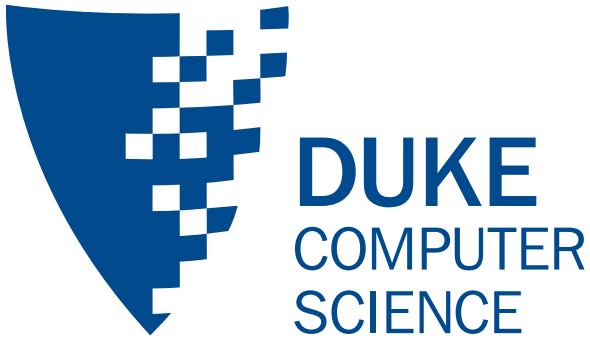
Unsupervised Learning

George Konidaris
gdk@cs.duke.edu



Spring 2015

Machine Learning

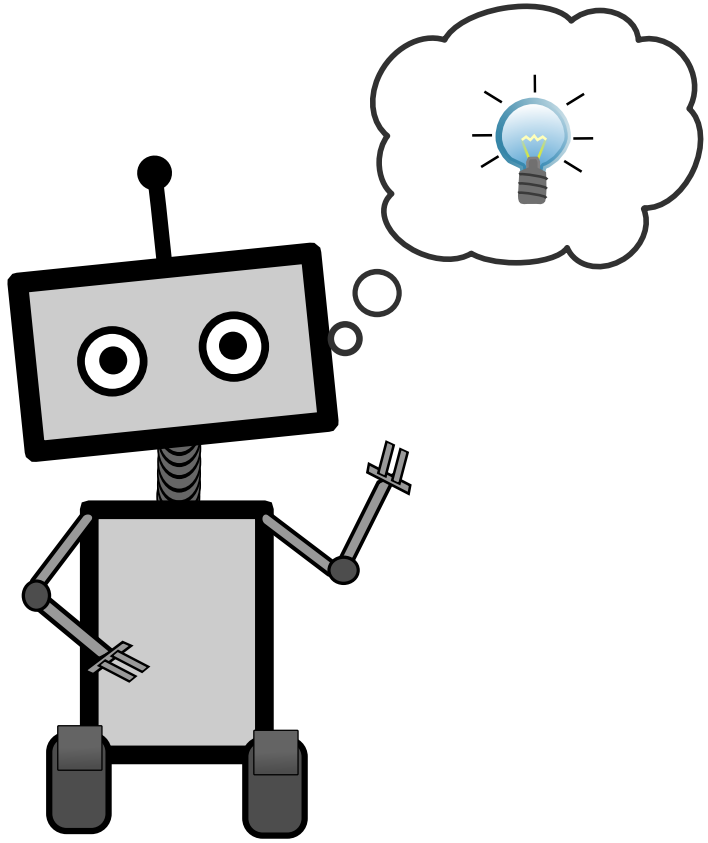
DUKE COMPUTER SCIENCE

Subfield of AI concerned with *learning from data*.

Broadly, using:

- ***Experience***
- To Improve ***Performance***
- On Some ***Task***

(Tom Mitchell, 1997)



Unsupervised Learning

Input:

$X = \{x_1, \ldots, x_n\}$ inputs

Try to understand the *structure of the data.*

E.g., how many types of cars?
How can they vary?

Dimensionality Reduction

DUKE
COMPUTER
SCIENCE

$X = \{x_1, \ldots, x_n\}$

If *n* is high, data can be hard to deal with.

- High-dimensional decision boundary.
- Need more data.
- But data is often not really high-dimensional.

Dimensionality reduction:

- Reduce or compress the data
- Try not to lose too much!
- Find intrinsic dimensionality

Dimensionality Reduction

DUKE
COMPUTER
SCIENCE

For example, imagine if x_1 and x_2 are meaningful features, and x_3 … x_n are random noise.

What happens to k-nearest neighbors?

What happens to a decision tree?

What happens to the perceptron algorithm?

What happens if you want to do clustering?

Dimensionality Reduction

DUKE
COMPUTER
SCIENCE

Often can be phrased as a projection:

$$f : X \rightarrow X'$$

where:

- $|X'| << |X|$
- our goal: retain as much *variance* as possible.

Variance captures what *varies within the data*.

PCA

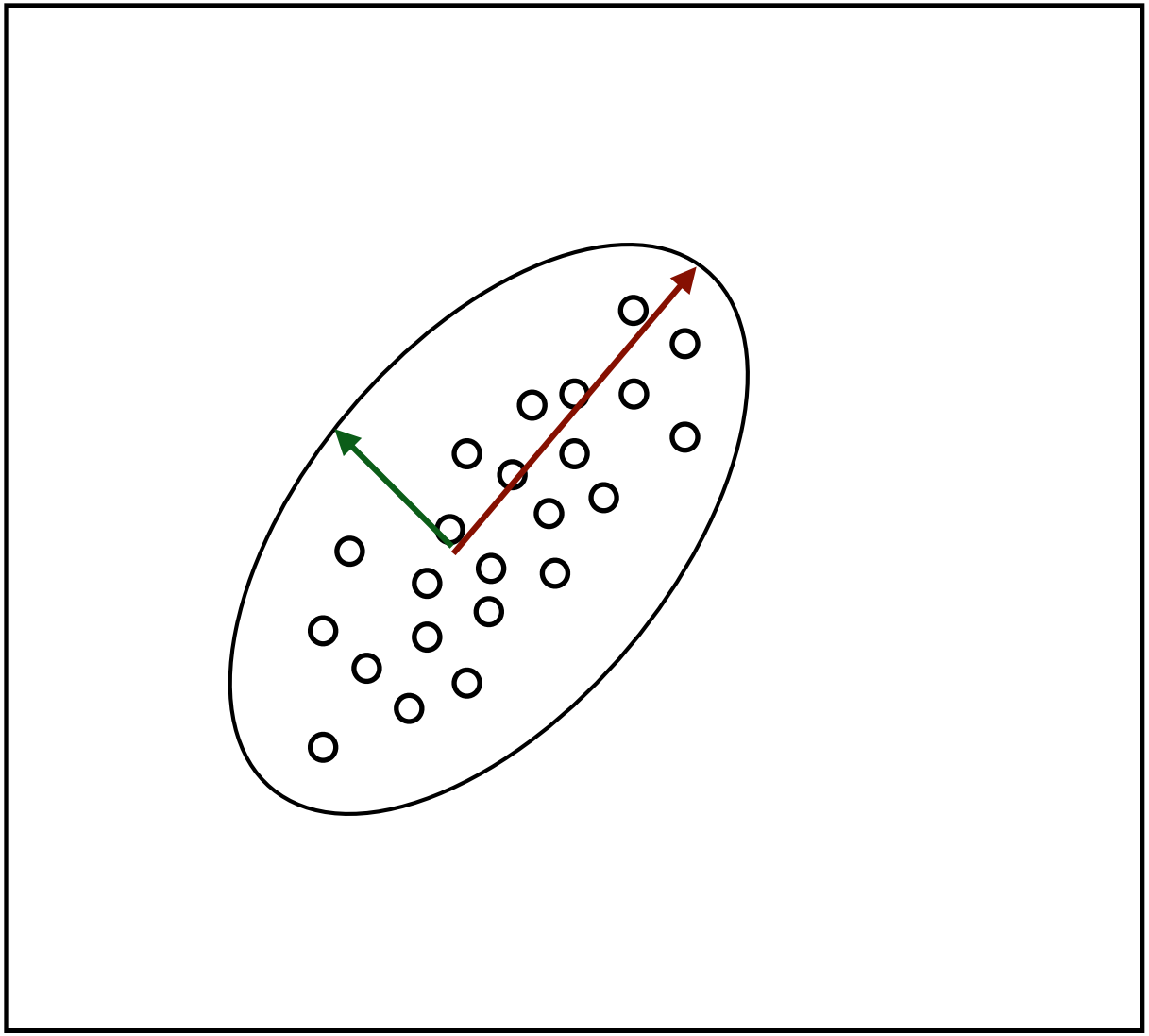
DUKE
COMPUTER
SCIENCE

Principle Components Analysis.

Project data into a new space:

- Dimensions are linearly uncorrelated.
- We have a measure of importance for each dimension.

PCA



PCA

- Gather data $X_1, \ldots, X_m$.
- Adjust data to be zero-mean:

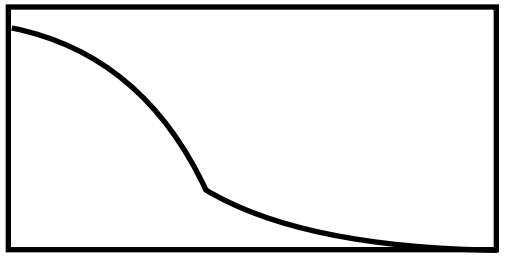
$$X_i = X_i - \sum_j \frac{X_j}{m}$$

- Compute covariance matrix C.
- Compute unit eigenvectors V_i and eigenvalues v_i of C.

Each V_i is a direction, and each v_i is its importance - the amount of the data's variance it accounts for.

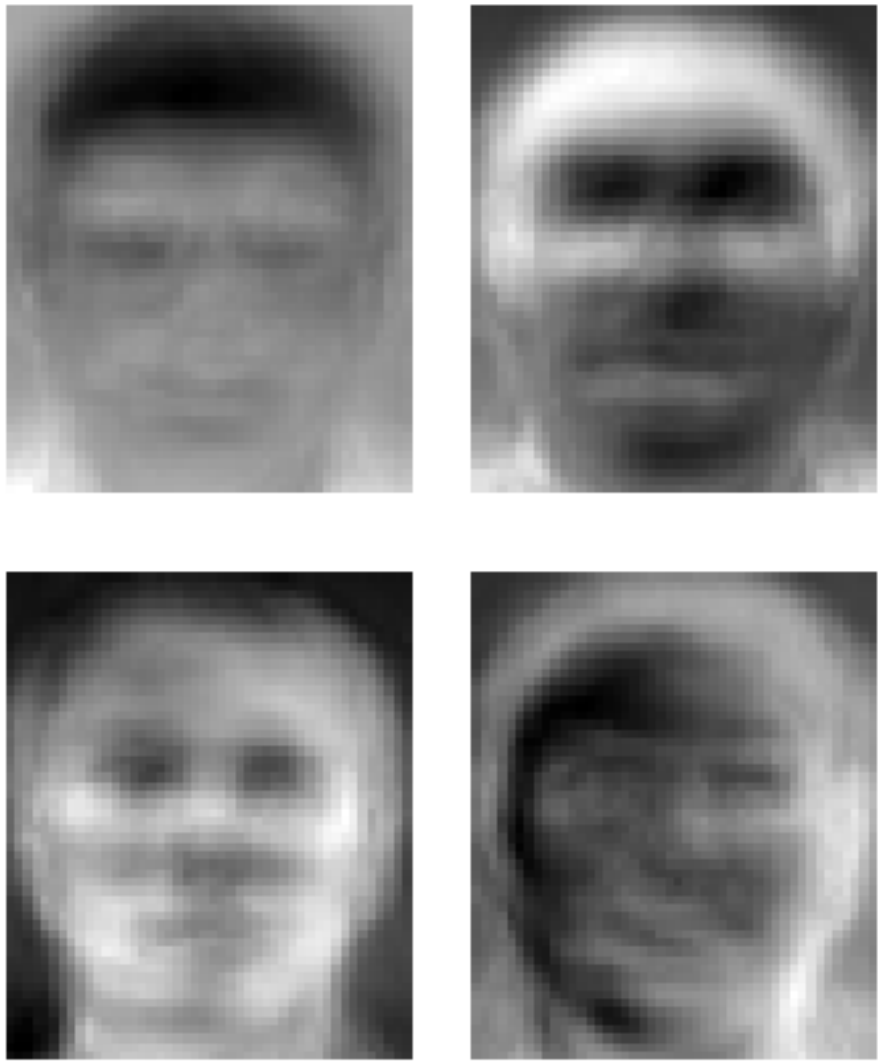
New data points:

$$\hat{X}_i = [V_1, ..., V_p]X_i$$



Eigenfaces

DUKE
COMPUTER
SCIENCE



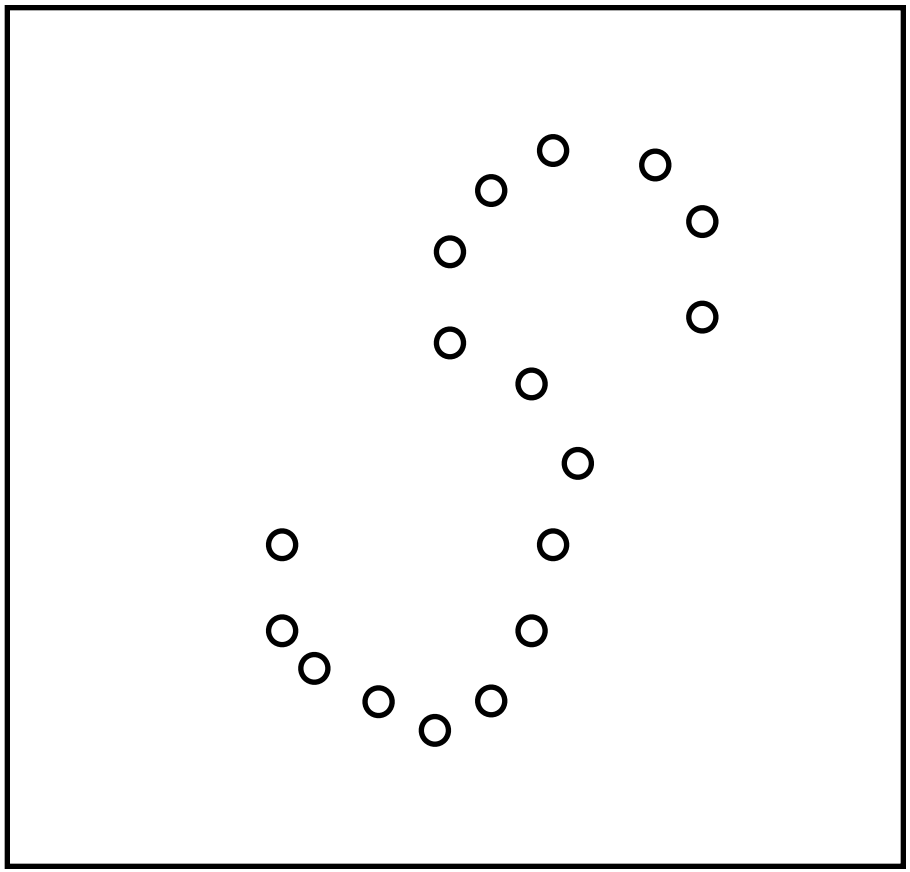
(courtesy ORL database)

ISOMAP

DUKE COMPUTER SCIENCE

Another approach:

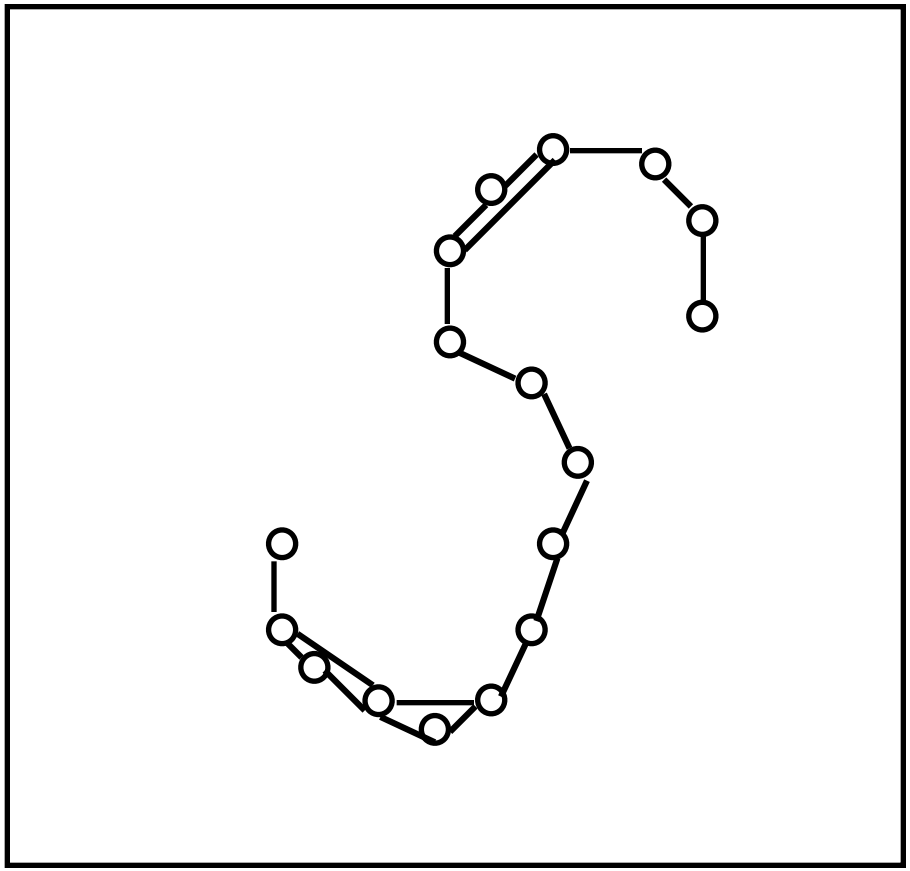
- Estimate intrinsic geometric dimensionality of data.
- Recover natural distance metric



ISOMAP

Core idea: distance metric *locally Euclidean*

- Small radius *r*, connect each point to neighbors
- Weight based on Euclidean distance

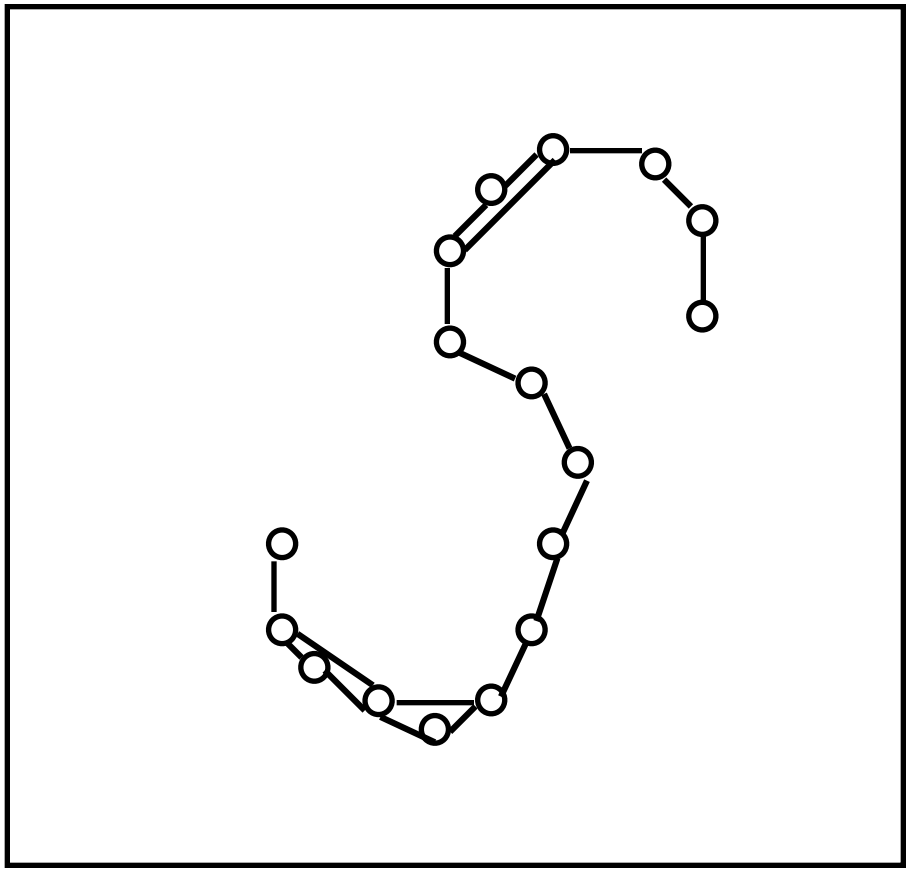


ISOMAP

DUKE COMPUTER SCIENCE

Solve all-points shortest pairs:

- Transforms local distance to global distance.
- Compute embedding.



ISOMAP

DUKE COMPUTER SCIENCE

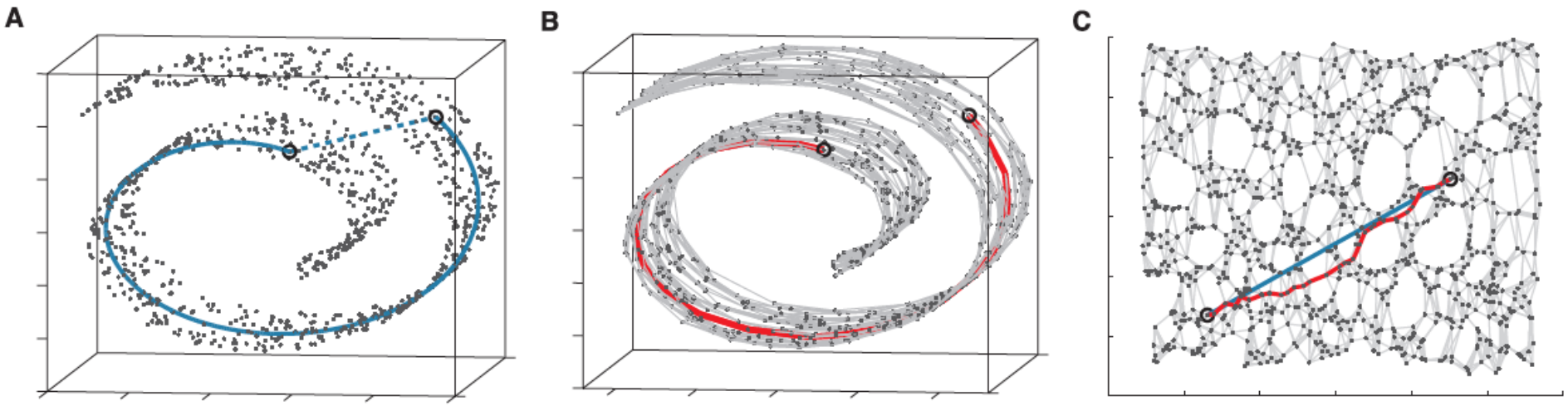


Fig. 3. The "Swiss roll" data set, illustrating how Isomap exploits geodesic paths for nonlinear dimensionality reduction. (**A**) For two arbitrary points (circled) on a nonlinear manifold, their Euclidean distance in the high-dimensional input space (length of dashed line) may not accurately reflect their intrinsic similarity, as measured by geodesic distance along the low-dimensional manifold (length of solid curve). (**B**) The neighborhood graph G constructed in step one of Isomap (with $K = 7$ and $N = 1000$ data points) allows an approximation (red segments) to the true geodesic path to be computed efficiently in step two, as the shortest path in G. (**C**) The two-dimensional embedding recovered by Isomap in step three, which best preserves the shortest path distances in the neighborhood graph (overlaid). Straight lines in the embedding (blue) now represent simpler and cleaner approximations to the true geodesic paths than do the corresponding graph paths (red).

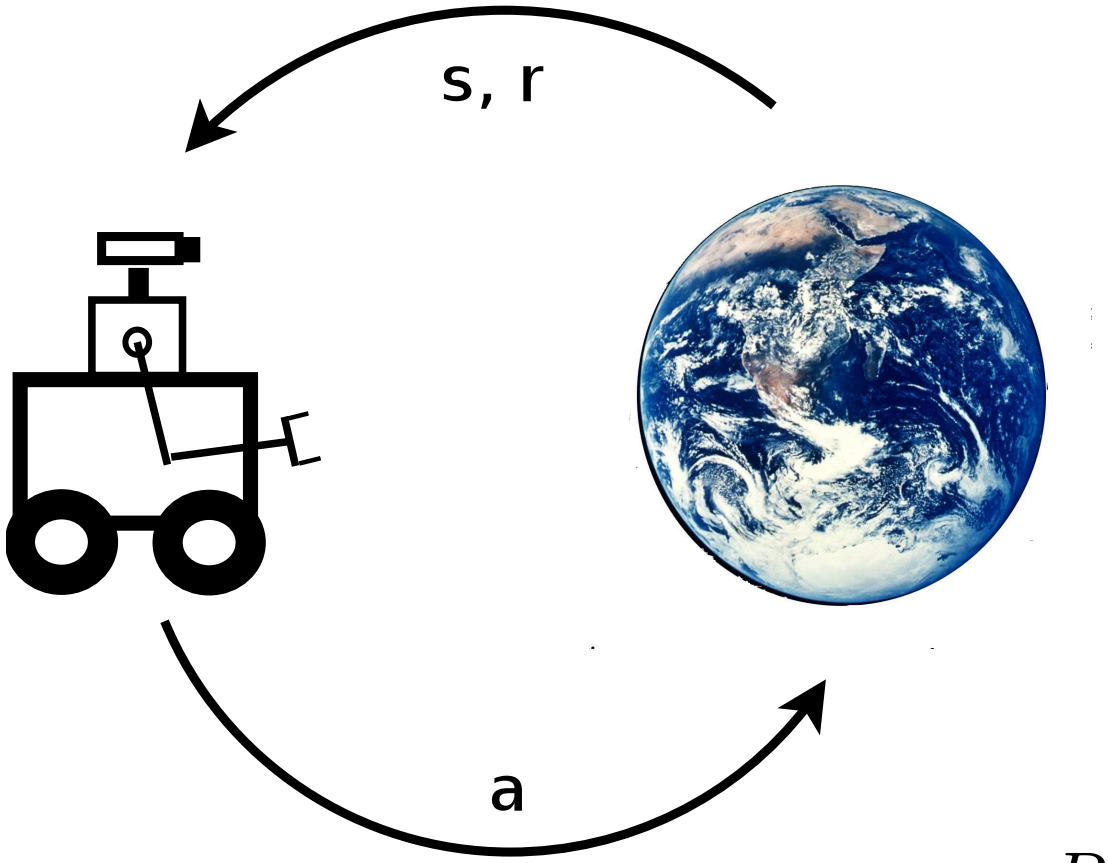
From Tenenbaum, de Silva, and Langford, *Science* 290:2319-2323, December 2000.

DUKE
COMPUTER
SCIENCE

Reinforcement Learning

Reinforcement Learning

Learning counterpart of planning.

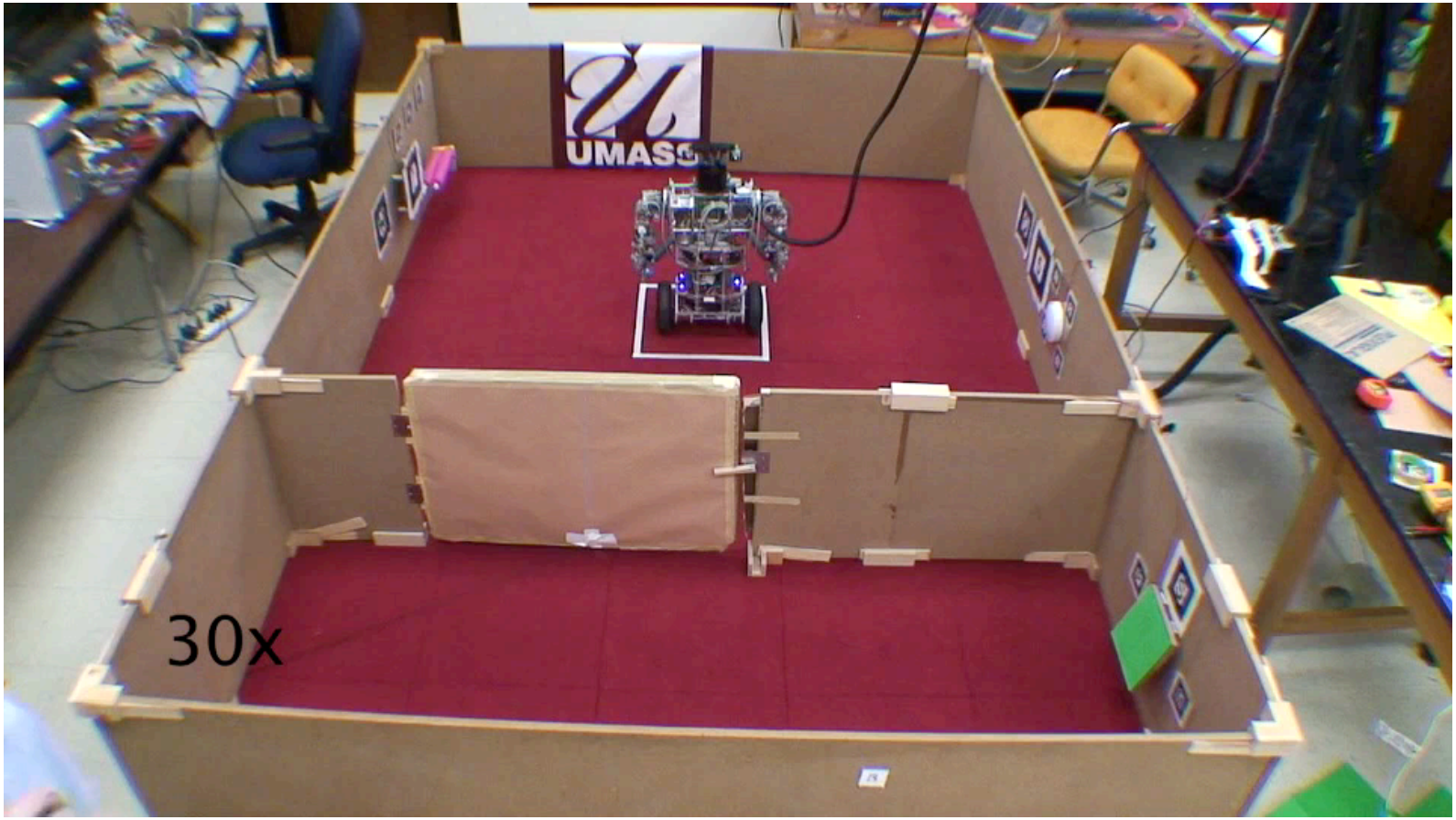


$\pi : S \rightarrow A$

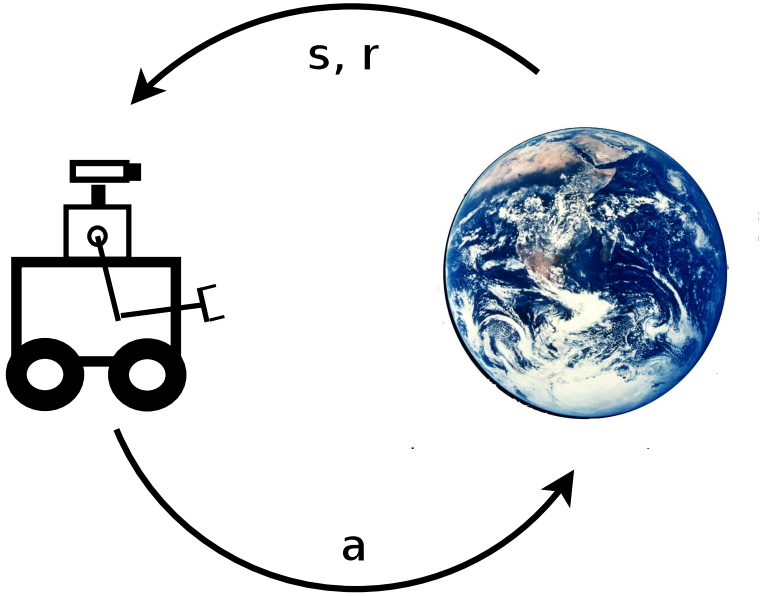
$$\max_{\pi} R = \sum_{t=0}^{\infty} \gamma^t r_t$$

RL

The problem of **learning how to interact** with an environment to **maximize reward**.



RL



Agent interacts with an environment

At each time t:

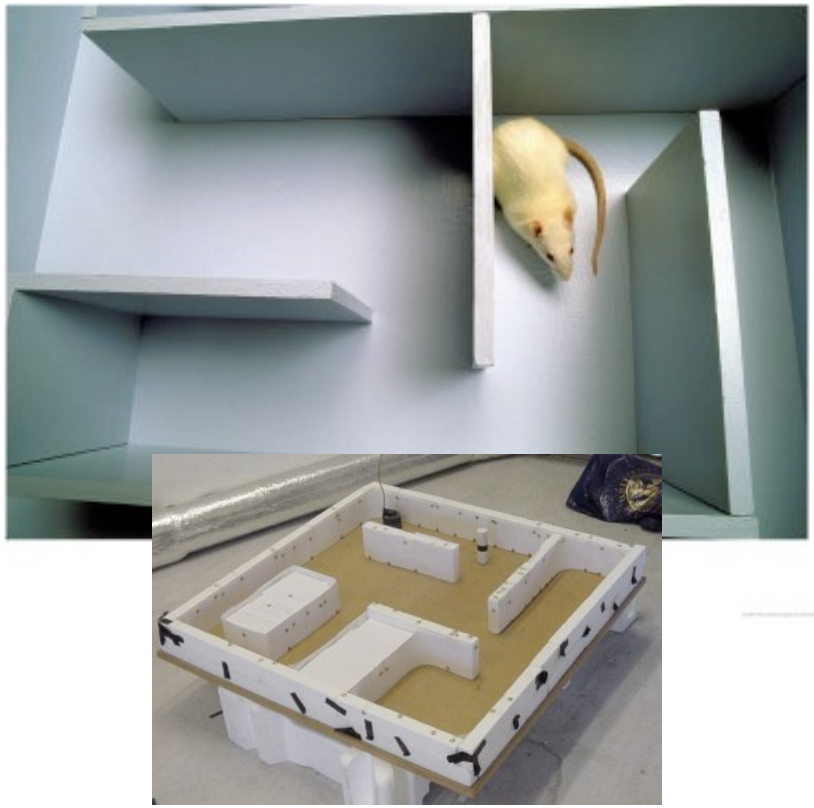
- Receives sensor signal s_t
- Executes action a_t
- *Transition*:
  - new sensor signal s_{t+1}
  - reward r_t

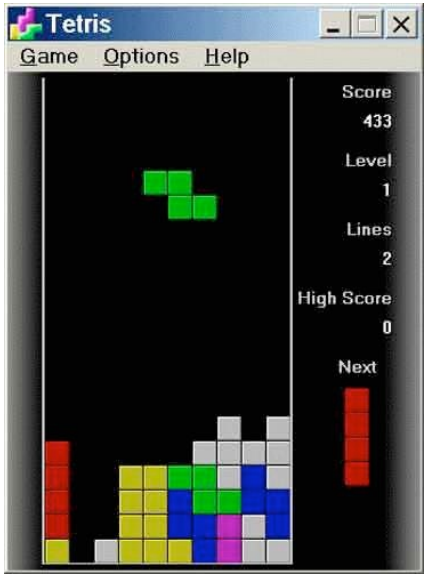
Goal: find policy π that maximizes expected return (sum of discounted future rewards):

$$\max_{\pi} \mathbb{E}\left[R = \sum_{t=0}^{\infty} \gamma^t r_t\right]$$

RL

This formulation is general enough to encompass a wide variety of learned control problems.





Markov Decision Processes

DUKE COMPUTER SCIENCE

S : set of *states*
A : set of actions
γ : discount factor

$$< S, A, \gamma, R, T >$$

R : reward function

$R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'.

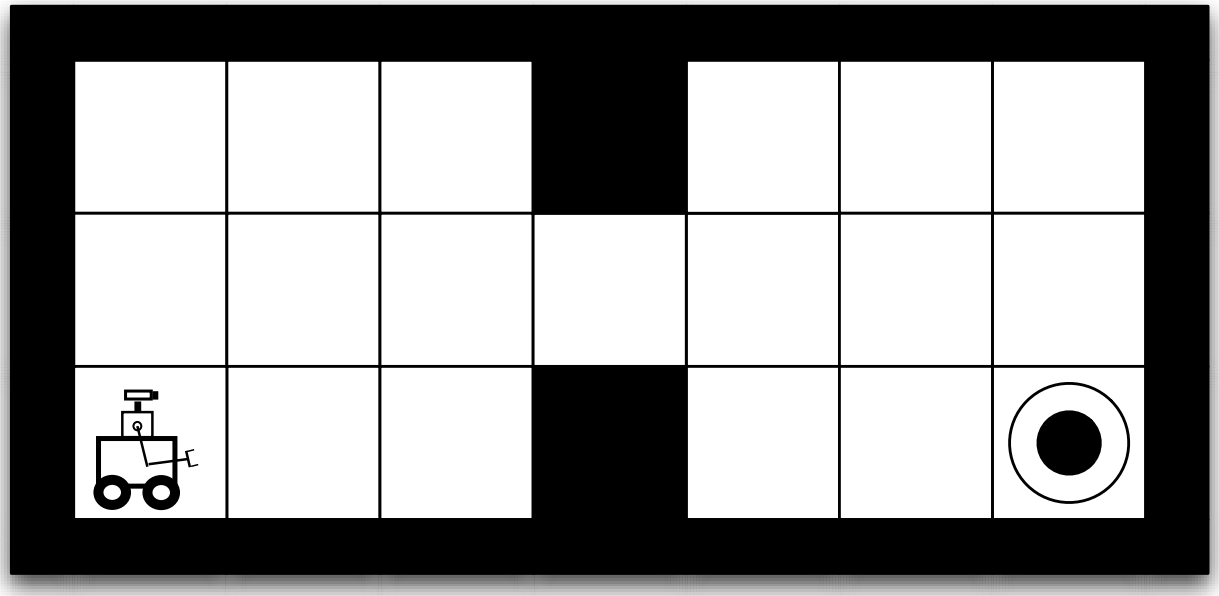
T : transition function

$T(s'|s, a)$ is the probability of transitioning to state s' after taking action a in state s.

RL: *one or both of T, R unknown.*

RL

Example:



States: set of grid locations

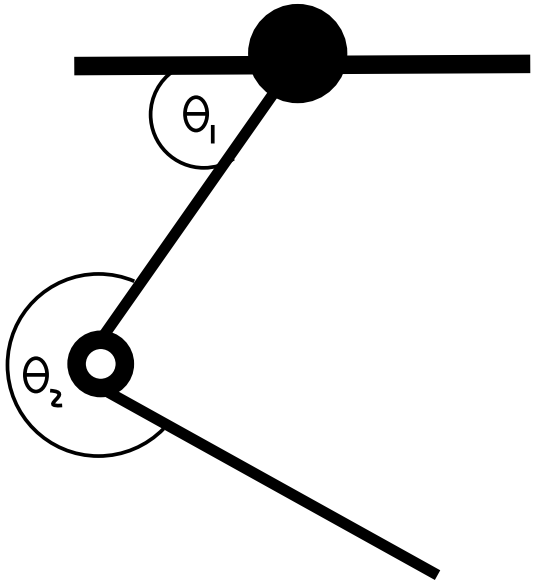
Actions: up, down, left, right

Transition function: move in direction of action with p=0.9

Reward function: -1 for every step, 1000 for finding the goal

RL

Example:



States: $(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ (real-valued vector)

Actions: +1, -1, 0 units of torque added to elbow

Transition function: physics!

Reward function: -1 for every step

MDPs

DUKE
COMPUTER
SCIENCE

Our target is a **policy**:

$$\pi : S \rightarrow A$$

A policy maps *states* to *actions*.

The **optimal policy** maximizes:

$$\max_{\pi} \forall s, \mathbb{E}\left[R(s) = \sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s\right]$$

This means that we wish to find a policy that maximizes the **return** from **every state.**

Value Functions

DUKE
COMPUTER
SCIENCE

Given a policy, we can estimate of $R(s)$ for every state.

- This is a *value function*.
- It can be used to improve our policy.

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s\right]$$

This is the value of state s **under policy** π.

Value Functions

DUKE
COMPUTER
SCIENCE

Similarly, we define a *state-action value function* as follows:

$$Q_\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| \pi, s_0 = s, a_0 = a\right]$$

This is the value of executing a in state s, *then following* π.

Note that:

$$Q_\pi(s, \pi(s)) = V_\pi(s)$$

Policy Iteration

DUKE
COMPUTER
SCIENCE

Recall that we seek the policy that maximizes $V_\pi(s), \forall s$.

Therefore we know that, for the optimal policy π^*:

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

$$Q_{\pi^*}(s, a) \geq Q_\pi(s, a), \forall \pi, s, a$$

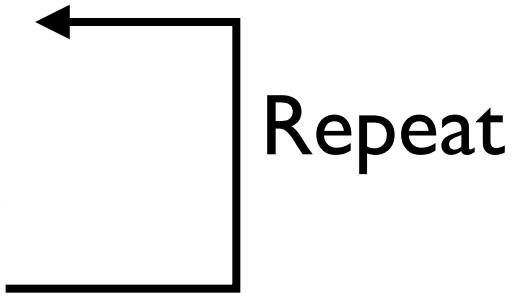
This means that any change to π that increases Q anywhere obtains a better policy.

Policy Iteration

DUKE
COMPUTER
SCIENCE

This leads to a general policy improvement framework:

1. Start with a policy π
2. Learn Q_π
3. Improve π
   a. $\pi(s) = \max_a Q(s, a), \forall s$



This is known as **policy iteration**.
It is guaranteed to converge to the optimal policy.

Steps 2 and 3 can be interleaved as rapidly as you like.
Usually, perform 3a *every time step*.

Friday

How to learn Q!