamma V_A + 0.2\gamma V_B + 0.2\gamma V_S$$

# Policy

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

# Value of Policy

always exist a optimal deterministic policy

Value:  $V_{\pi}(\mathbf{x})$

➔

Expected long-term reward starting from  $\mathbf{x}$

Start from  $\mathbf{x}_0$   $V_{\pi}(\mathbf{x}_0)$

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

Future rewards discounted by  $\gamma \in [0, 1)$

©2005-2007 Carlos Guestrin 35

# Computing the value of a policy

linearity of expectations  $E[a+b] = E[a] + E[b]$

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

- Discounted value of a state:
  - value of starting from  $\mathbf{x}_0$  and continuing with policy  $\pi$  from then on
$$V_{\pi}(\mathbf{x}_0) = \mathbf{E}_{\pi} [R(\mathbf{x}_0) + \gamma R(\mathbf{x}_1) + \gamma^2 R(\mathbf{x}_2) + \gamma^3 R(\mathbf{x}_3) + \dots]$$

$$= \mathbf{E}_{\pi} \left[ \sum_{t=0}^{\infty} \gamma^t R(\mathbf{x}_t) \right]$$
- A recursion!
 
$$V_{\pi}(\mathbf{x}_0) = \mathbf{E}_{\pi} [R(\mathbf{x}_0) + \gamma R(\mathbf{x}_1) + \gamma^2 R(\mathbf{x}_2) + \dots]$$

$$= \underbrace{\mathbf{E}_{\pi} [R(\mathbf{x}_0)]}_{R(\mathbf{x}_0)} + \gamma \underbrace{\mathbf{E}_{\pi} [R(\mathbf{x}_1) + \gamma R(\mathbf{x}_2) + \gamma^2 R(\mathbf{x}_3) + \dots]}_{\mathbf{E}_{\pi, \pi} [V_{\pi}(\mathbf{x}_1)]}$$

$$= R(\mathbf{x}_0) + \gamma \sum_{\mathbf{x}_1} P(\mathbf{x}_1 | \mathbf{x}_0, \pi(\mathbf{x}_0)) V_{\pi}(\mathbf{x}_1)$$

©2005-2007 Carlos Guestrin 36

# Simple approach for computing the value of a policy: Iteratively

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

- Can solve using a simple convergent iterative approach: (a.k.a. dynamic programming)

- Start with some guess  $V_0$

*any guess works, but a good guess is  $V_0(x) = R(x_0)$*

- Iteratively say:

- $V_{t+1} = R + \gamma P_{\pi} V_t$

*$V_{t+1}(x) = R(x) + \gamma \sum_{x'} P(x'|x, \pi(x)) V_t(x')$*

- Stop when  $\|V_{t+1} - V_t\| \leq \epsilon$

- means that  $\|V_{\pi} - V_{t+1}\| \leq \epsilon / (1 - \gamma)$

# But we want to learn a Policy

- So far, told you how good a policy is...  $V_{\pi}(x)$

- 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!

*for state  $x$*

$$a^* = \arg \max_a R(x) + \sum_{x'} P(x'|x, a) R(x')$$

