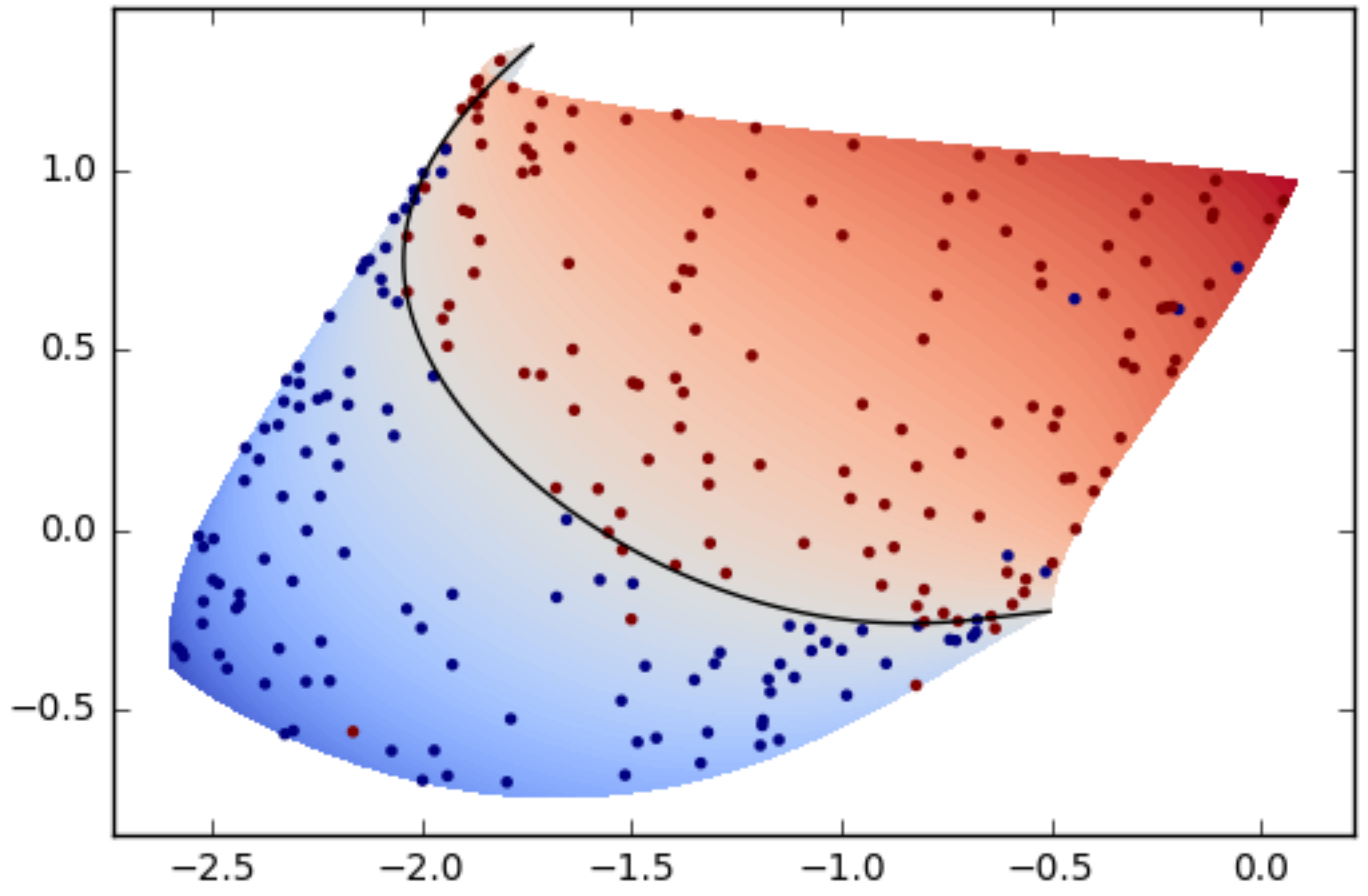
CS 273A: Machine Learning

Fall 2021

Lecture 16: Active and Online Learning

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

All slides in this course adapted from Alex Ihler & Sameer Singh



Logistics

assignments

- Assignment 5 due Tuesday, Nov 30

project

- Final report due next Thursday, Dec 2

final exam

- Review: next Thursday, Dec 2
- Final: Tuesday, Dec 7, 10:30am–12:30

Today's lecture

Latent-space models

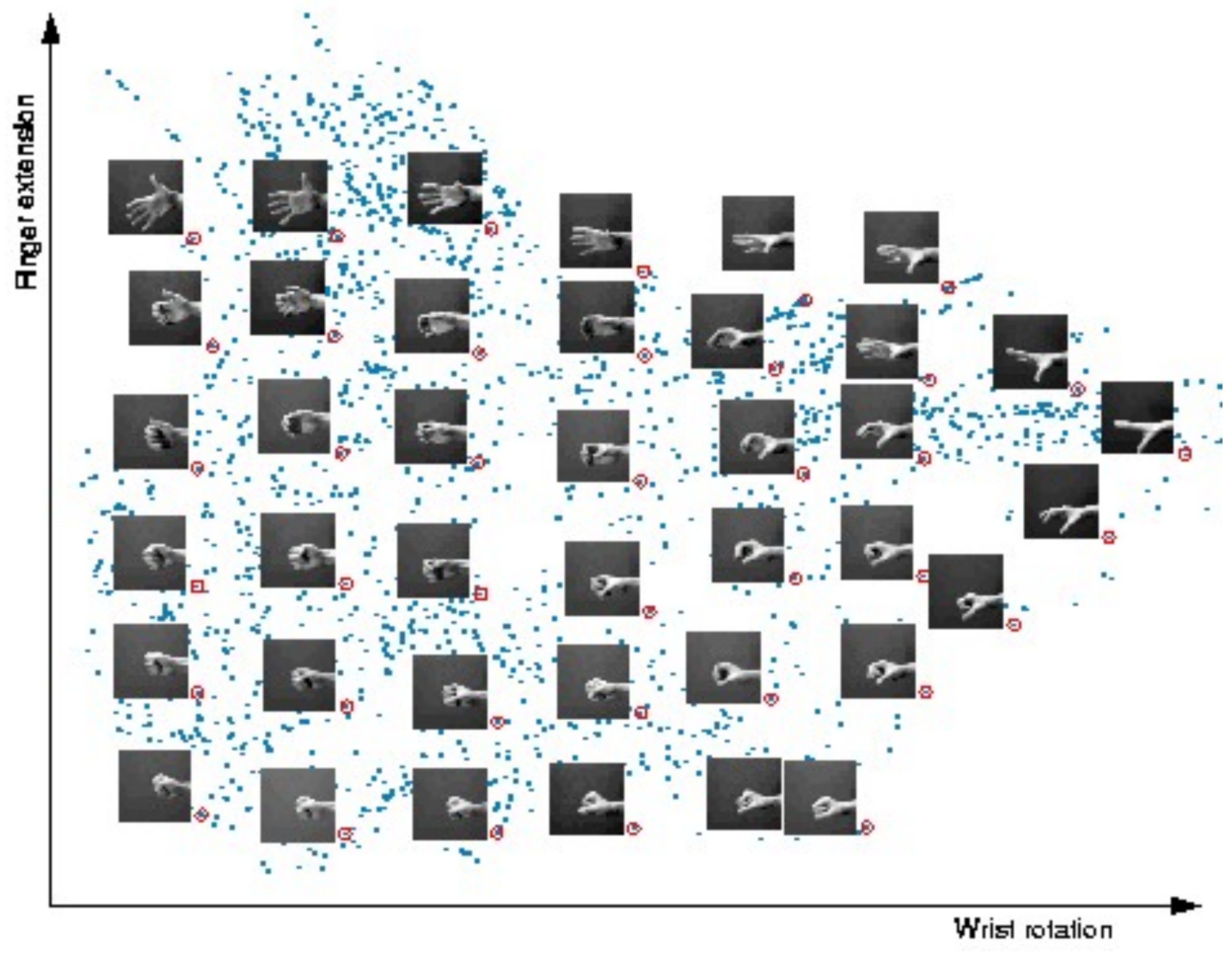
Active learning

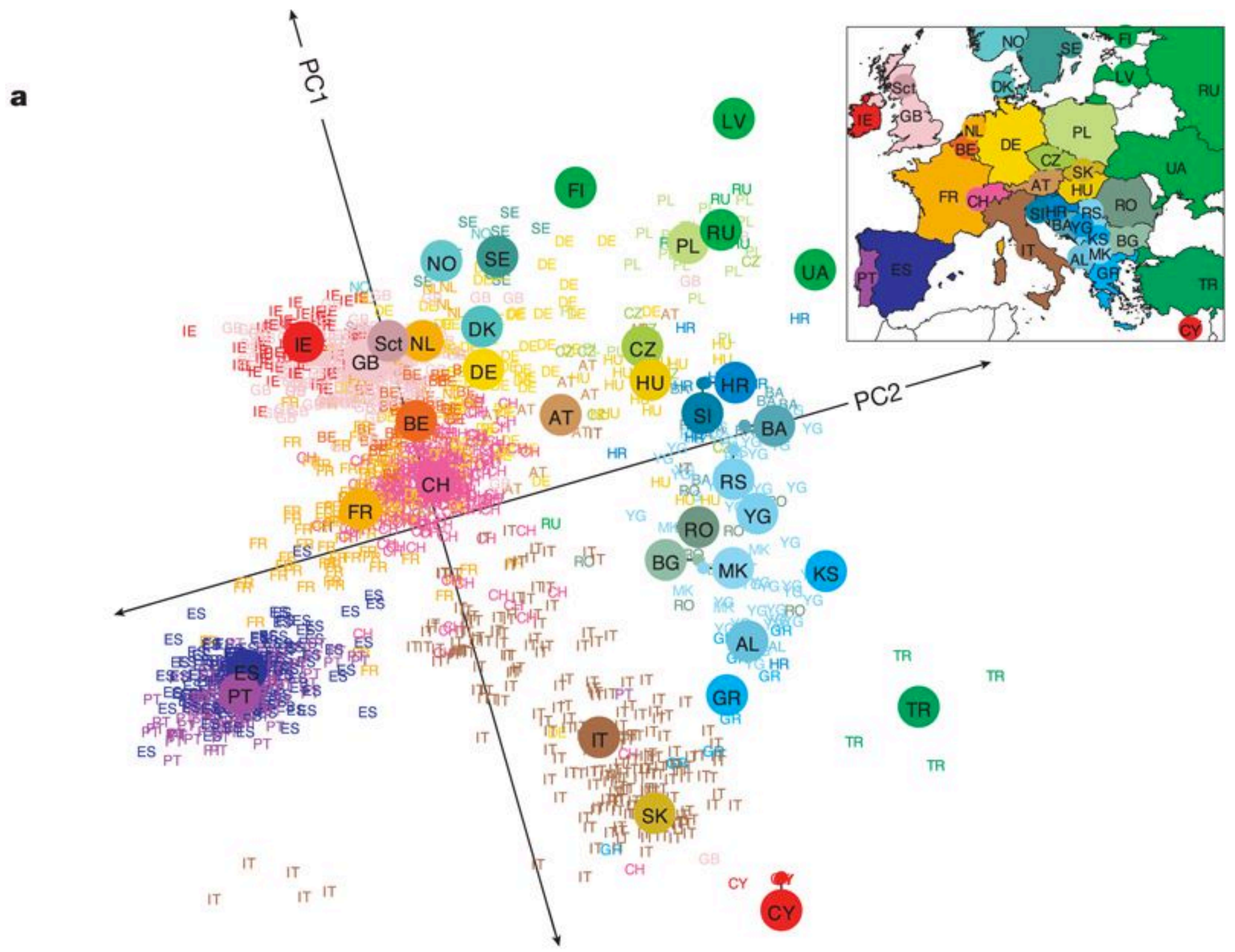
Online learning

Sequential decision making

Why reduce dimensionality?

- Data is often high-dimensional = many features
  - Images (even at 28x28 pixels)
  - Text (even a “bag of words”)
  - Stock prices (e.g. S&P500)
- Issues with high-dimensionality:
  - Computational complexity of analyzing the data
  - Model complexity (more parameters)
  - Sparse data = cannot cover all combinations of features
  - Correlated features can be independently noisy
  - Hard to visualize





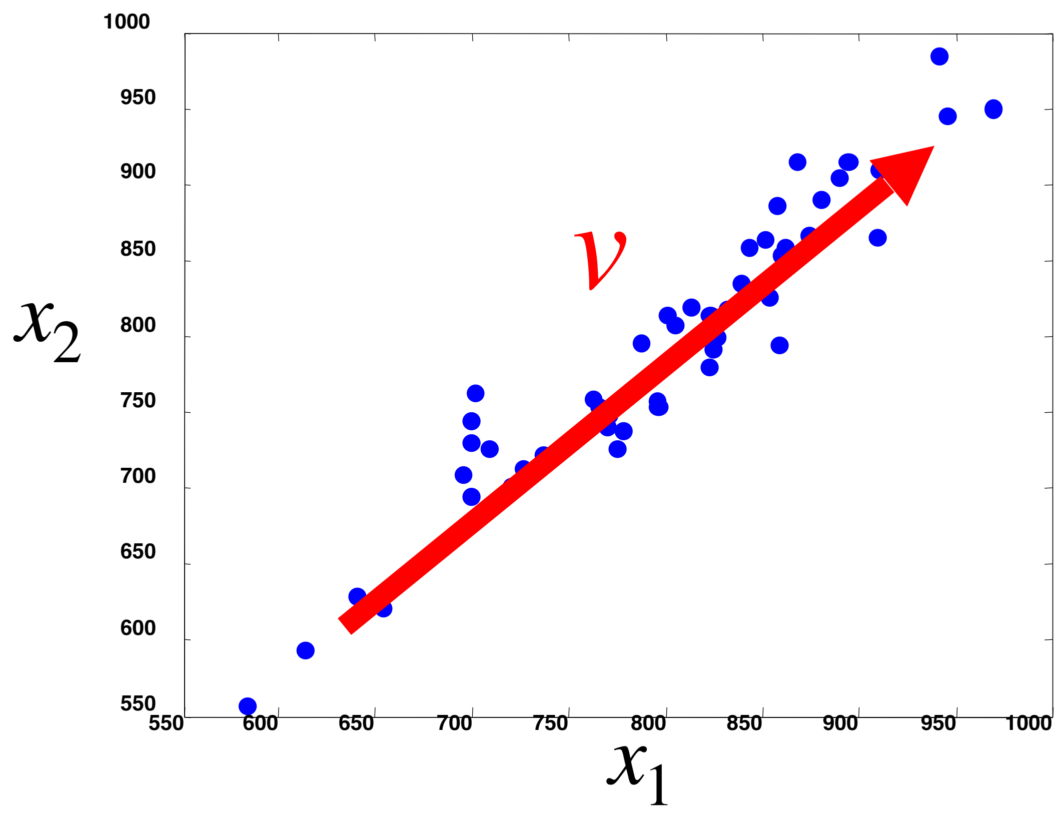
Dimensionality reduction

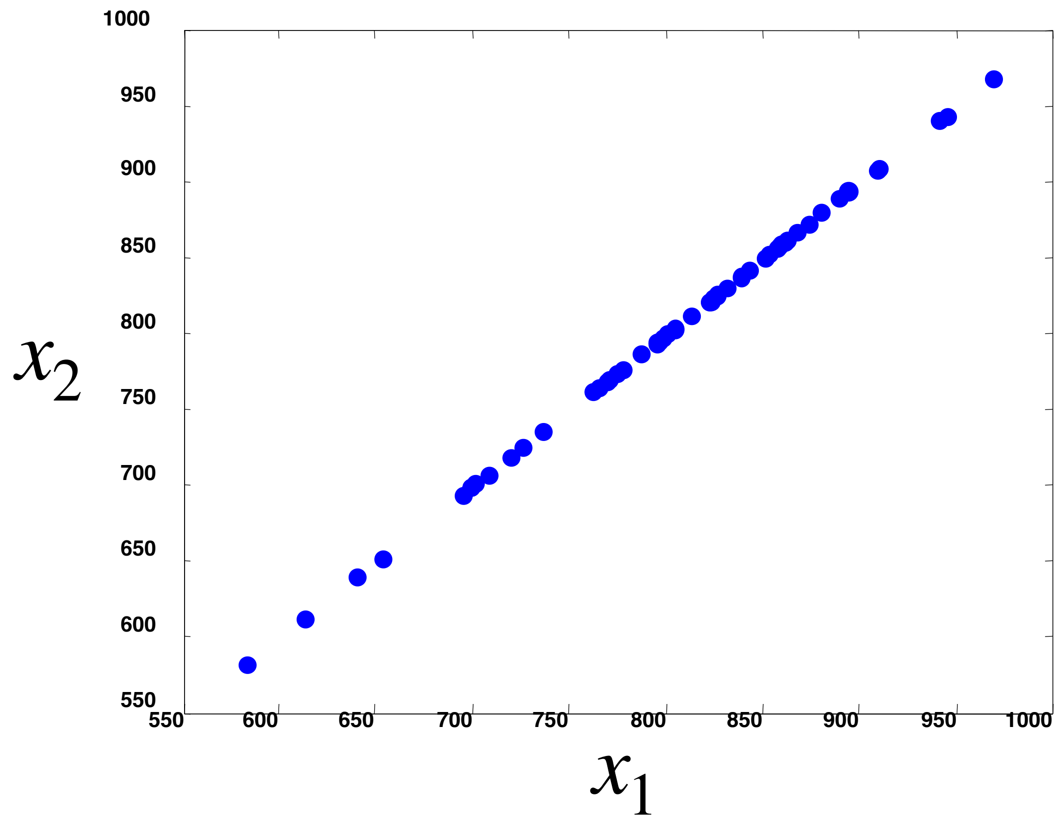
- With many features, some tend to change together
  - Can be summarized together
  - Others may have little or irrelevant change
- Example: S&P500 → "Tech stocks up 2x, manufacturing up 1.5x, …"
- Embed instances in lower-dimensional space $f : \mathbb{R}^n \mapsto \mathbb{R}^d$
  - Keep dimensions of "interesting" variability of data
  - Discard dimensions of noise or unimportant variability; or no variability at all
  - Keep "similar" data close $\implies$ may preserve cluster structure, other insights

Linear features

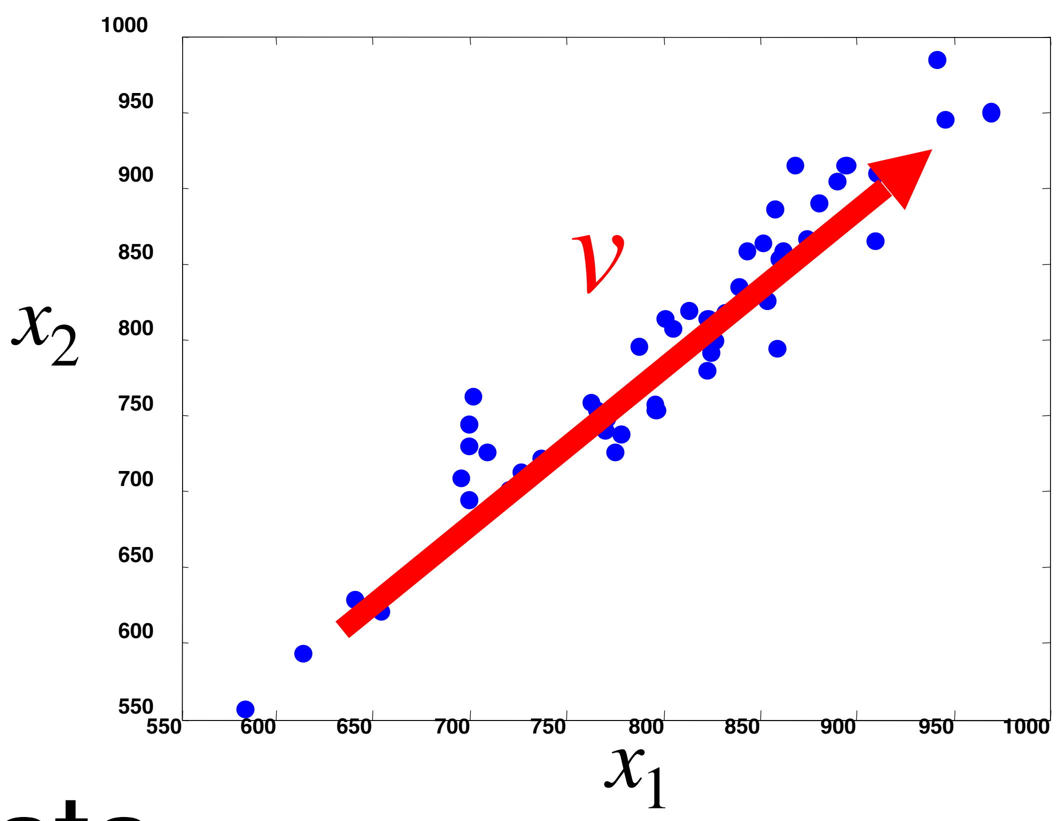
- Example: summarize two real features $x = [x_1, x_2] \to$ one real feature z
  - If z preserves much information about x, should be able to find $x \approx f(z)$
- Linear embedding:
  - $x \approx zv$
  - zv should be the closest point to x along v
    - $z = \arg\min \|x - zv\|^2 \implies z = \frac{x^\intercal v}{v^\intercal v}$

projection of x on v





Principal Component Analysis (PCA)

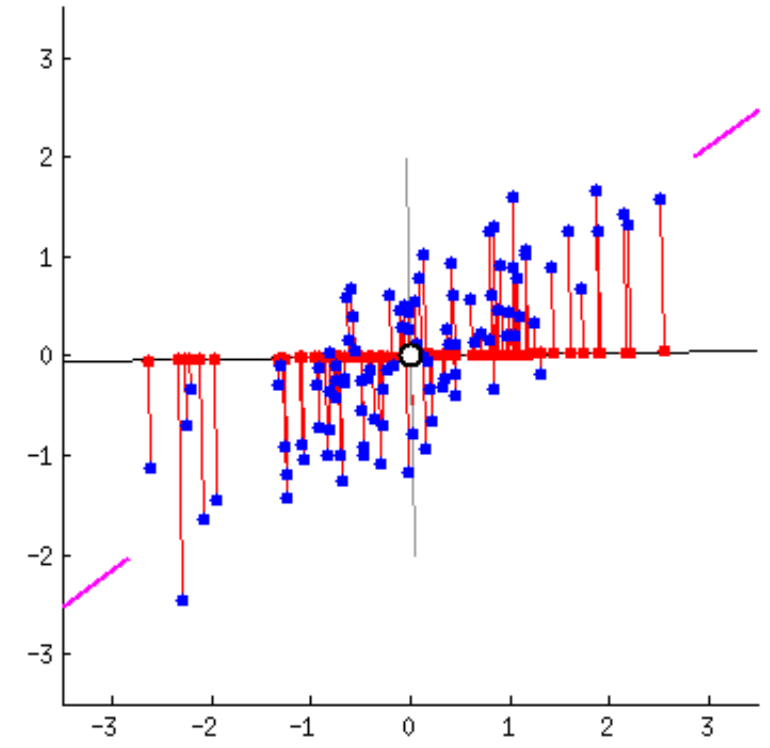


- How to find a good v?
  - Assume X has mean 0; otherwise, subtract the mean $\tilde{X} = X - \mu$
  - Idea: find the direction v of maximum "spread" (variance) of the data
  - Project $\tilde{X}$ on v: $z = \tilde{X}v$

empirical covariance

$$\max_{v:\|v\|=1} \sum_i (z_i)^2 = z^\intercal z = v^\intercal \tilde{X}^\intercal \tilde{X} v \implies v \text{ is eigenvector of } \tilde{X}^\intercal \tilde{X} \text{ of largest eigenvalue}$$

  - = minimum MSE of the residual $\tilde{X} - zv^\intercal = \tilde{X} - \tilde{X}vv^\intercal$



Source

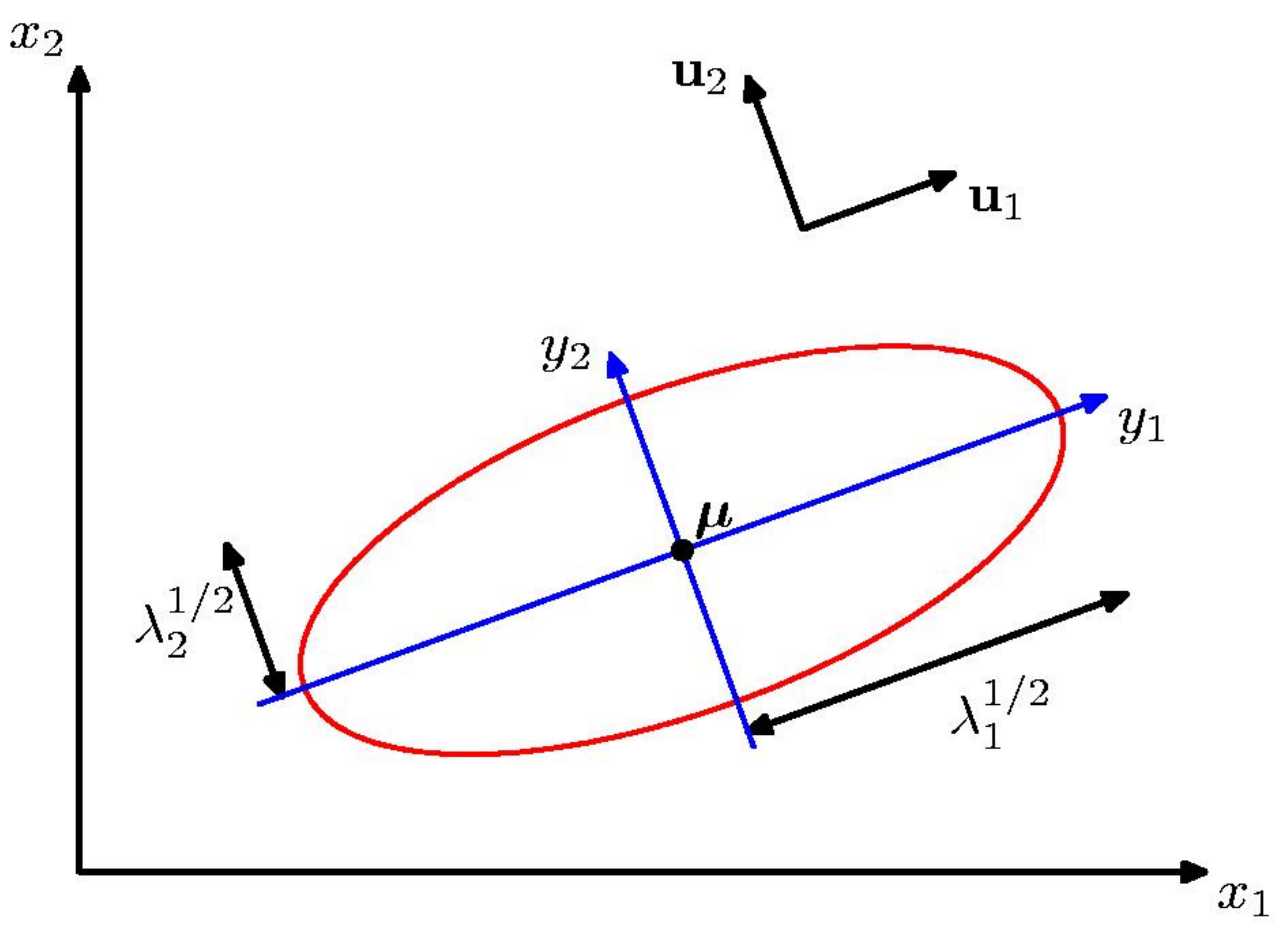
Geometry of a Gaussian

- Data covariance: $\Sigma = \frac{1}{m}\tilde{X}^\intercal \tilde{X}$ $\qquad \tilde{X} = X - \mu$

- Gaussian fit: $p(x) \sim \mathcal{N}(\mu, \Sigma)$

- Value contour for $p(x)$: $\Delta^2 = (x-\mu)^\intercal \Sigma^{-1} (x-\mu) = \text{const}$

- It's always possible to write Σ in terms of its eigenvectors U, eigenvalues λ:
  - $\Sigma = U \Lambda U^\intercal = \sum_{i=1}^{n} \lambda_i u_i u_i^\intercal \implies \Sigma^{-1} = \sum_{i=1}^{n} \frac{1}{\lambda_i} u_i u_i^\intercal$
  - In the eigenvector basis: $\Delta^2 = \sum_{i=1}^{n} \frac{y_i^2}{\lambda_i}$, with $y_i = u_i^\intercal (x - \mu)$



PCA representation

- Subtract data mean from data points
- (Optional) Scale each dimension by its variance
  - Don't just focus on large-scale features (e.g., +1 mileage $\ll$ +1yr ownership)
  - Focus on correlation between features
- Compute empirical covariance matrix $\Sigma = \frac{1}{m} \sum_i \tilde{x}_i \tilde{x}_i^\intercal$
- Take k largest eigenvectors of $\Sigma = U \Lambda U^\intercal$

Singular Value Decomposition (SVD)

- Alternative method for finding covariance eigenvectors
  - Has many other uses
- Singular Value Decomposition (SVD): $X = UDV^\intercal$

$$\underset{m \times n}{X} = \underset{m \times m}{U} \cdot \underset{m \times n}{D} \cdot \underset{n \times n}{V^\intercal}$$

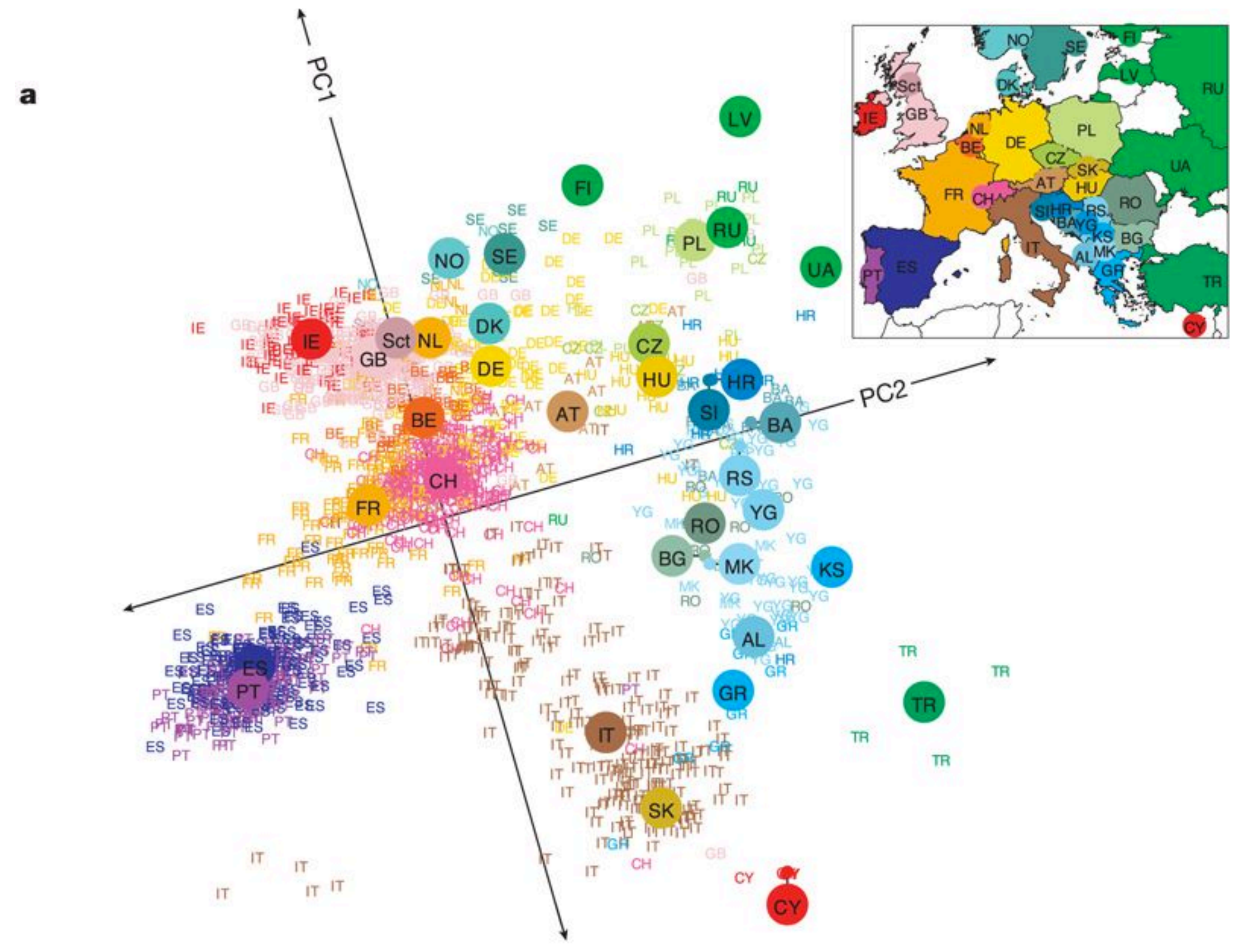
  - U and V (left- and right singular vectors) are orthogonal: $U^\intercal U = I$, $V^\intercal V = I$
  - D (singular values) is rectangular-diagonal
  - $\Sigma = X^\intercal X = VD^\intercal U^\intercal UDV^\intercal = V(D^\intercal D)V^\intercal$

$$\underset{m \times n}{X} \approx \underset{m \times k}{U_{1:k}} \cdot \underset{k \times k}{D_{1:k}} \cdot \underset{k \times n}{V^\intercal_{1:k}}$$

- UD matrix gives coefficients to reconstruct data: $x_i = U_{i,1}D_{1,1}v_1 + U_{i,2}D_{2,2}v_2 + \cdots$
  - We can truncate this after top k singular values (square root of eigenvalues)

Latent-space representations: uses

- Remove unneeded features
  - Features that add very little information (e.g. low variability, high noise)
  - Features that are similar to others (e.g. almost linearly dependent)
  - Reduce dimensionality for downstream application
    - Supervised learning: fewer parameters, need less data
    - Compression: less bandwidth



- Can also add features
  - Summarize multiple features into few cleaner / higher-level ones

PCA: applications

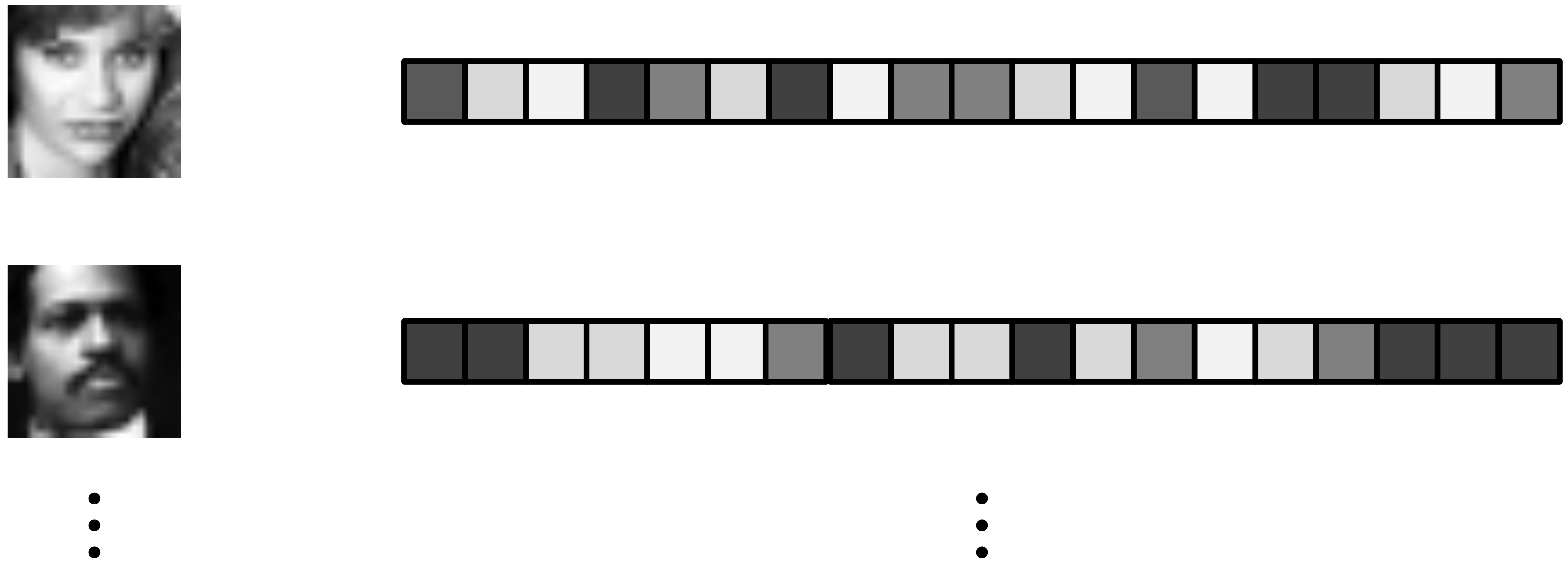
- Eigen-faces
  - Represent image data (e.g. faces) using PCA
- Latent-Semantic Analysis (“Topic Models”)
  - Represent text data (e.g. bag of words) using PCA
- Collaborative Filtering for Recommendation Systems
  - Represent sentiment data (e.g. ratings) using PCA

Eigen-faces

- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image

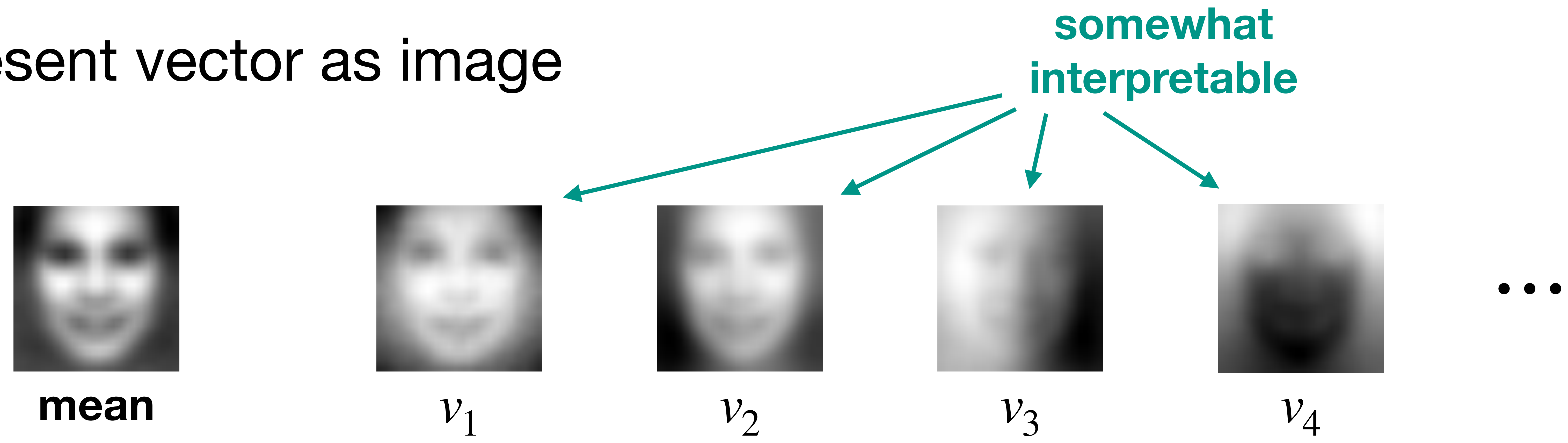


Eigen-faces

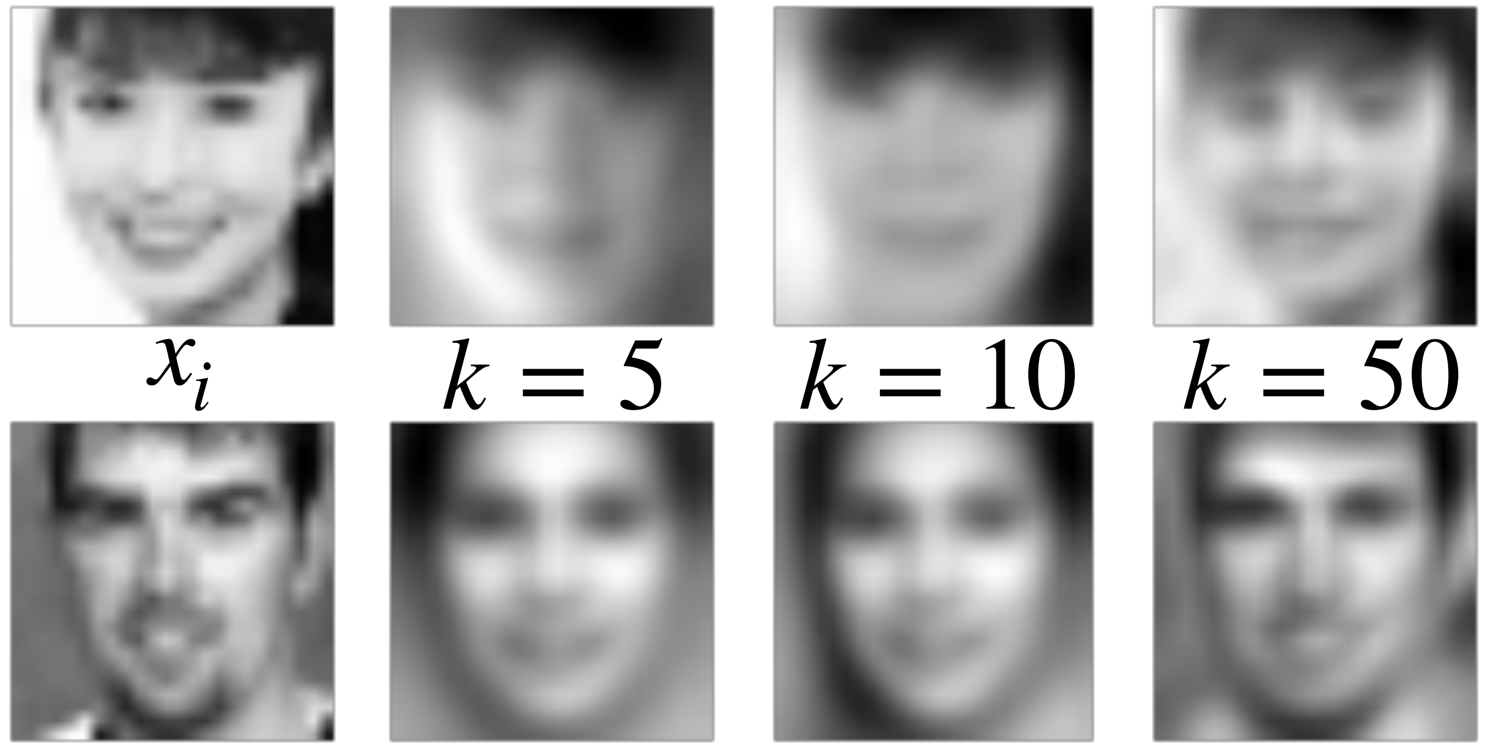
- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image



  - Project data on k principal components

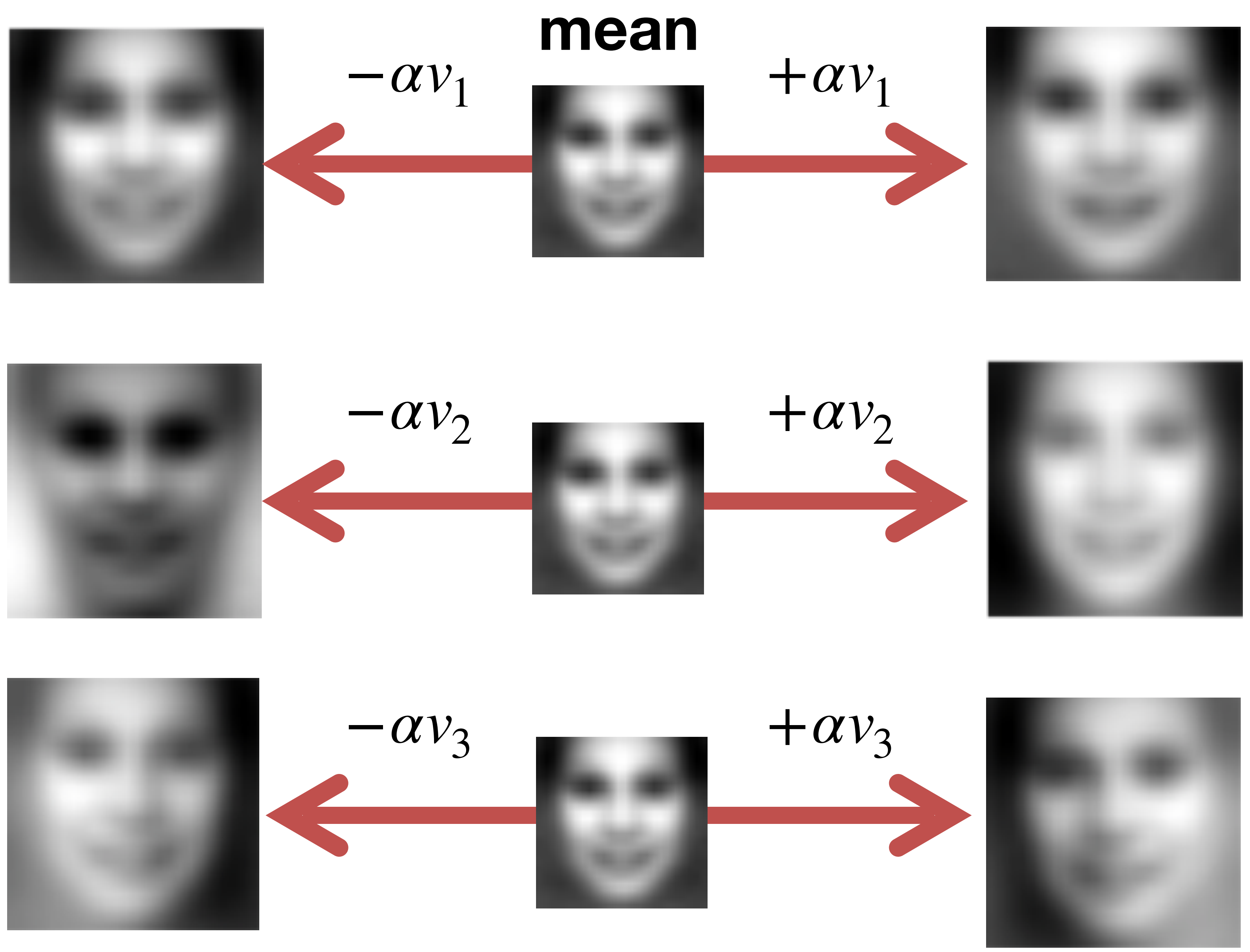


Eigen-faces

- "Eigen-X" = represent X using its principal components

$$\underset{m \times n}{X} \approx \underset{m \times k}{U} \cdot \underset{k \times k}{D} \cdot \underset{k \times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image
  - Visualize basis vectors v_i as $\mu \pm \alpha v_i$

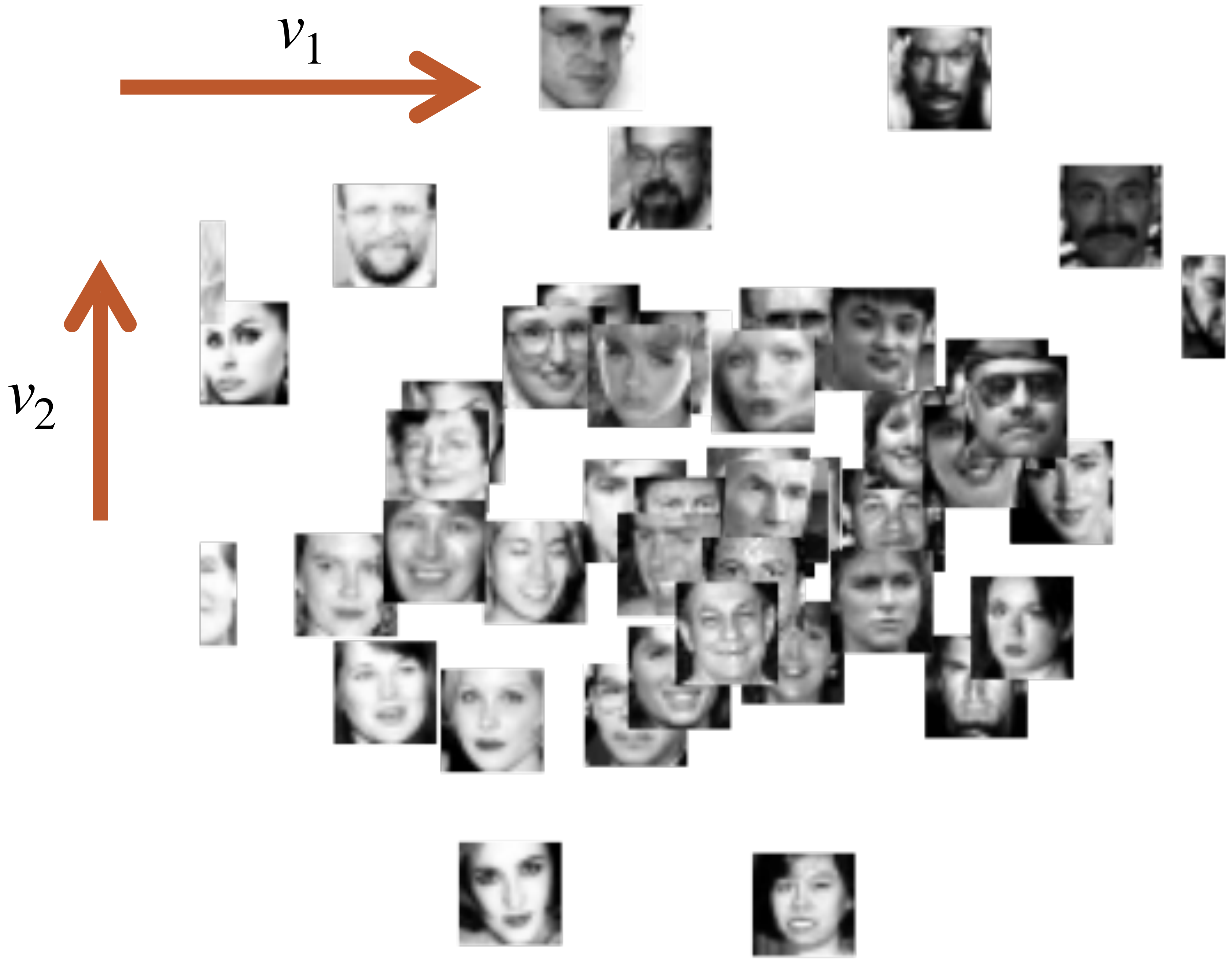


Eigen-faces

- "Eigen-X" = represent X using its principal components

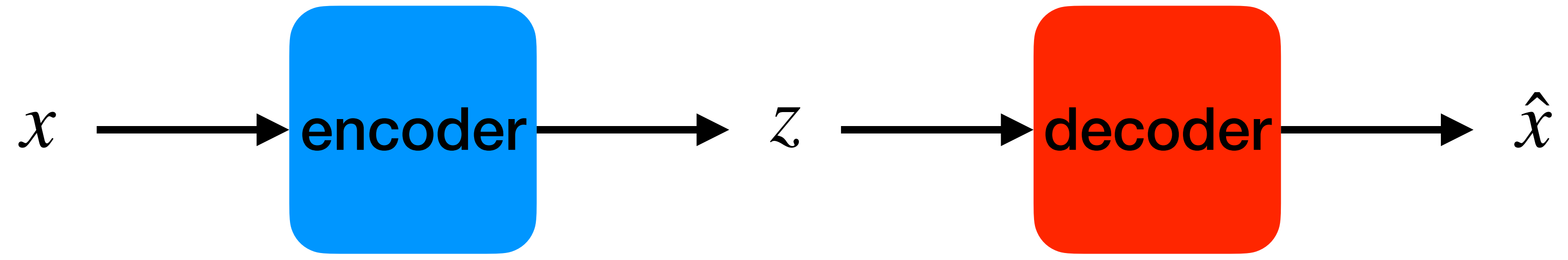
$$\underset{m\times n}{X} \approx \underset{m\times k}{U} \cdot \underset{k\times k}{D} \cdot \underset{k\times n}{V^\intercal}$$

- Viola Jones dataset: 24×24 images $\in \mathbb{R}^{576}$
  - Can represent vector as image
  - Visualize data by projecting onto 2 principal components

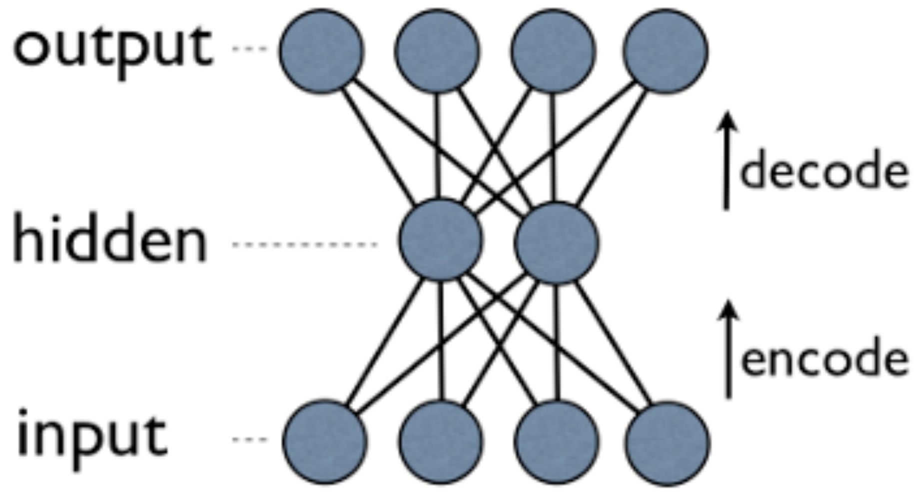


Nonlinear latent spaces

- Latent-space representation = represent x_i as z_i
  - Usually more succinct, less noisy
  - Preserves most (interesting) information on $x_i \implies$ can reconstruct $\hat{x}_i \approx x_i$
  - Auto-encoder = encode $x \to z$, decode $z \to \hat{x}$

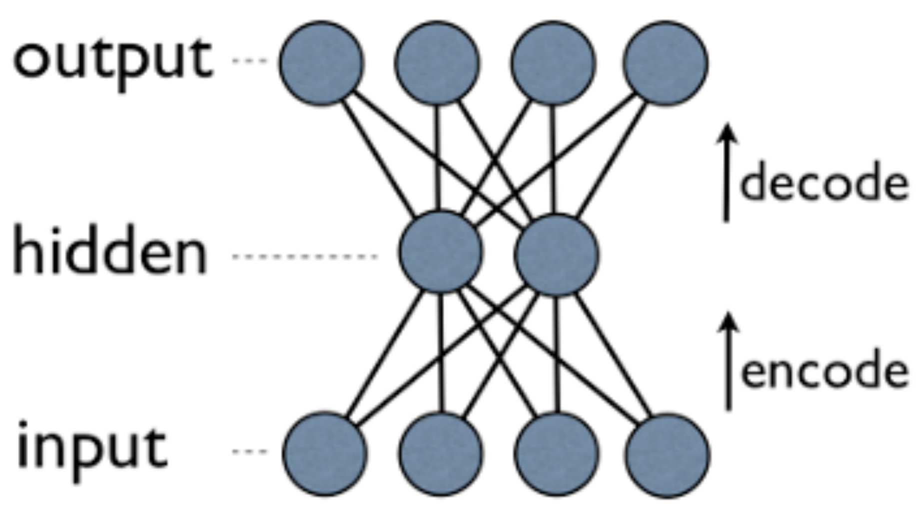


- Linear latent-space representation:
  - Encode: $Z = XV_{\leq k} = (UDV^\intercal V)_{\leq k} = U_{\leq k}D_{\leq k}$; Decode: $X \approx ZV^\intercal_{\leq k}$
- Nonlinear: e.g., encoder + decoder are neural networks
  - Restrict z to be shorter than $x \implies$ requires succinctness



Variational Auto-Encoders (VAE)

- Probabilistic model:



  - Simple prior over latent space $p(z)$ (e.g. Gaussian)
  - Decoder = generator $p_\theta(x|z)$, tries to match data distribution $p_\theta(x) \approx \mathcal{D}$
  - Encoder = inference $q_\phi(z|x)$, tries to match posterior $q_\phi(z|x) \approx \frac{p(z)p_\theta(x|z)}{p_\theta(x)}$
  - Can control generation of x through z in $p_\theta(x|z)$

Today's lecture

Latent-space models

Active learning

Online learning

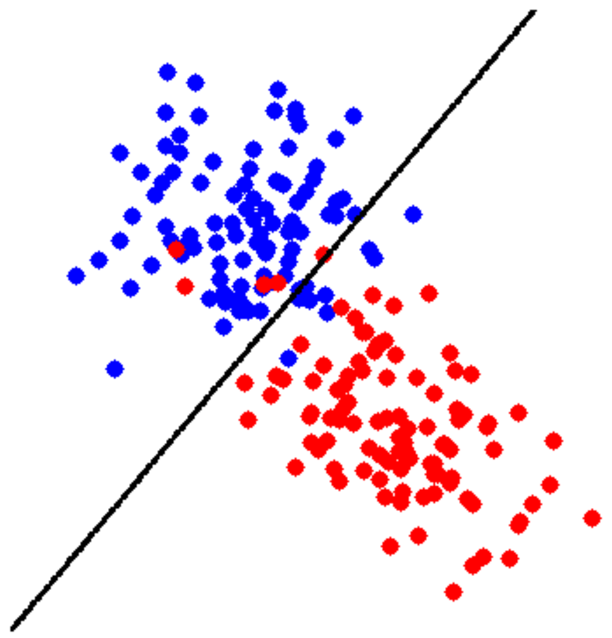
Sequential decision making

Motivation

- Supervised learning: classification
  - Pro: training data $\mathcal{D} = \{(x^{(j)}, y^{(j)})\}$ very informative
  - Con: expert labels $y^{(j)}$ may be expensive to get for big data
- Unsupervised learning: clustering
  - Pro: training data $\mathcal{D} = \{x^{(j)}\}$ may be easier to get
  - Con: discovered clusters may not match intended classes
- Semi-supervised learning: best of both worlds?
  - Few labels $\implies$ class identity; much unlabeled data $\implies$ class borders

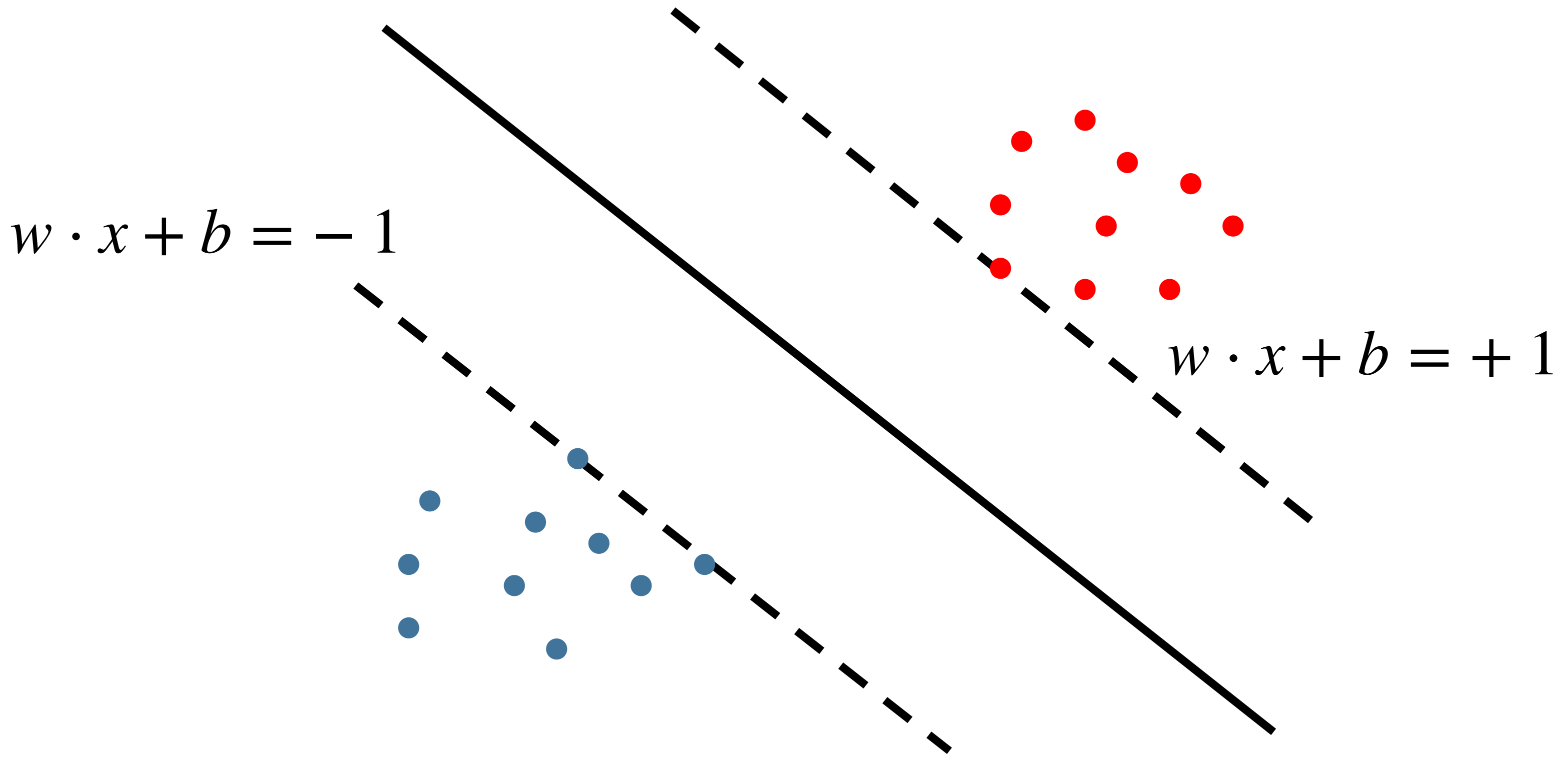
Example: semi-supervised SVM

- Problem: only few instances are labeled
  - Do unlabeled instances violate the margin constraints $y^{(j)}(w \cdot x^{(j)} + b) \geq 1$?
    - We don't know $y^{(j)}$...



- Let's assume labels are correct $\implies y^{(j)} = \text{sign}(w \cdot x^{(j)} + b)$
  - Constraint becomes $|w \cdot x^{(j)} + b| \geq 1 \iff x^{(j)}$ outside margin on either side
- Constraints no longer linear
  - Can solve with Integer Programming

    or other approximation methods

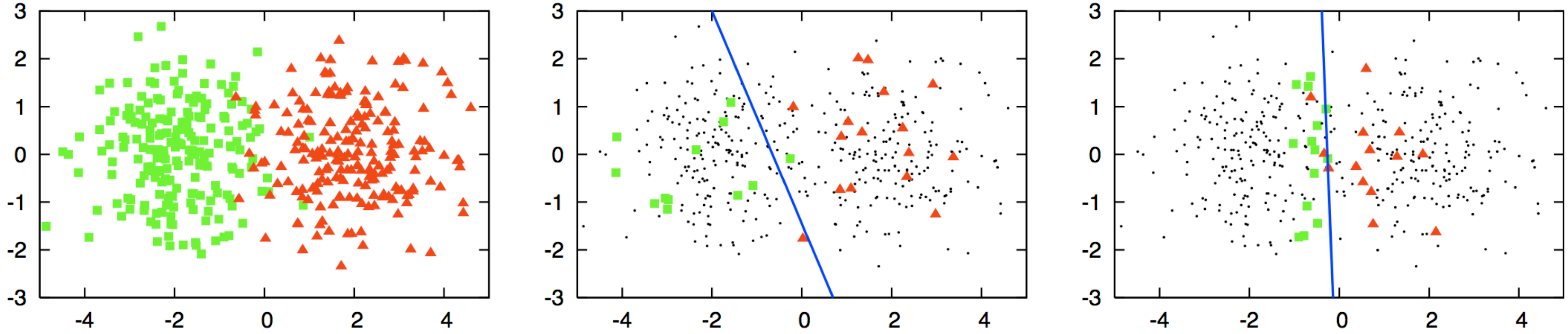


Who selects which instances to label?

- Random = semi-supervised learning
  - Labeled points $\sim p(x,y)$, unlabeled points from marginal distribution $\sim p(x)$
  - Equivalently: select instances $\sim p(x)$, select uniformly which to label $\sim p(y \mid x)$
- Teacher = exact learning, curriculum learning
  - Teacher identifies where learner is wrong, provides corrective labels
  - Some learners benefit from gradual increase in complexity (e.g. boosting)
- Learner = active learning
  - Automate the process of selecting good points to label

Why active learning?

full labeled data (unavailable) | **SVM on random sample of labeled data** | **SVM on selected sample of labeled data**



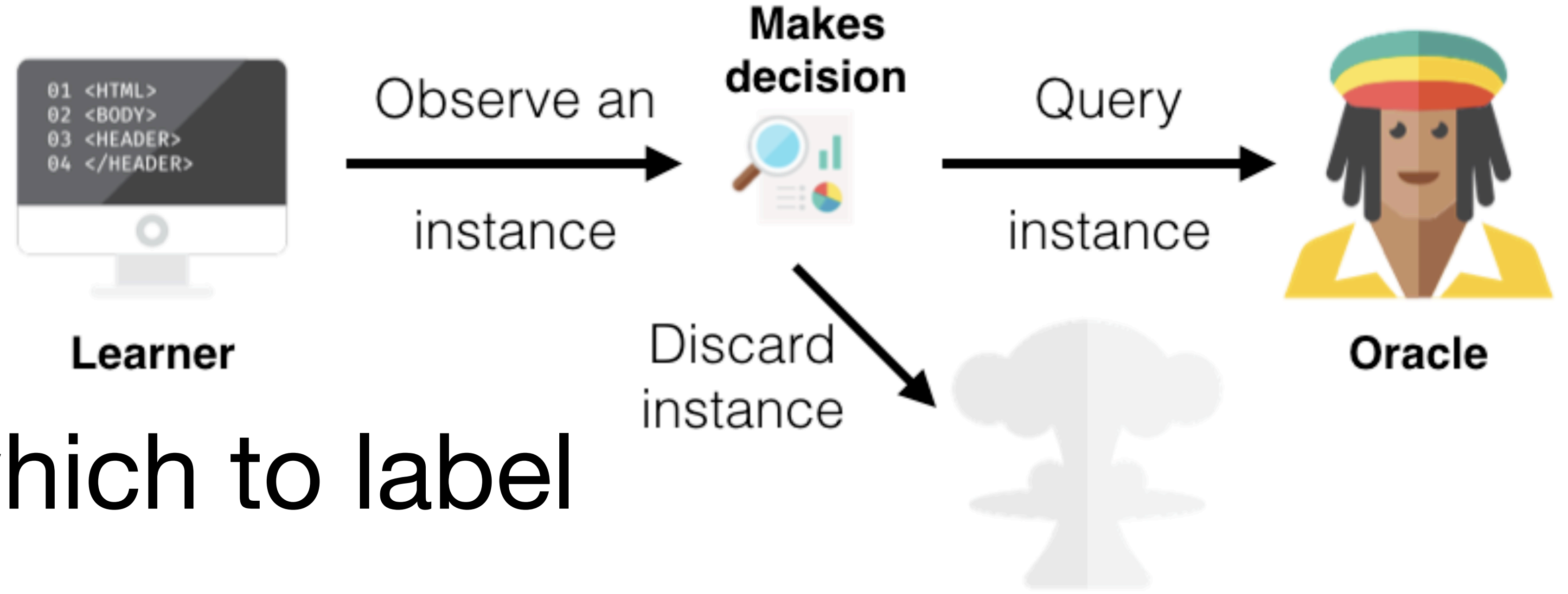
Source: https://www.datacamp.com/community/tutorials/active-learning

- Expensive labels $\Longrightarrow$ prefer to label instances relevant to the decision
- Selecting relevant points may be hard too $\Longrightarrow$ automate with active learning
- Objective: learn good model while minimizing #queries for labels

Active learning settings

- Pool-Based Sampling
  - Learner selects instances in dataset $x \in \mathscr{D}$ to label

- Stream-Based Selective Sampling
  - Learner gets stream of instances $x_1, x_2, \ldots,$ decides which to label

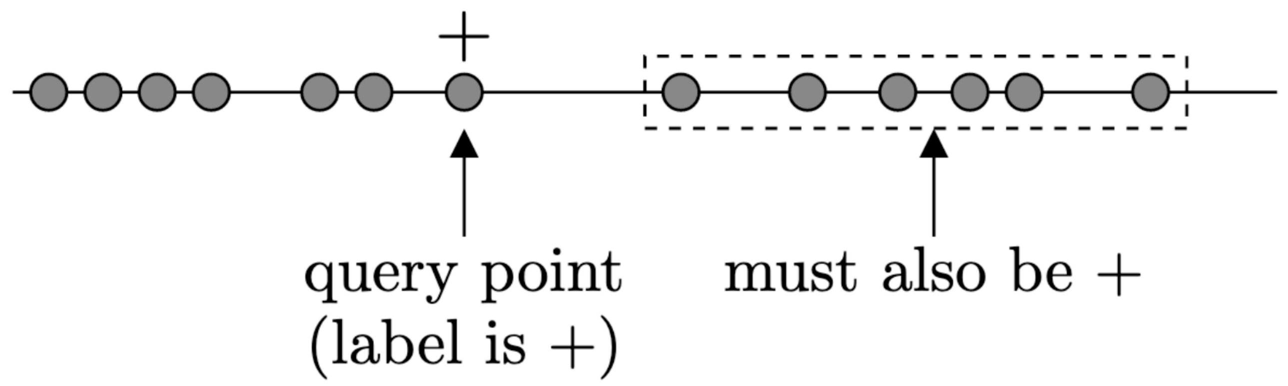


- Membership Query Synthesis
  - Learner generates instance x
  - Doesn't have to occur naturally = $p(x)$ may be low
    - $\implies$ May be harder for teacher to label ("is this synthesized image a dog or a cat?")

Source: https://www.datacamp.com/community/tutorials/active-learning

Simple example: find decision threshold

- When building decision tree on continuous features
  - Where to put the threshold on a given feature?
- If all data points are labeled and sorted $\Longrightarrow$ binary search
  - Split data in half until you find switch point of $-1 \to +1$
- Active learning = ask for labels



  - Same strategy: query mid point, if -1 / $+1$ $\Longrightarrow$ determines left / right half
  - #queries = $\log m$

How to select relevant data points?

- Least Confidence
  - Query point about which learner is most uncertain of the label
  - Requires learner to know its uncertainty, e.g. a probabilistic model $p_\theta(y|x)$
- Margin Sampling
  - Multi-class $\implies$ least confident doesn't mean least likely to get confused
    - Example: $p_\theta(y|x)$ = [0.3, 0.4, 0.3] vs. [0.45, 0.5, 0.05]
  - Query point about which two classes are most similar (near margin between them)
- Entropy Sampling
  - Query point that has most entropy = maximum information gain by revealing true label

Today's lecture

Latent-space models

Active learning

Online learning

Sequential decision making

Online learning

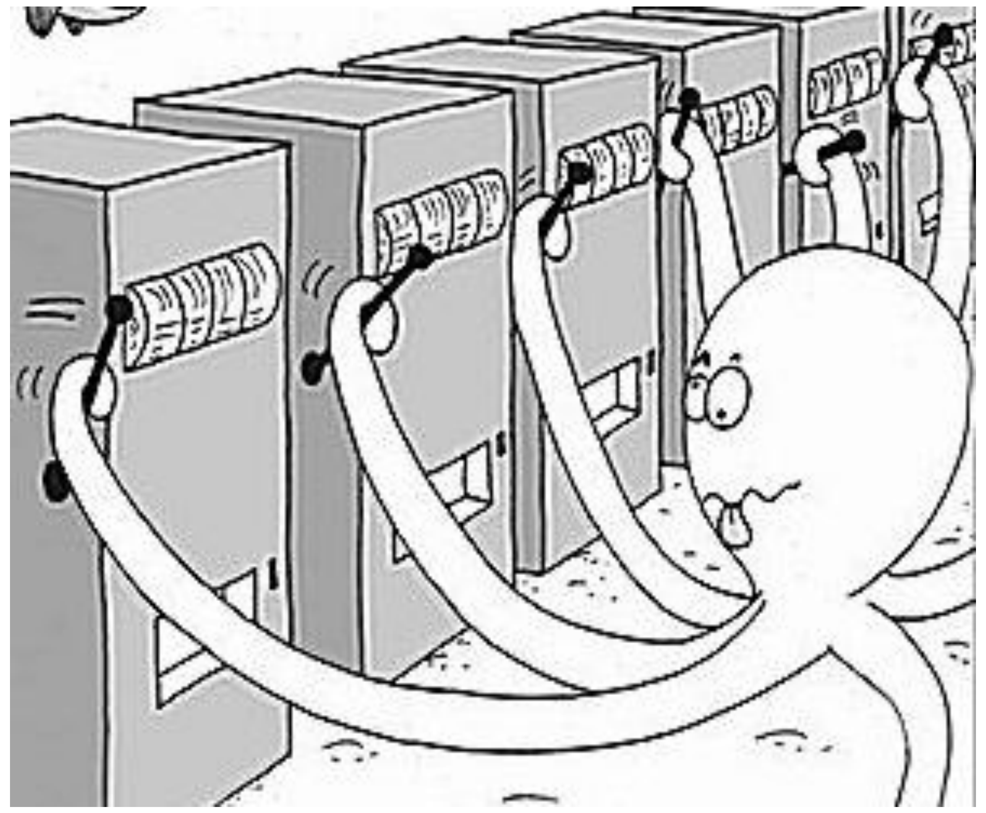
- In multi-class classification, we often assume 0–1 loss $\mathscr{L}(y, \hat{y}) = \delta[y \neq \hat{y}]$
- More generally, we can have different costs $\mathscr{L}(y, \hat{y}) = d(y, \hat{y})$
- Online learning:
  - Stream of instances, need to make predictions / decisions / actions online
  - We don't know the reward = -cost until we actually select $\hat{y}$
  - We'll never know the reward of other actions
- Objective:
  - Make better and better decisions (compared to what? later...)

Multi-Armed Bandits (MABs)

- Basic setting: single instance x, multiple actions $a_1, \ldots, a_k$
  - Each time we take action a_i we see a noisy reward $r_t \sim p_i$
- Can we maximize the expected reward $\max_i \mathbb{E}_{r \sim p_i}[r]$?
  - We can use the mean as an estimate $\mu_i = \mathbb{E}_{r \sim p_i}[r] \approx \frac{1}{m_i} \sum_{t \in T_i} r_t$
- Challenge: is the best mean so far the best action?
  - Or is there another that's better than it appeared so far?

One-armed bandit

Multi-armed bandit



Exploration vs. exploitation

- Exploitation = choose actions that seems good (so far)
- Exploration = see if we're missing out on even better ones
- Naïve solution: learn r by trying every action enough times
  - Suppose we can't wait that long: we care about rewards while we learn
- Regret = how much worse our return is than an optimal action

$$\rho(T) = T\mu_{a*} - \sum_{t=0}^{T-1} r_t$$

  - Can we get the regret to grow sub-linearly with T? $\implies$ average goes to 0: $\frac{\rho(T)}{T} \to 0$

Let's play!

- http://iosband.github.io/2015/07/28/Beat-the-bandit.html

Simple exploration: ϵ-greedy

- With probability ϵ:
  - Select action uniformly at random
- Otherwise (w.p. $1 - \epsilon$):
  - Select best (on average) action so far
- Problem 1: all non-greedy actions selected with same probability
- Problem 2: must have $\epsilon \to 0$, or we keep accumulating regret
  - But at what rate should ϵ vanish?

Optimism under uncertainty

- Tradeoff: explore less used actions, but don't be late to start exploiting what's known
  - Principle: optimism under uncertainty = explore to the extent you're uncertain, otherwise exploit
- By the central limit theorem, the mean reward of each arm $\hat{\mu}_i$ quickly $\to \mathcal{N}\left(\mu_i, O\left(\frac{1}{m_i}\right)\right)$
- Be optimistic by slowly-growing number of standard deviations: $a = \arg\max_i \hat{\mu}_i + \sqrt{\frac{2\ln T}{m_i}}$
  - Confidence bound: likely $\mu_i \leq \hat{\mu}_i + c\sigma_i$; unknown constant in the variance $\implies$ let c grow
  - But not too fast, or we fail to exploit what we do know
- Regret: $\rho(T) = O(\log T)$, provably optimal

Thompson sampling

- Consider a model of the reward distribution $p_{\theta_i}(r \mid a_i)$
- Suppose we start with some prior $q(\theta)$
  - Taking action a_t, see reward $r_t \implies$ update posterior $q(\theta \mid \{(a_{\leq t}, r_{\leq t})\})$
- Thompson sampling:
  - Sample $\theta \sim q$ from the posterior
  - Take the optimal action $a^* = \max_i \mathbb{E}_{r \sim p_{\theta i}}[r]$
  - Update the belief (different methods for doing this)
  - Repeat

Other online learning settings

- What is the reward for action a_i?
  - MAB: random variable with distribution $p_i(r)$
  - Adversarial bandits: adversary selects r_i for every action
    - The adversary knows our algorithm! And past action selection! But not future actions
      - Learner must be stochastic (= unpredictable) in choosing actions
    - Amazingly, there are learners with regret guarantees
- Contextual bandits: we also get instance x, make decision $\pi(a \mid x)$
  - Can we generalize to unseen instances?

Today's lecture

Latent-space models

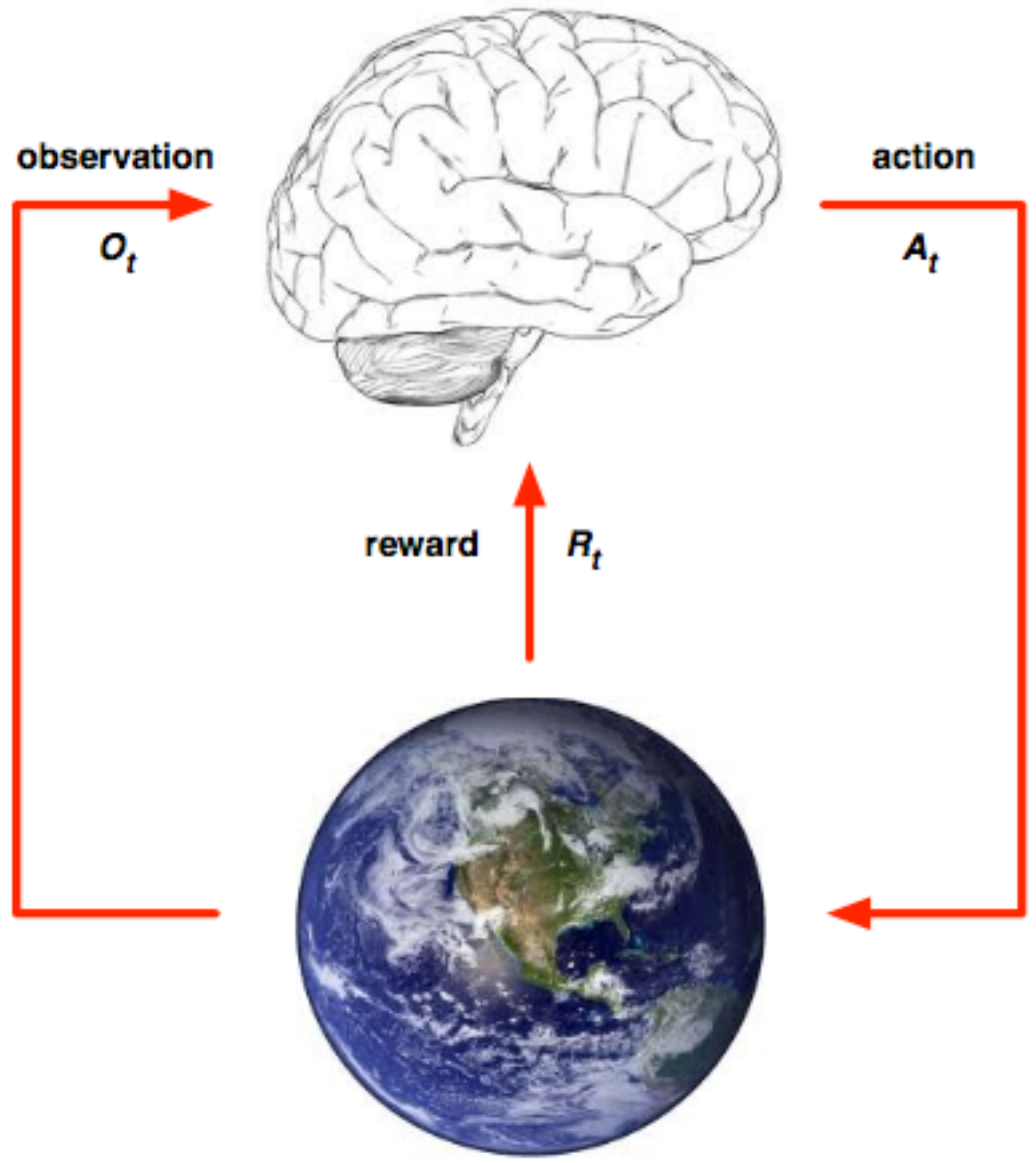
Active learning

Online learning

Sequential decision making

Agent–environment interface

- Agent
  - Decides on next action
  - Receives next reward
  - Receives next observation
- Environment
  - Executes the action → changes its state
  - Generates next observation
  - Supervisor: reveals the reward



Sequential decision making

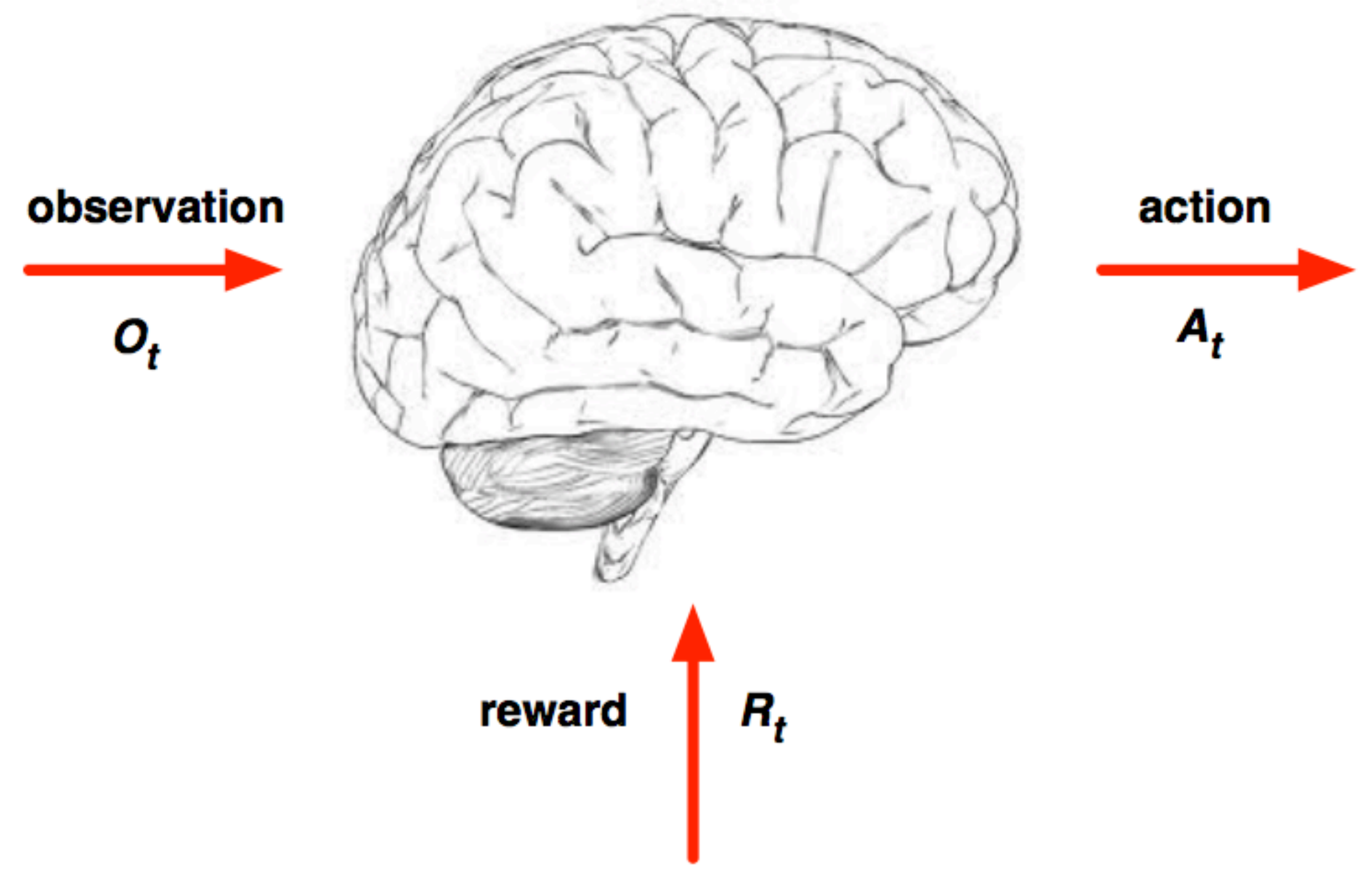
- Reinforcement learning = learning to make sequential decisions
- Challenges:
  - Online learning: reward is only given for actions taken (not for other actions)
  - Active learning: future “instances” determined by what the learner does
  - Sequential decisions: which of the decisions gets credit for a good reward?
- Examples:
  - Fly drone • play Go • trade stocks • control power station • control walking robot
- Rewards: track trajectory • win game • make $ • produce power (safely!)

Long-term planning

- Tradeoff: short-term rewards vs. long-term returns (accumulated rewards)
  - Fly drone: slow down to avoid crash?
  - Games: slowly build strength? block opponent? all out attack?
  - Stock trading: sell now or wait for growth?
  - Infrastructure control: reduce output to prevent blackout?
  - Life: invest in college, obey laws, get started early on course project
- Forward thinking and planning are hallmarks of intelligence

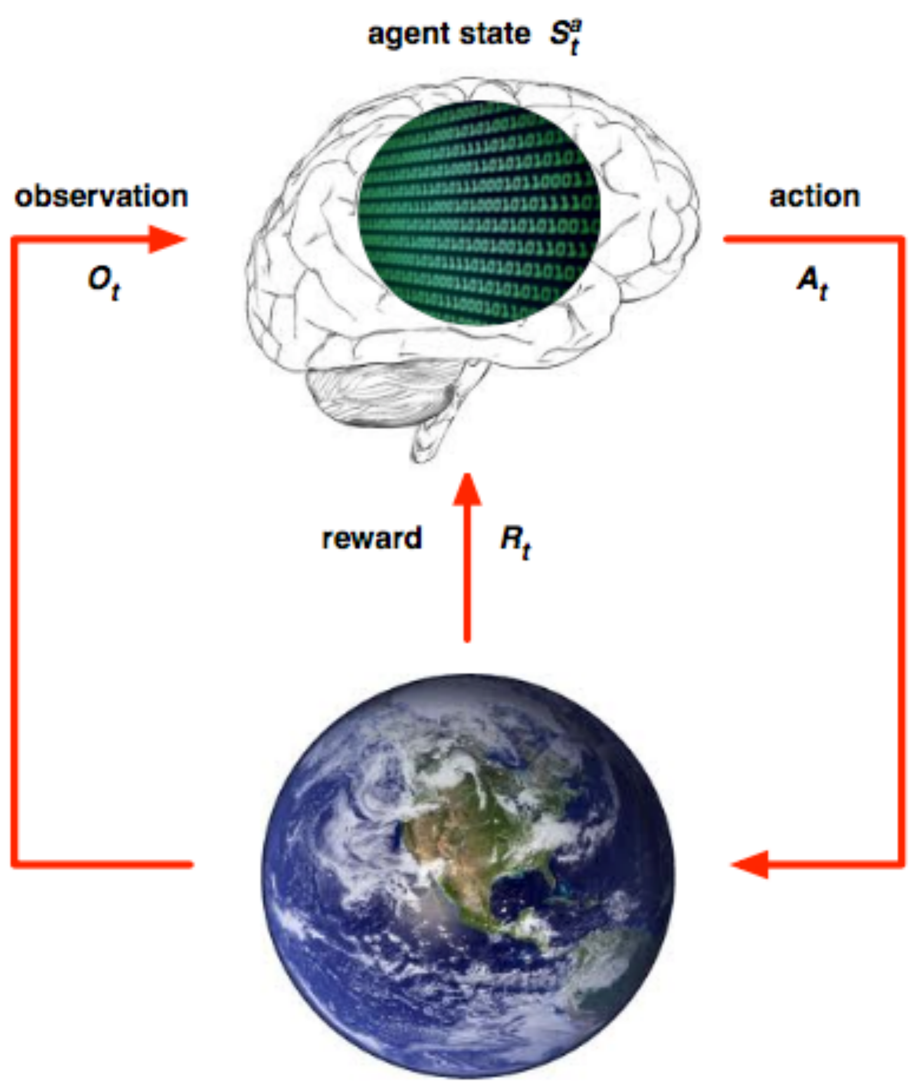
Intelligent agents

- Agent outputs action a_t
  - Function of the context: $a_t = f(x_t)$
    - Perhaps stochastic: $\pi(a_t \mid x_t)$
- What is the context needed for decisions?
  - Ignore all inputs? (open-loop control = sequence of actions)
  - Current observation o_t?
  - Previous action a_{t-1}? reward r_{t-1}?
  - All observations so far $o_{\le t}$?



Agent context x_t

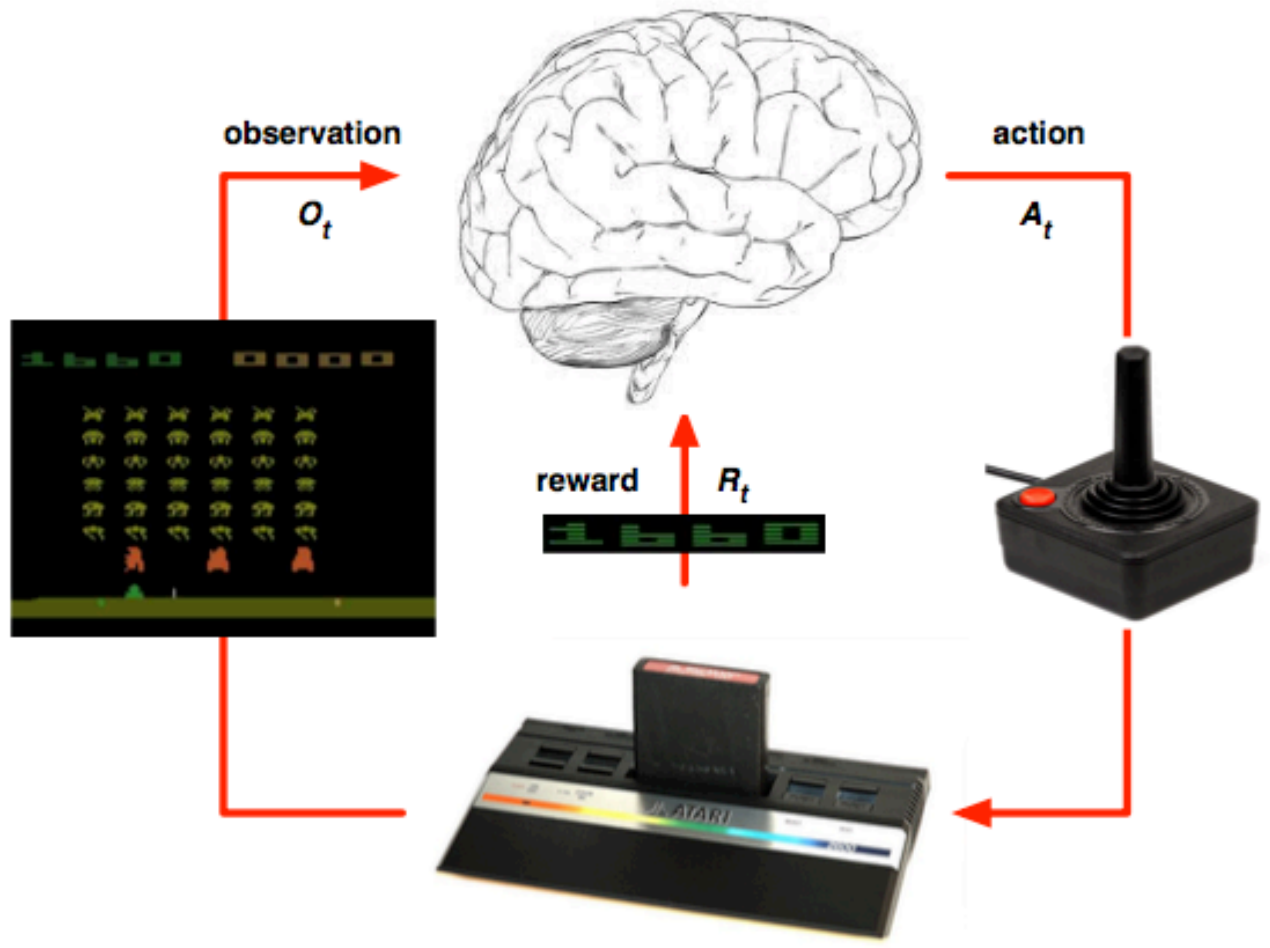
- Observable history: everything the agent saw so far
  - $h_t = (o_1, a_1, r_1, o_2, \ldots, a_{t-1}, r_{t-1}, o_t)$

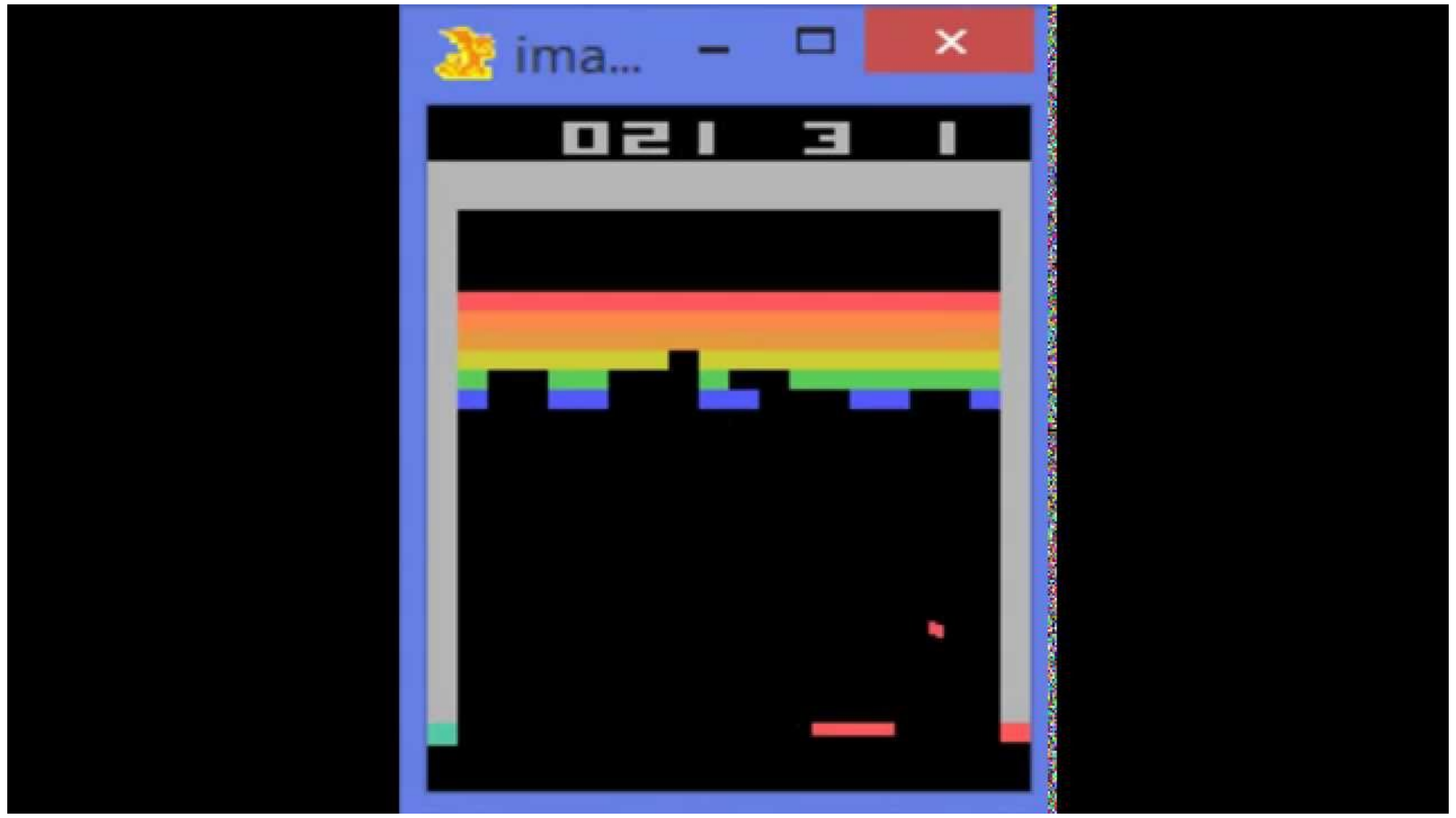


- The context x_t used for the agent's policy $\pi(a_t | x_t)$ can be:
  - Reactive policy: $x_t = o_t$ (optimal under full observability: $o_t = s_t$)
  - Using previous action: $x_t = (a_{t-1}, o_t) \implies$ can be useful if policy is stochastic
  - Using previous reward: $x_t = (r_{t-1}, o_t) \implies$ extra information about the environment
  - Window of past observations: $x_t = (o_{t-3}, o_{t-2}, o_{t-1}, o_t) \implies$ better see dynamics
  - Generally: any summary (= memory) of observable history $x_t = f(h_t)$

Example: Atari

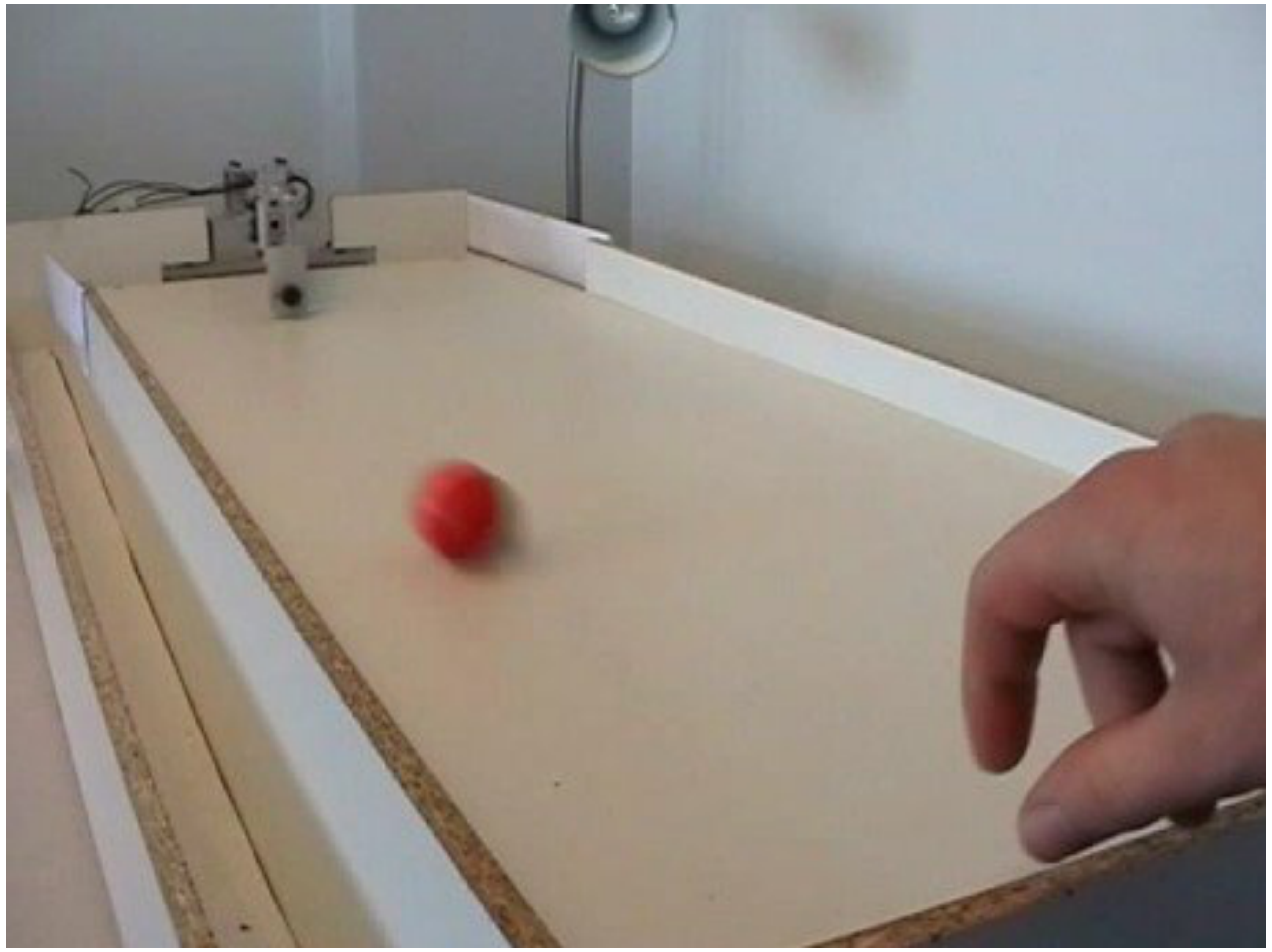
- Rules are unknown
  - What makes the score increase?
- Dynamics are unknown
  - How do actions change pixels?





https://www.youtube.com/watch?v=V1eYniJ0Rnk

Example: Table Soccer



https://www.youtube.com/watch?v=CIF2SBVY-J0

Logistics

assignments

- Assignment 5 due Tuesday, Nov 30

project

- Final report due next Thursday, Dec 2

final exam

- Review: next Thursday, Dec 2
- Final: Tuesday, Dec 7, 10:30am–12:30