UCI University of California, Irvine

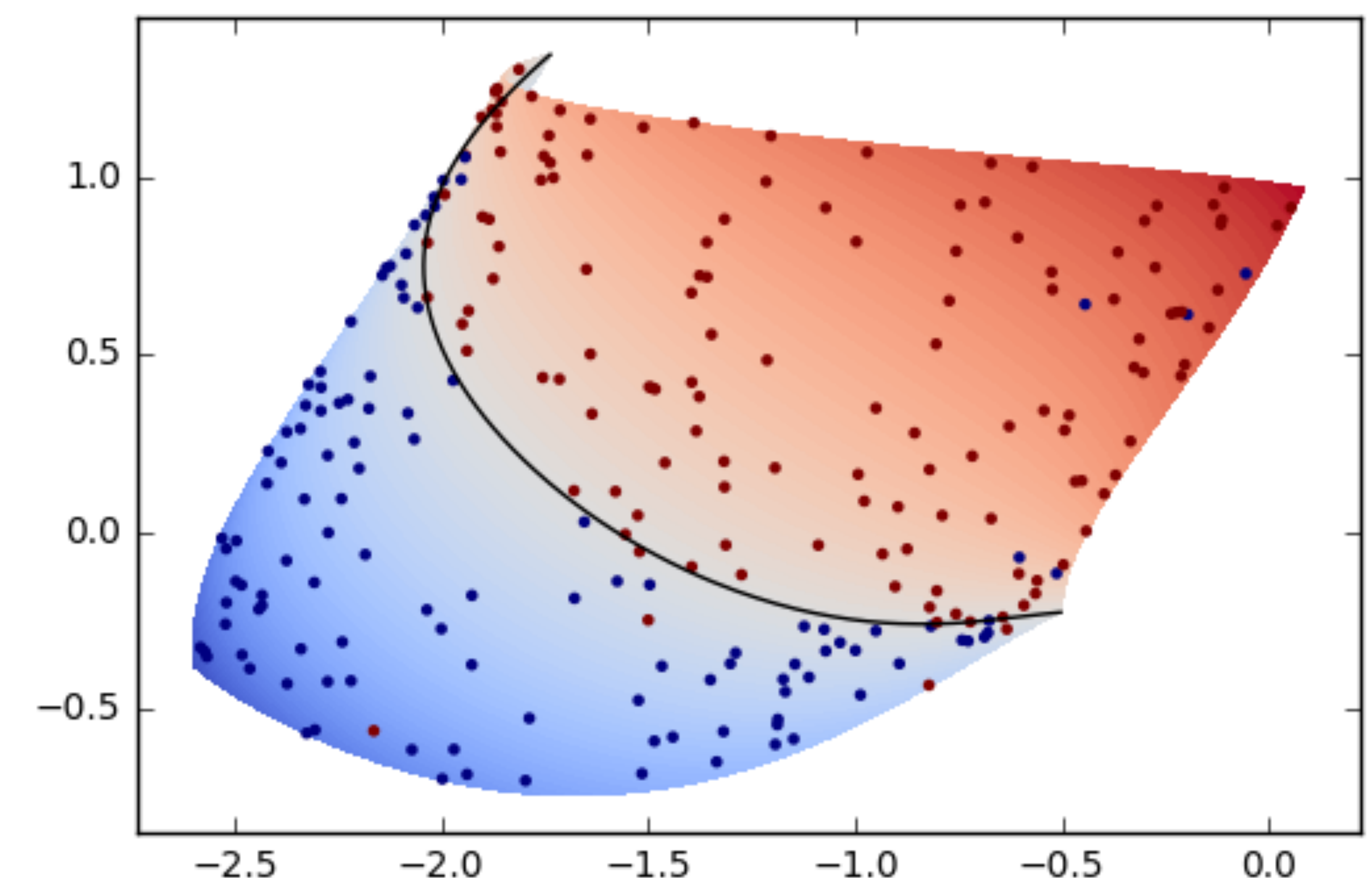
CS 273A: Machine Learning

Winter 2021

Lecture 19: Final Review

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



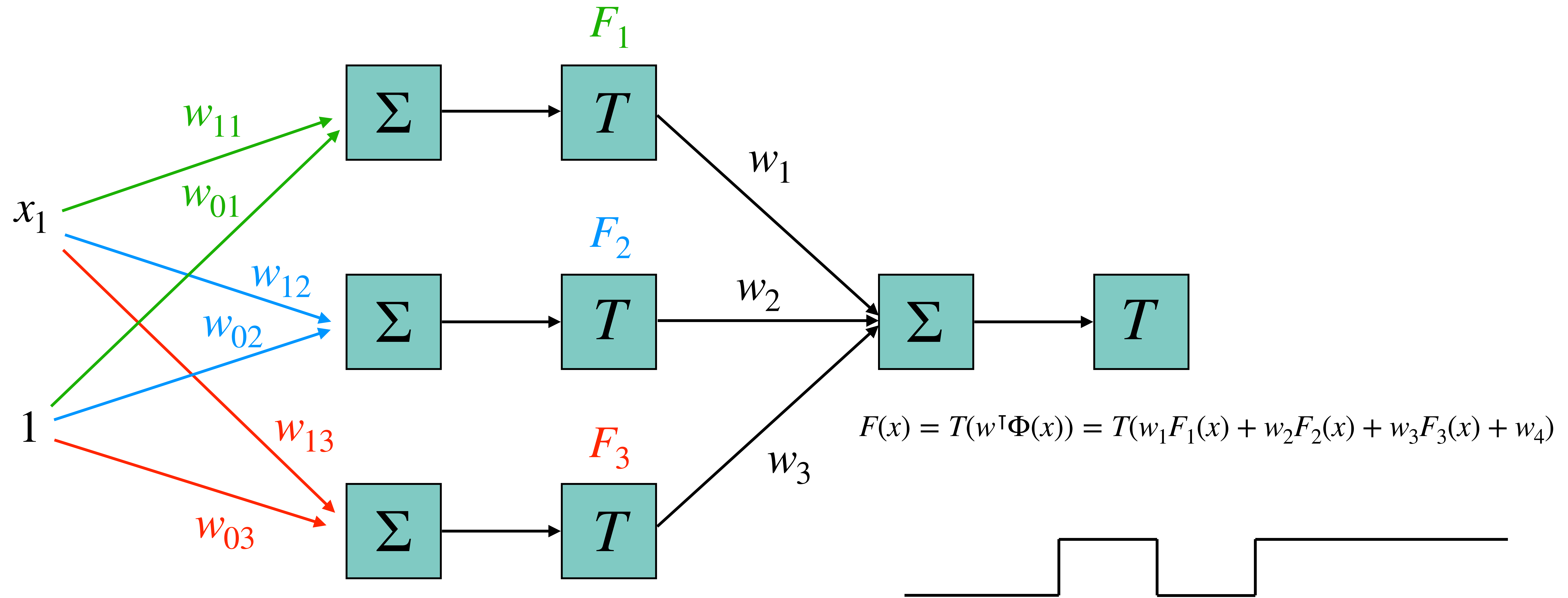
Final Logistics

- Format:
  - Time: Thursday, March 18, 1:30–4pm
  - Canvas "quiz": multiple choice, numerical, textual, drawing $\implies$ let us know about technical difficulties
  - Many questions, ~75% longer than midterm, but should be doable in < 2 hours
  - We'll be on zoom to address questions and issues: https://uci.zoom.us/j/94903054276
- You can use:
  - Self-prepared A4 / Letter-size two-sided single page with anything you'd like on it
  - A basic arithmetic calculator; no phones, no computers
  - Blank paper sheets for your calculations
  - Brainpower and good vibes
- No proctoring; the penalty for cheating is being the kind of person who cheats

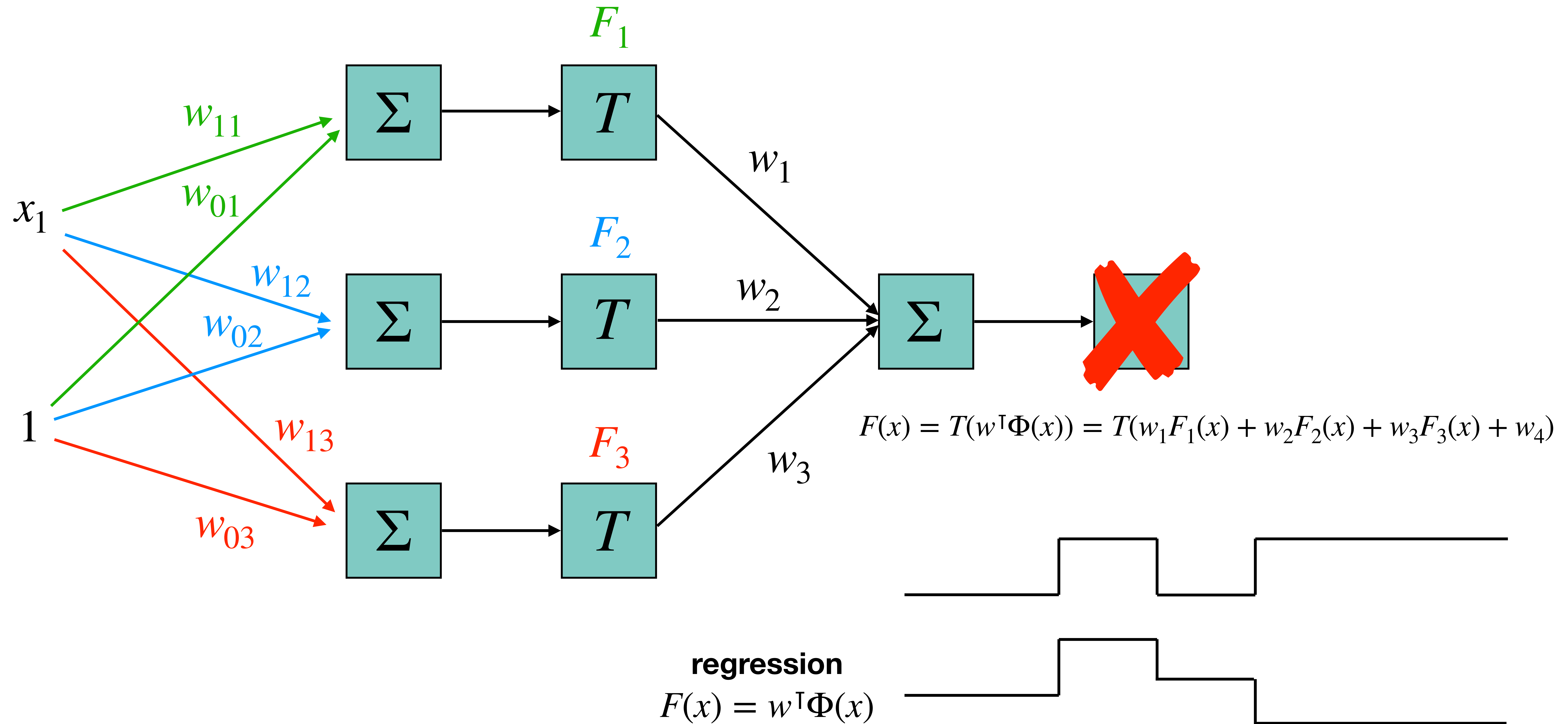
Exam suggestions

- Large majority of the questions are on topics taught after midterm
- Look at past exams
  - Train yourself by reading some solutions, evaluate yourself on held-out exams
- Organize / join study groups (e.g. on piazza)
- During the exam:
  - Start with questions you find easy
  - Don't get bogged down by exact calculations
  - Leave expressions unsolved and come back to them later
  - Optional: upload your calculation sheet(s)
    - They won't be graded, but can be used for regrading

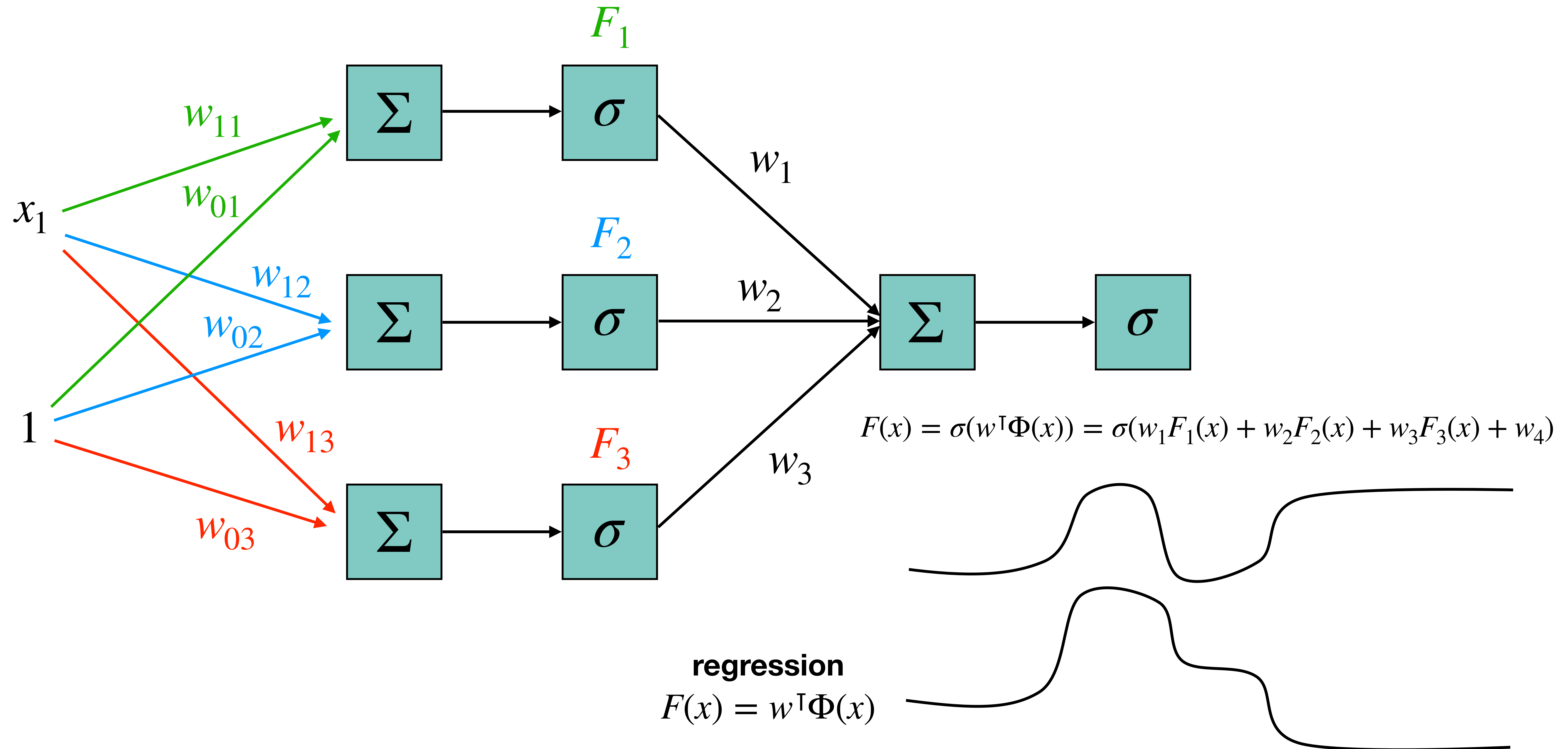
Multi-Layer Perceptron (MLP)



Multi-Layer Perceptron (MLP)

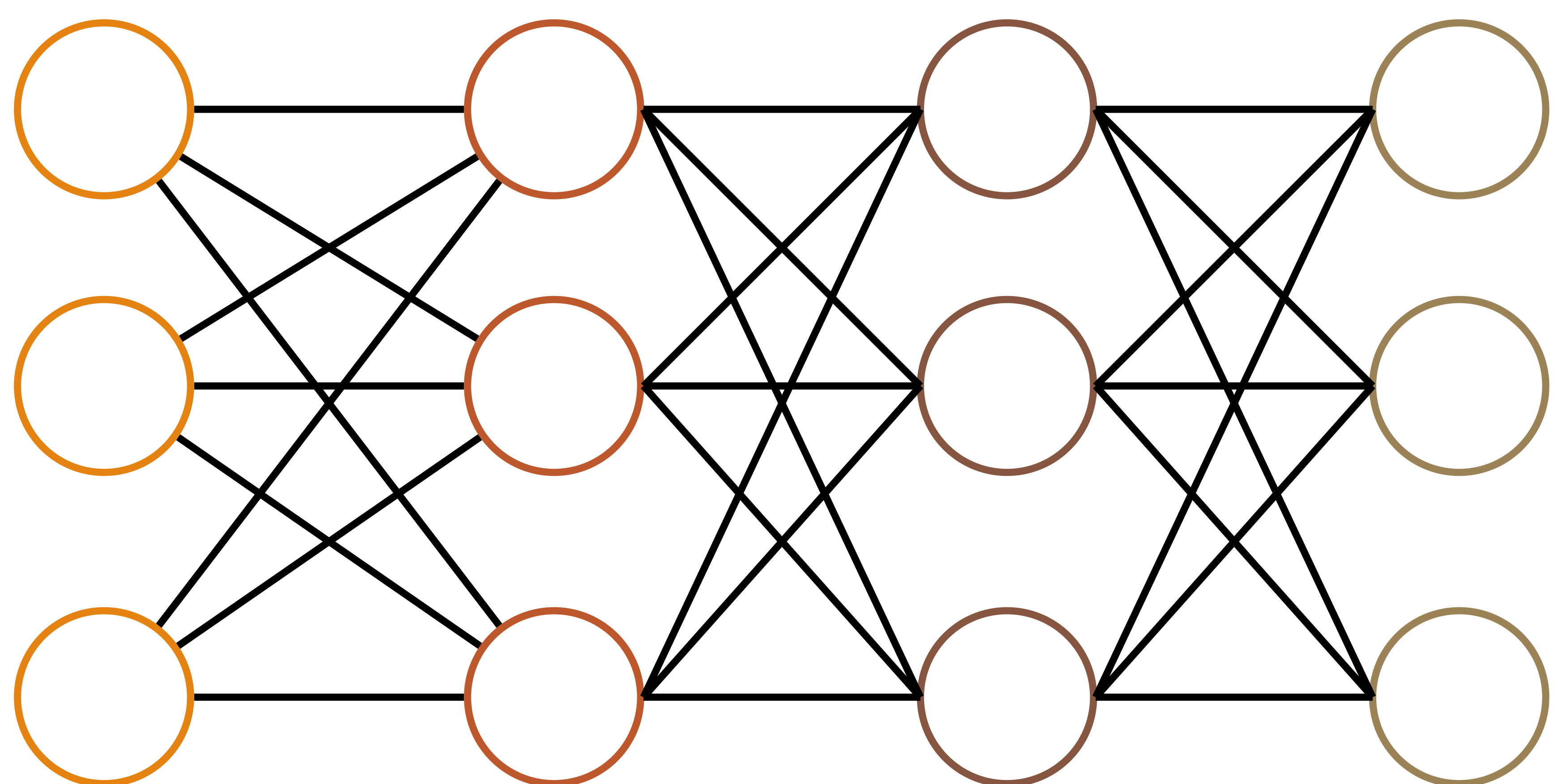


Multi-Layer Perceptron (MLP)



Deep Neural Networks (DNNs)

- Layers of perceptrons can be stacked deeply
  - Deep architectures are subject of much current research



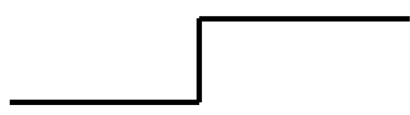
…

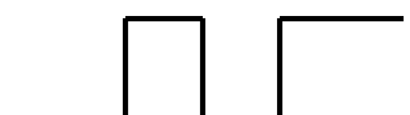
```
r1 = w[0].T @ x + b[0]    # linear response
h1 = sig(r1)              # activation function

r2 = w[1].T @ h1 + b[1]   # linear response
h2 = sig(r2)              # activation function

                          # ...
```

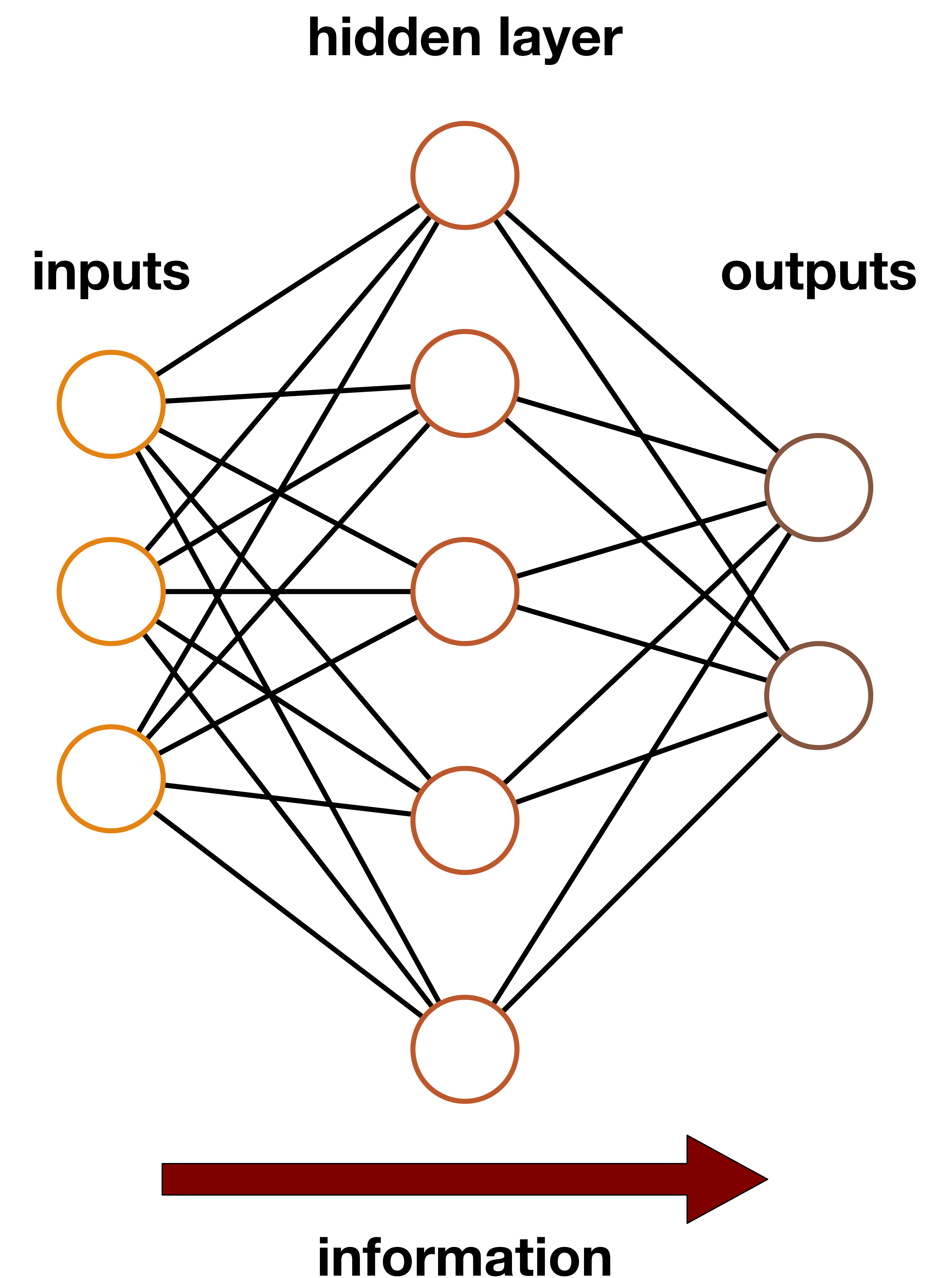
input features | **layer 1** | **layer 2** | **layer 3** | …





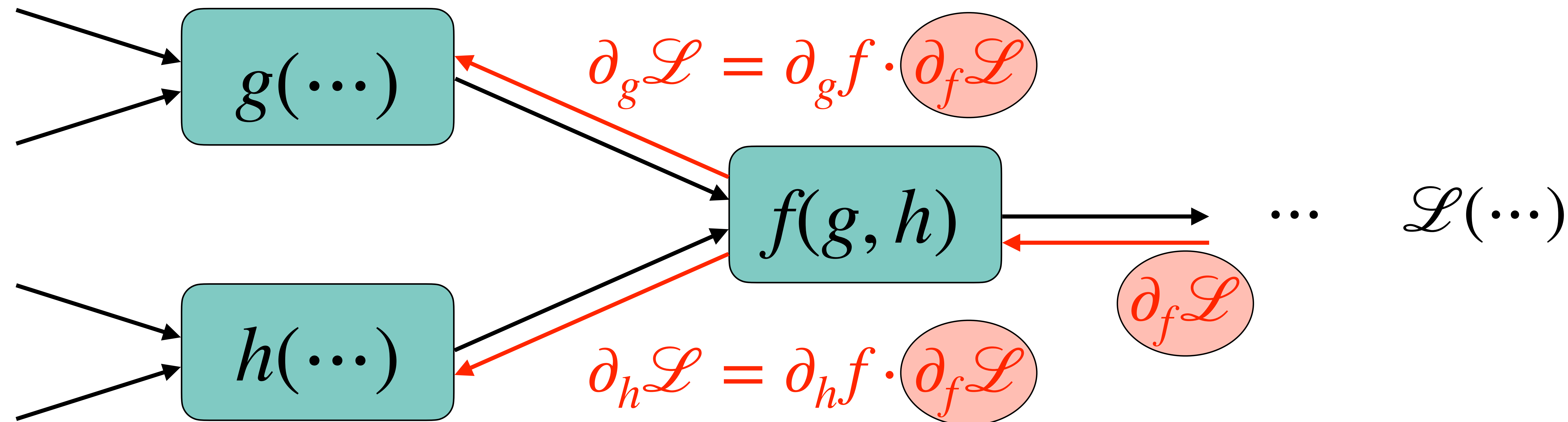
Feed-forward (FF) networks

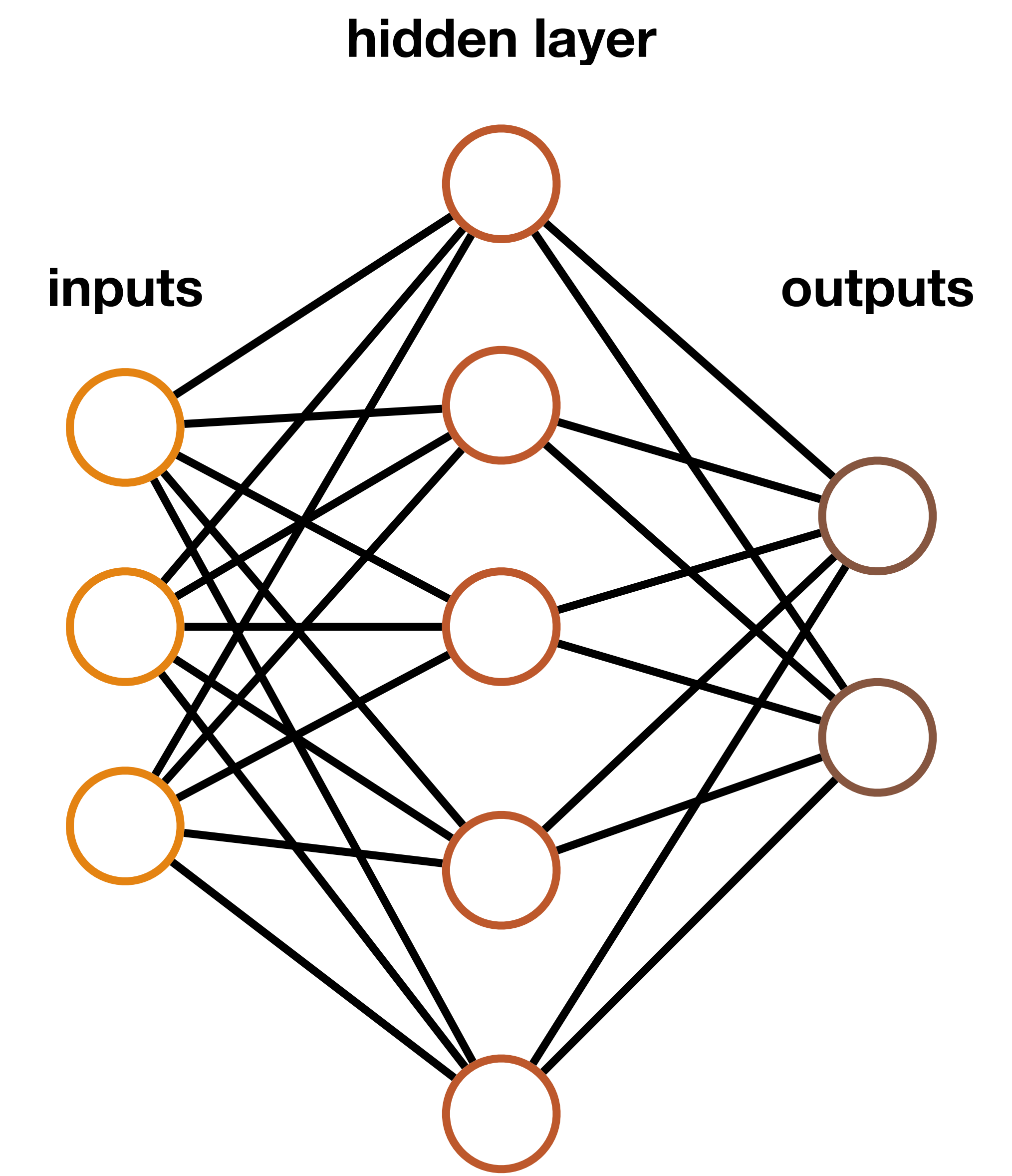
- Information flow in feed-forward (FF) networks:
  - Inputs → shallow layers → deeper layers → outputs
  - Alternative: recurrent NNs (information loops back)
- Multiple outputs $\implies$ efficiency:
  - Shared parameters, less data, less computation
- Multi-class classification:
  - One-hot labels $y = [0 \quad 0 \quad 1 \quad 0 \quad \cdots]$
  - Multilogistic regression (softmax): $\hat{y}_c = \frac{\exp(h_c)}{\sum_{\bar{c}} \exp(h_{\bar{c}})}$



Gradient computation

- MLPs are function compositions of single layers
  - Apply chain rule:





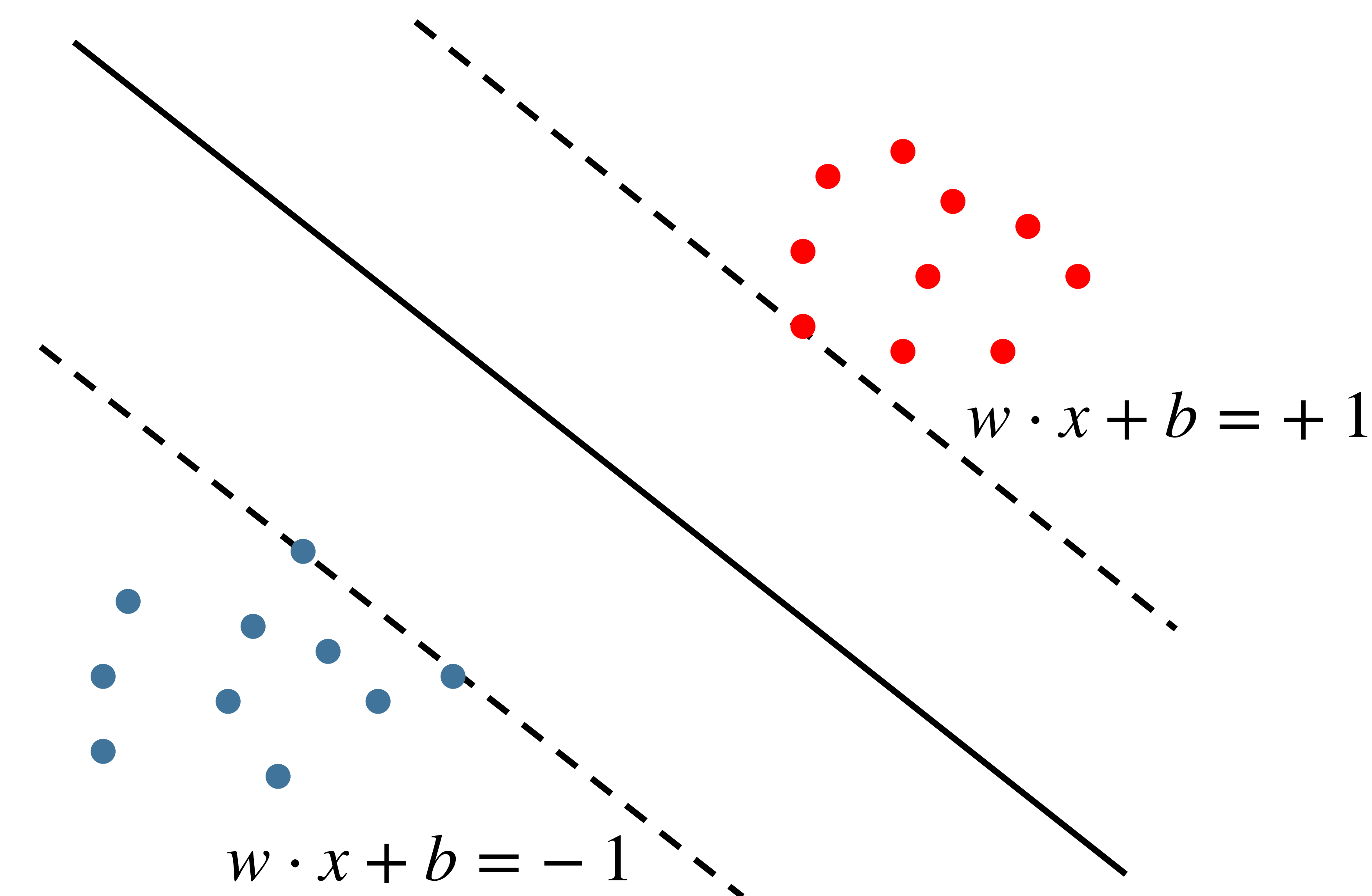
example: $f(g,h) = \sigma(g+h) \implies \partial_g f = f(1-f)$

$\implies$ **reuse f from the forward pass**

- Backpropagation = chain rule + dynamic programming to avoid repetitions

Maximizing the margin

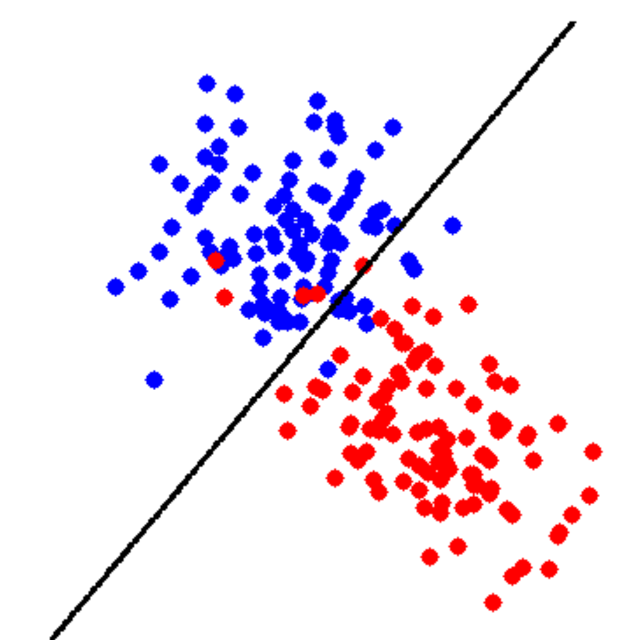
- Constrained optimization: get all data points correctly + maximize the margin
- $w^* = \arg\max_w \frac{2}{\|w\|} = \arg\min_w \|w\|$
  - such that all data points predicted with enough margin: $\begin{cases} w \cdot x^{(j)} + b \geq +1 & \text{if } y^{(j)} = +1 \\ w \cdot x^{(j)} + b \leq -1 & \text{if } y^{(j)} = -1 \end{cases}$
  - $\implies$ s.t. $y^{(j)}(w \cdot x^{(j)} + b) \geq 1$ (m constraints)



- Example of Quadratic Program (QP)
  - Quadratic objective, linear constraints

Soft margin: dual form

- Primal problem: $w^*, b^* = \arg\min_{w,b} \min_{\epsilon} \frac{1}{2}\|w\|^2 + R \sum_j \epsilon^{(j)}$
  - s.t. $y^{(j)}(w \cdot x^{(j)} + b) \geq 1 - \epsilon^{(j)}; \quad \epsilon^{(j)} \geq 0$



- Dual problem: $\max_{0 \leq \lambda \leq R} \sum_j \left(\lambda_j - \frac{1}{2} \sum_k \lambda_j \lambda_k y^{(j)} y^{(k)} x^{(j)} \cdot x^{(k)} \right)$ s.t. $\sum_j \lambda_j y^{(j)} = 0$
  - Optimally: $w^* = \sum_j \lambda_j y^{(j)} x^{(j)}$; to handle b: add constant feature $x_0 = 1$
  - Support vector = points on or inside margin = $\lambda_j > 0$
  - Gram matrix = $K_{jk} = x^{(j)} \cdot x^{(k)}$ = similarity of every pair of instances

Kernel SVMs

- Define kernel $K : (x, x') \mapsto \mathbb{R}$
- Solve dual QP: $\max_{0 \leq \lambda \leq R} \sum_j \left(\lambda_j - \frac{1}{2} \sum_k \lambda_j \lambda_k y^{(j)} y^{(k)} K(x^{(j)}, x^{(k)}) \right)$ s.t. $\sum_j \lambda_j y^{(j)} = 0$
- Learned parameters = λ (m parameters)
  - But also need to store all support vectors (having $\lambda_j > 0$)
- Prediction: $\hat{y}(x) = \text{sign}(w \cdot \Phi(x))$

$$= \text{sign}\left(\sum_j \lambda_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) \right) = \text{sign}\left(\sum_j \lambda_j y^{(j)} K(x^{(j)}, x) \right)$$

Bagging

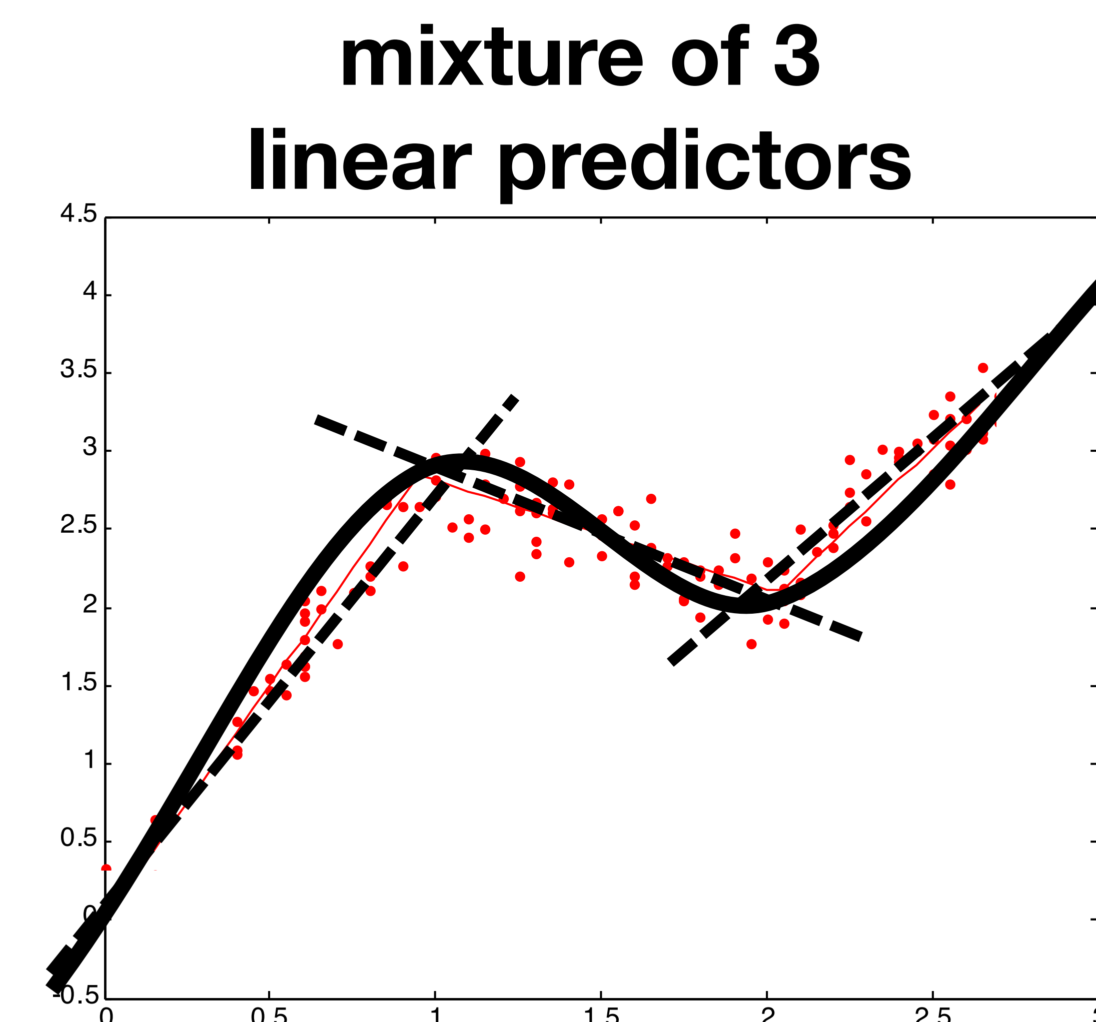
- Bagging = bootstrap aggregating:
  - Resample K datasets $\mathscr{D}_1, \ldots, \mathscr{D}_K$ of size b
  - Train K models $\theta_1, \ldots, \theta_K$ on each dataset
  - Regression: output $f_\theta : x \mapsto \frac{1}{K} \sum_k f_{\theta_k}(x)$
  - Classification: output $f_\theta : x \mapsto \text{majority}\{f_{\theta_k}(x)\}$
- Similar to cross-validation (for different purpose), but outputs average model
  - Also, datasets are resampled (with replacement), not a partition

Ensemble methods

- Ensemble = "committee" of models: $\hat{y}_k(x) = f_{\theta_k}(x)$
  - Decisions made by average / majority vote: $\hat{y}(x) = \frac{1}{K}\sum_k \hat{y}_k(x)$
  - May be weighted: better model = higher weight: $\hat{y}(x) = \sum_k \alpha_k \hat{y}_k(x)$
- Stacking = use ensemble as inputs (as in MLP): $\hat{y}(x) = f_\theta(\hat{y}_1(x), \dots, \hat{y}_K(x))$
  - f_θ trained on held out data = validation of which model should be trusted
  - f_θ linear $\implies$ weighted committee, with learned weights

Mixture of Experts (MoE)

- Experts = models can "specialize", good only for some instances
  - Let weights depend on x: $\hat{y}(x) = \sum_k \alpha_k(x)\hat{y}_k(x)$



- Can we predict which model will perform well?
  - Learn a predictor $\alpha_\phi(k \mid x)$
    - E.g., multilogistic regression (softmax) $\alpha_\phi(k \mid x) = \frac{\exp(\phi_k \cdot x)}{\sum_{k'} \exp(\phi_{k'} \cdot x)}$
- Loss, experts, weights differentiable $\implies$ end-to-end gradient-based learning

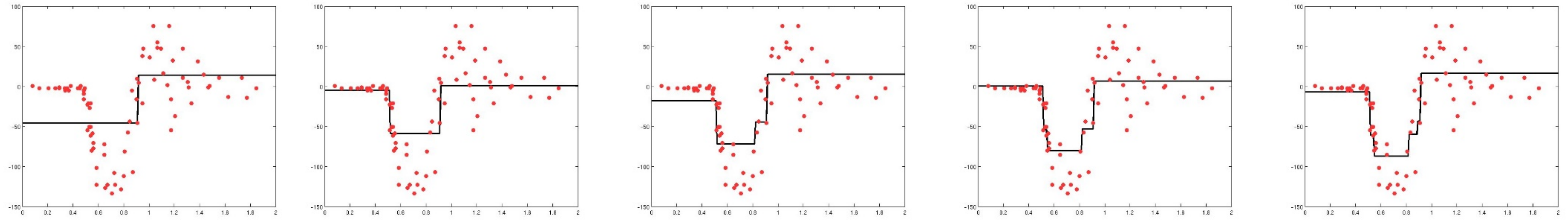
Random Forests

- Bagging over decision trees: which feature at root?
  - Much data $\implies$ max info gain stable across data samples
  - Little diversity among models $\implies$ little gained from ensemble
- Random Forests = subsample features
  - Each tree only allowed to use a subset of features
  - Still low, but higher bias
  - Average over trees for lower variance
- Works very well in practice $\implies$ go-to algorithm for small ML tasks

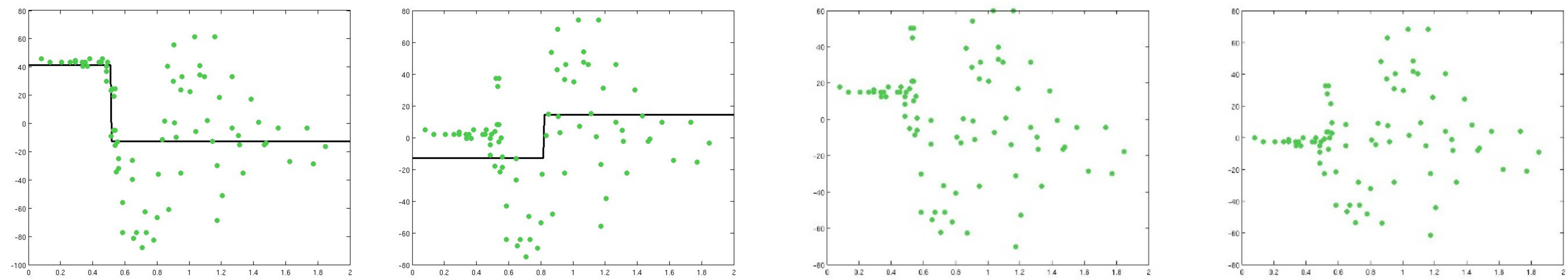
Gradient Boosting example: MSE loss

- Ensemble: $\hat{y}_K = \sum_k f_k(x)$; MSE loss: $\mathscr{L}(y, \hat{y}_k) = \frac{1}{2}(y - \hat{y}_{k-1} - f_k(x))^2$
  - To minimize: have $f_k(x)$ try to predict $y - \hat{y}_{k-1}$
  - Then update $\hat{y}_k = \hat{y}_{k-1} + f_k(x)$

data
prediction



residual
weak model



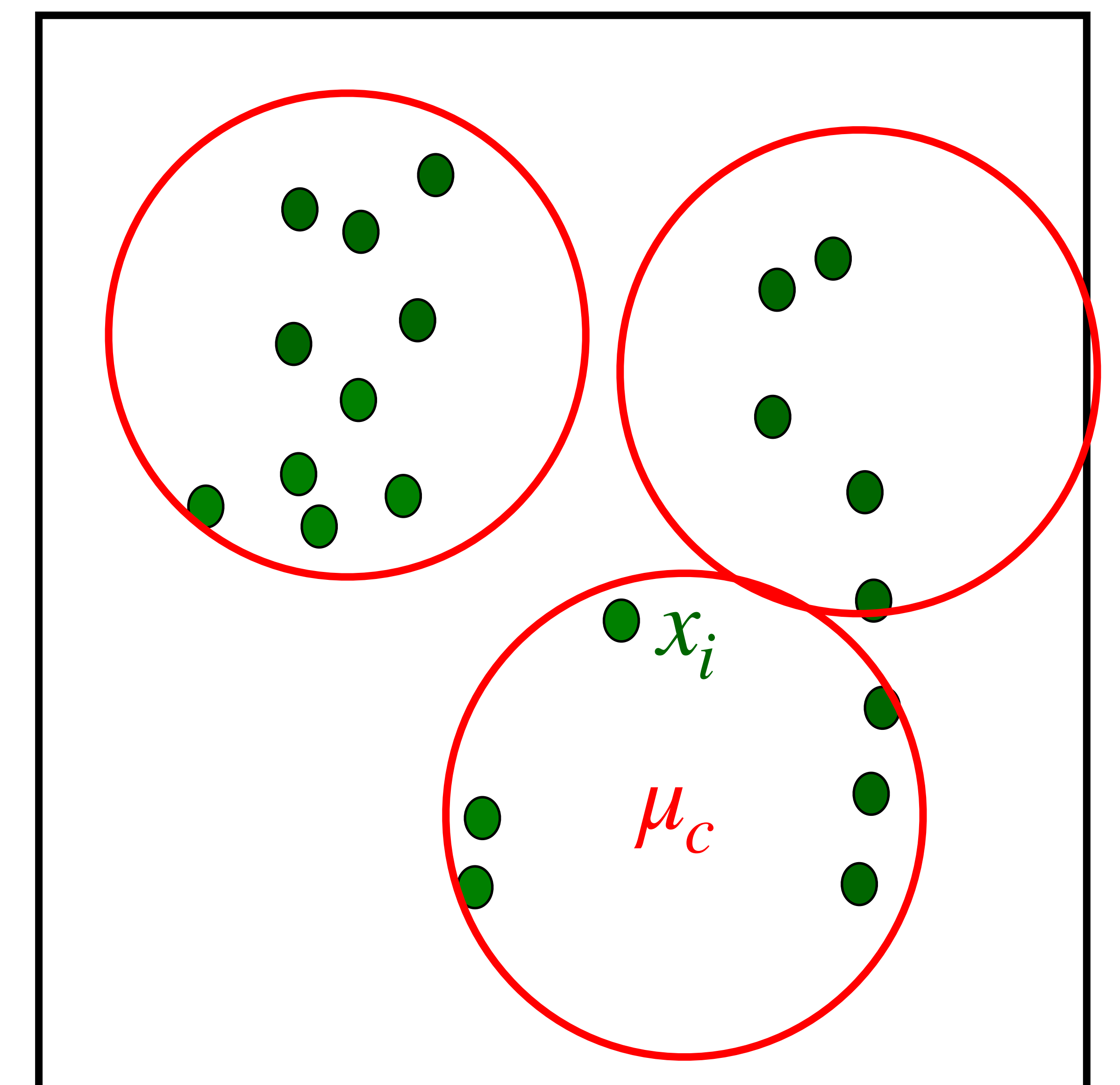
increasingly accurate
increasingly complex

AdaBoost

- AdaBoost = adaptive boosting:
  - Initialize $w_0^{(j)} = \frac{1}{m}$
  - Train classifier f_k on training data with weights w_{k-1}
  - Compute weighted error rate $\epsilon_k = \frac{\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})]}{\sum_j w_{k-1}^{(j)}}$
  - Compute $\alpha_k = \frac{1}{2} \ln \frac{1 - \epsilon_k}{\epsilon_k}$
  - Update weights $w_k^{(j)} = w_{k-1}^{(j)} e^{-y^{(j)} \alpha_k f_k(x^{(j)})}$ (increase weight for misclassified points)
- Predict $\hat{y}(x) = \text{sign} \sum_k \alpha_k f_k(x)$

k-Means

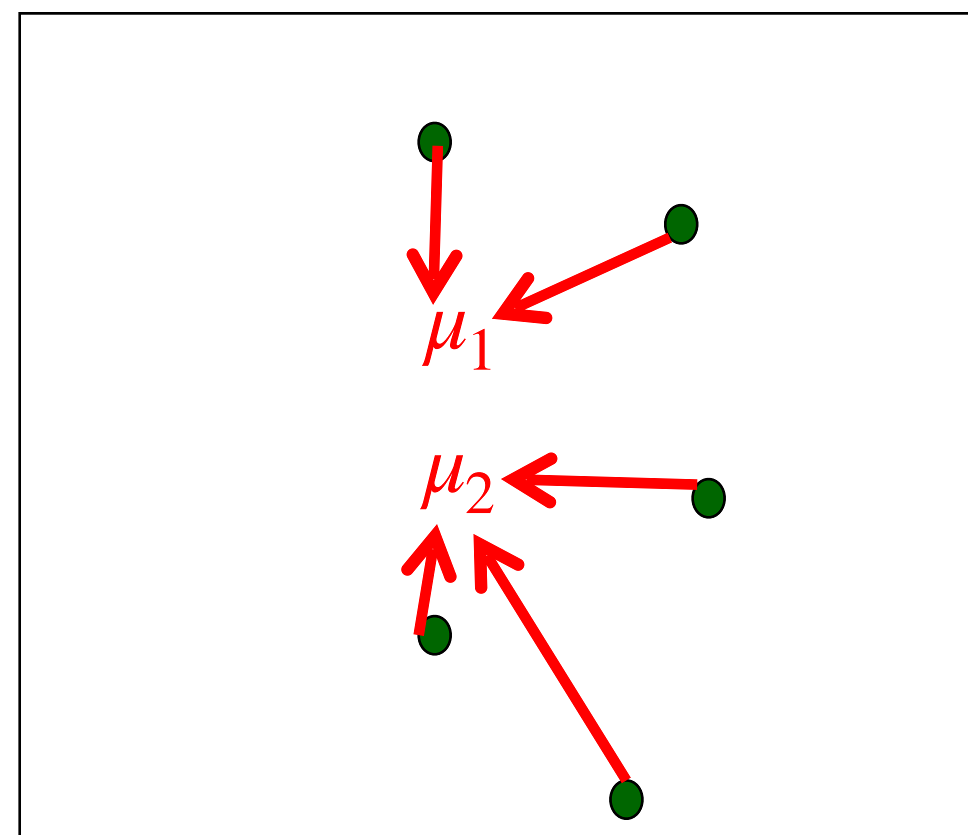
- Simple clustering algorithm
- Repeat:
    - Update the clustering = assignment of data points to clusters
    - Update the cluster's representation to match the assigned points
- Notation:
    - x_i = data point in the dataset
    - k = number of clusters
    - μ_c = representation of cluster c

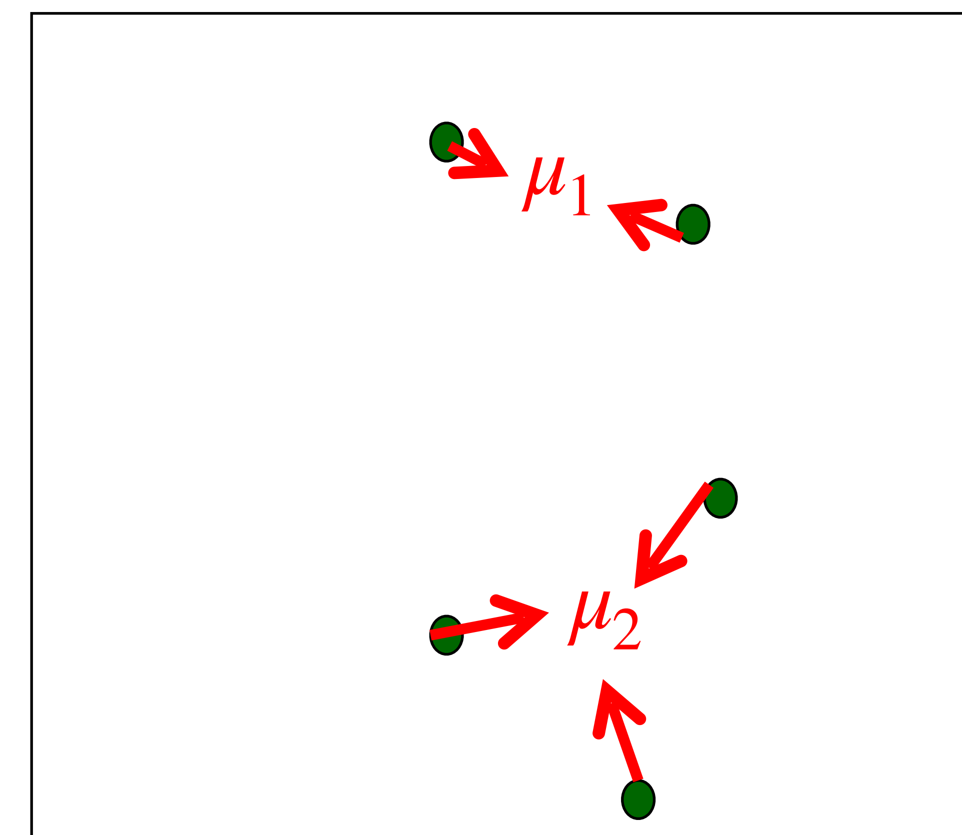


k-Means

- k-Means optimizes the MSE loss: $\mathscr{L}(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$

- Iterate until convergence:
  - For each $x_i \in \mathscr{D}$, find the closest cluster: $z_i = \arg\min_c \|x_i - \mu_c\|^2$
  - Set each cluster centroid μ_c to the mean of assigned points: $\mu_c = \frac{1}{m_c} \sum_{i:z_i=c} x_i$





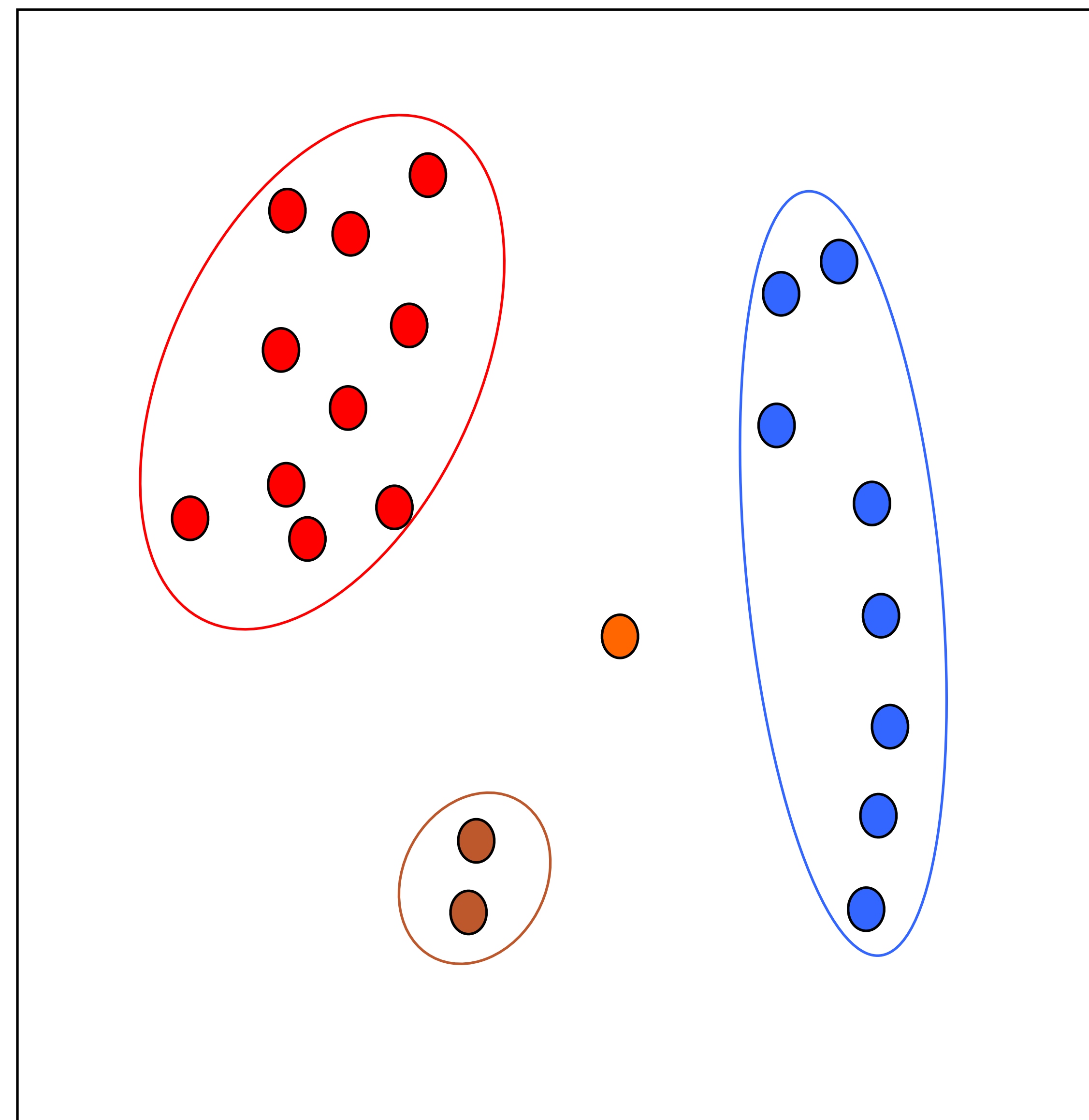
Hierarchical agglomerative clustering

- Another simple clustering algorithm
- Define distance (dissimilarity) between clusters $d(C_i, C_j)$
- Initialize: every data point is its own cluster
- Repeat:
  - Compute distance between each pair of clusters
  - Merge two closest clusters
- Output: tree of merge operations ("dendrogram")
- Complexity: in $m-1$ iterations, merge distances and sort $\implies O(m^2 \log m)$

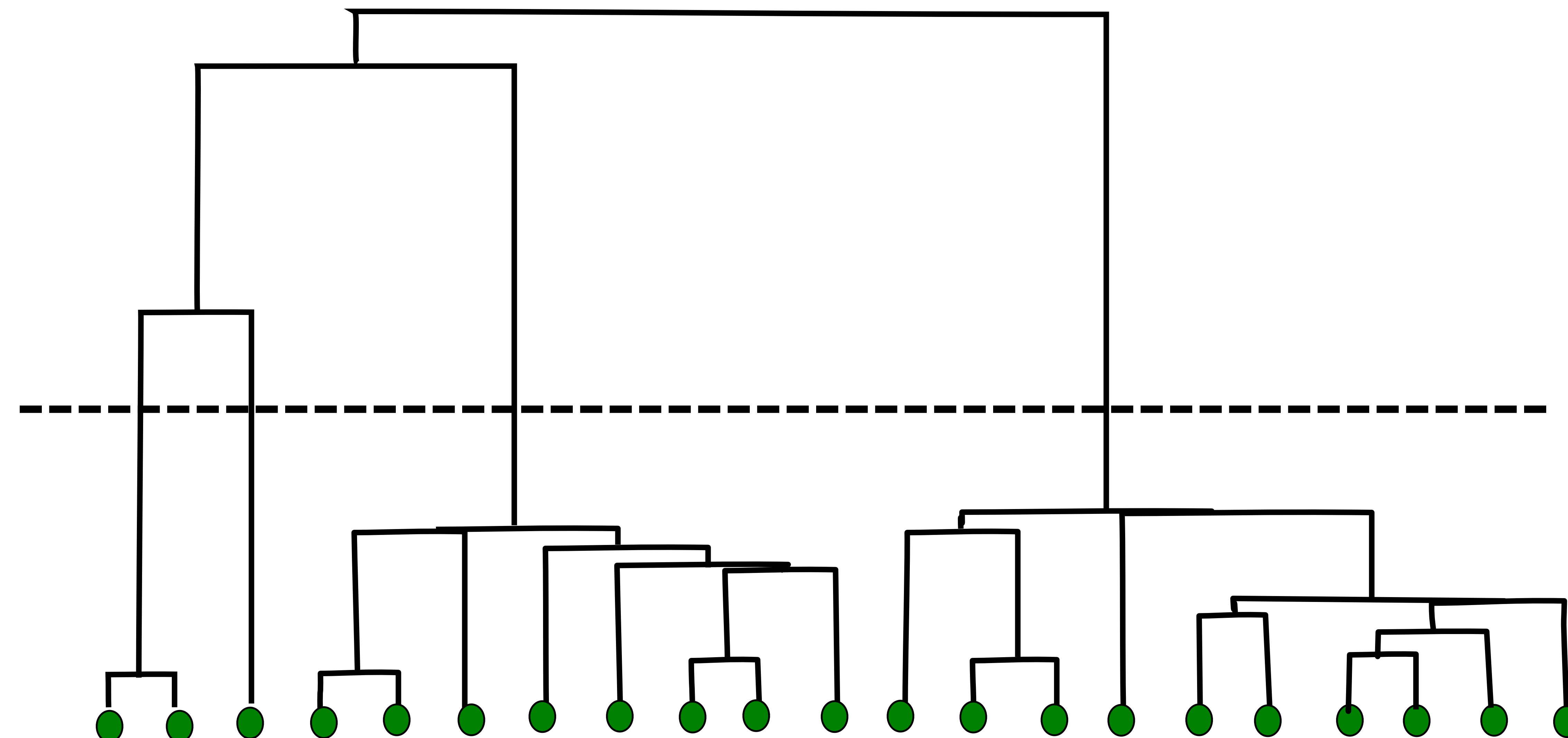
From dendrogram to clusters

- Given the hierarchy of clusters, choose a frontier of subtrees = clusters

data



dendrogram



- For a given k, or a given level of dissimilarity

Distance measures

- $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|^2$

- $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|^2$

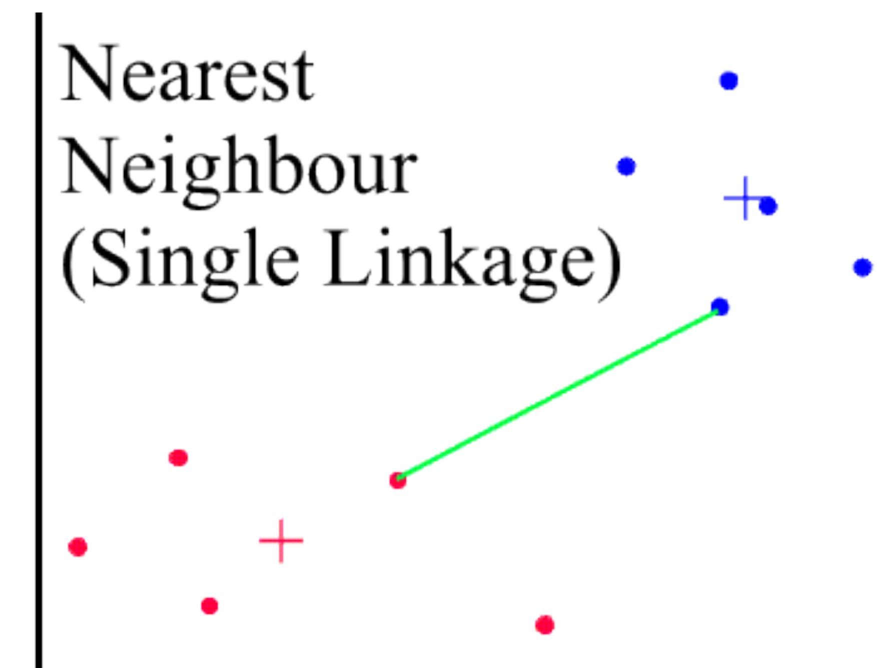
- $d_{\text{avg}}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i, y \in C_j} \|x - y\|^2$

- $d_{\text{means}}(C_i, C_j) = \|\mu_i - \mu_j\|^2$

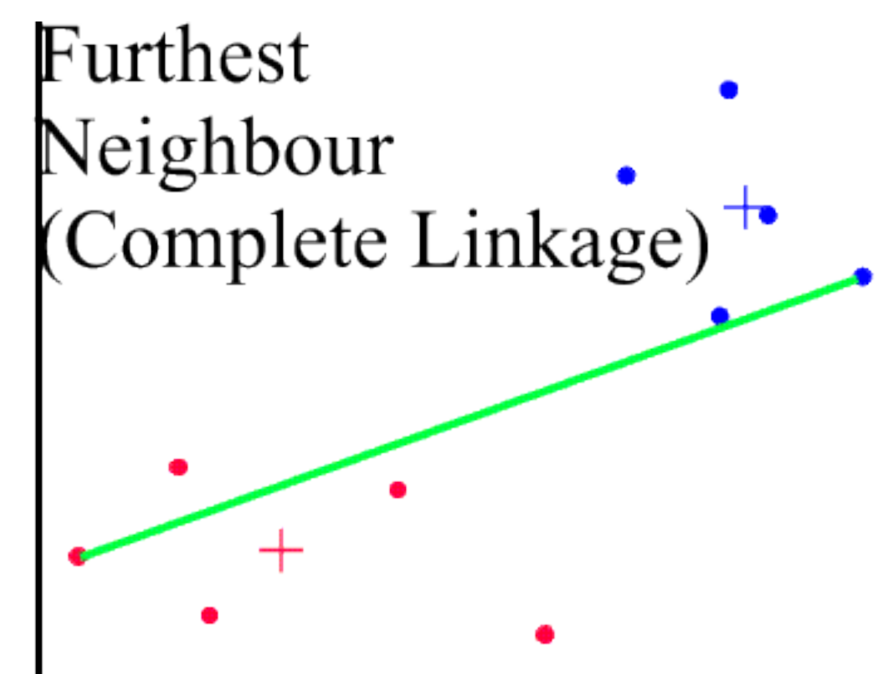
- Important property: iterative computation

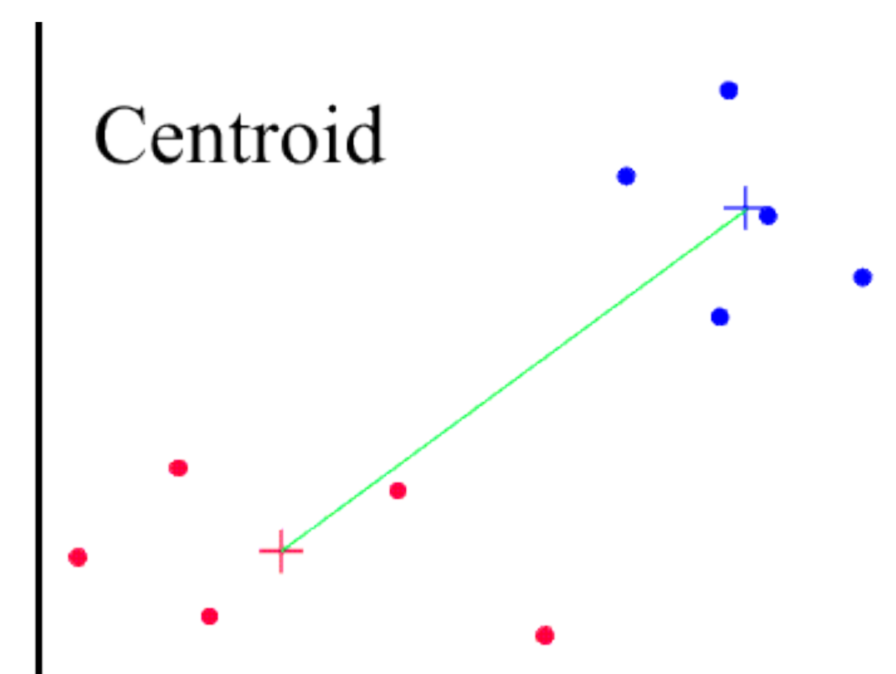
$$d(C_i \cup C_j, C_k) = f(d(C_i, C_k), d(C_j, C_k))$$

produces minimum spanning tree



avoids elongated clusters

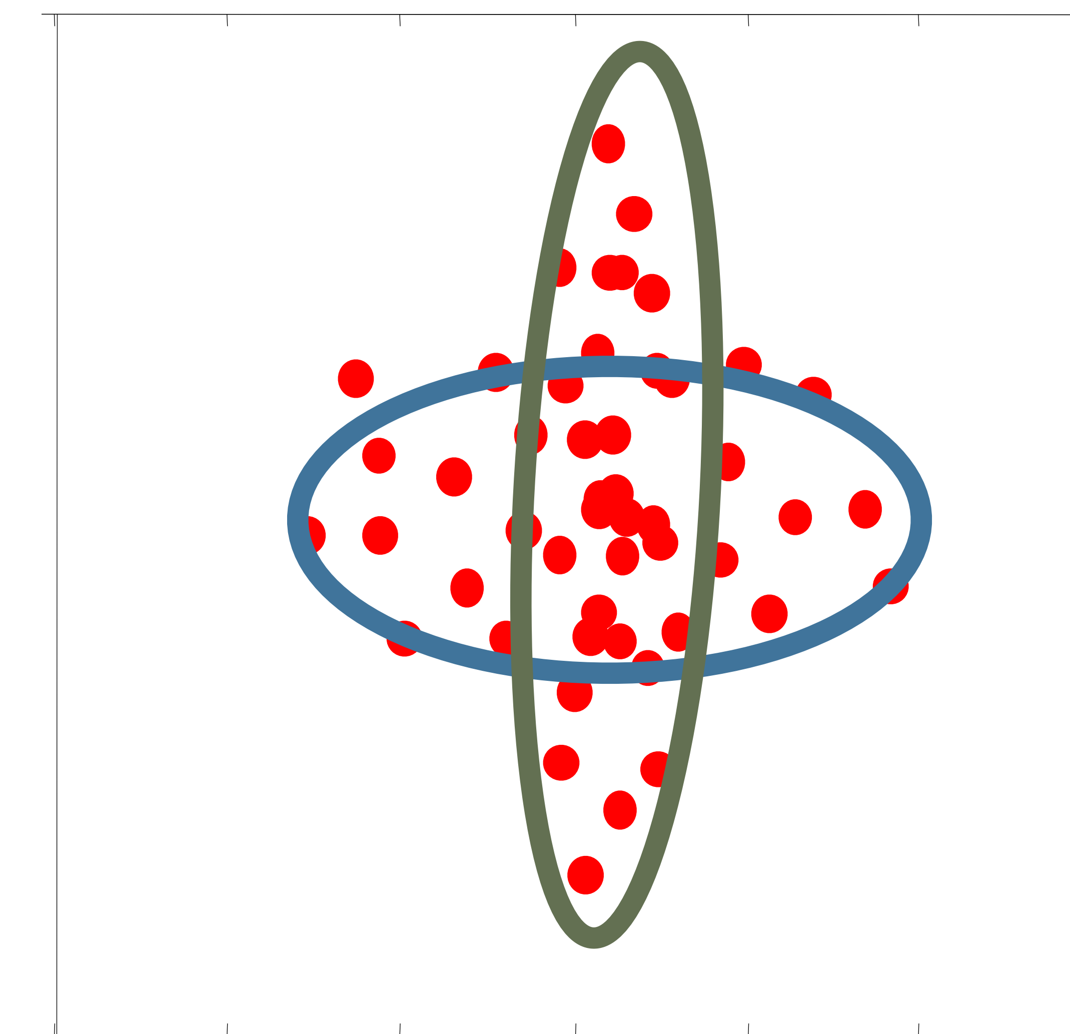




Gaussian Mixture Models (GMMs)

- Each cluster is modeled by a Gaussian $p(x \mid c) = \mathcal{N}(x; \mu_c, \Sigma_c)$
  - Σ_c allows non-isotropic clusters $\Longrightarrow$ weighted Euclidean distance
- Mixture = distribution over Gaussians is given by a probability vector $p(c)$
- Generative model = we can sample $p(x)$:
  - Sample $z \sim p(c)$
  - Sample $x \sim p(x \mid c = z)$
  - Probability of this x: $\sum_c p(c = z) p(x \mid c = z) = \sum_c p(c, x) = p(x)$

we don't output z, it is "latent" = hidden
$\Longrightarrow$ can be any of them



Training GMMs

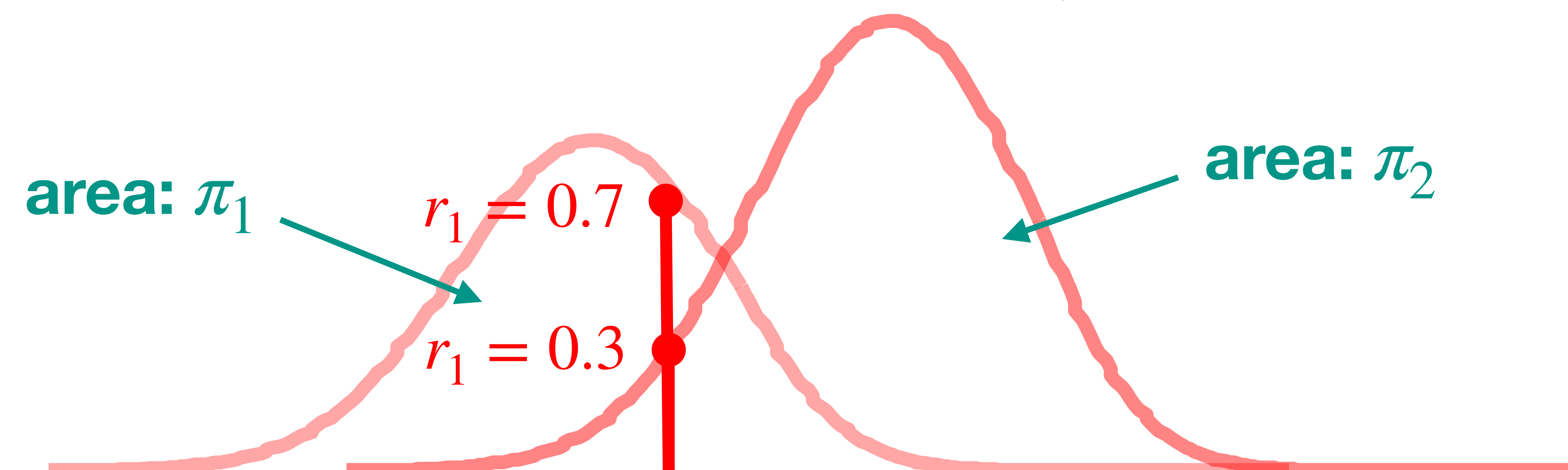
- k-Means:
  - Assign data points to clusters z_i
  - Update each cluster's parameters μ_c
- A “soft” version of k-Means: Expectation–Maximization (EM) algorithm
  - Find a “soft” assignment $p(c|x)$
  - Update model parameters $p(c)$, $p(x|c)$
- The EM algorithm is extremely general, GMMs are a very special case

Expectation–Maximization: E-step

- Initialize model parameters $\pi_c = p(c)$, μ_c, Σ_c
- E-step (Expectation): [why "expectation"? comes from the general EM algorithm]
  - For each data point x_i, use Bayes' rule to compute:

$$r_{ic} = p(c \,|\, x_i) = \frac{p(c)p(x_i \,|\, c)}{\sum_{\bar{c}} p(\bar{c})p(x_i \,|\, \bar{c})} = \frac{\pi_c \mathcal{N}(x_i; \mu_c, \Sigma_c)}{\sum_{\bar{c}} \pi_{\bar{c}} \mathcal{N}(x_i; \mu_{\bar{c}}, \Sigma_{\bar{c}})}$$

  - High weight to clusters that are likely a-priori, or in which x_i is relatively probable



Expectation–Maximization: M-step

- Given assignment probabilities r_{ic}
- M-step (Maximization):
  - For each cluster c, fit the best Gaussian to the weighted assignment

total weight assigned to cluster c

$$m_c = \sum_i r_{ic}$$

what is $\sum_c m_c$? m

fraction of weight assigned to cluster c

$$\pi_c = \frac{m_c}{m} \qquad \mu_c = \frac{1}{m_c} \sum_i r_{ic} x_i$$

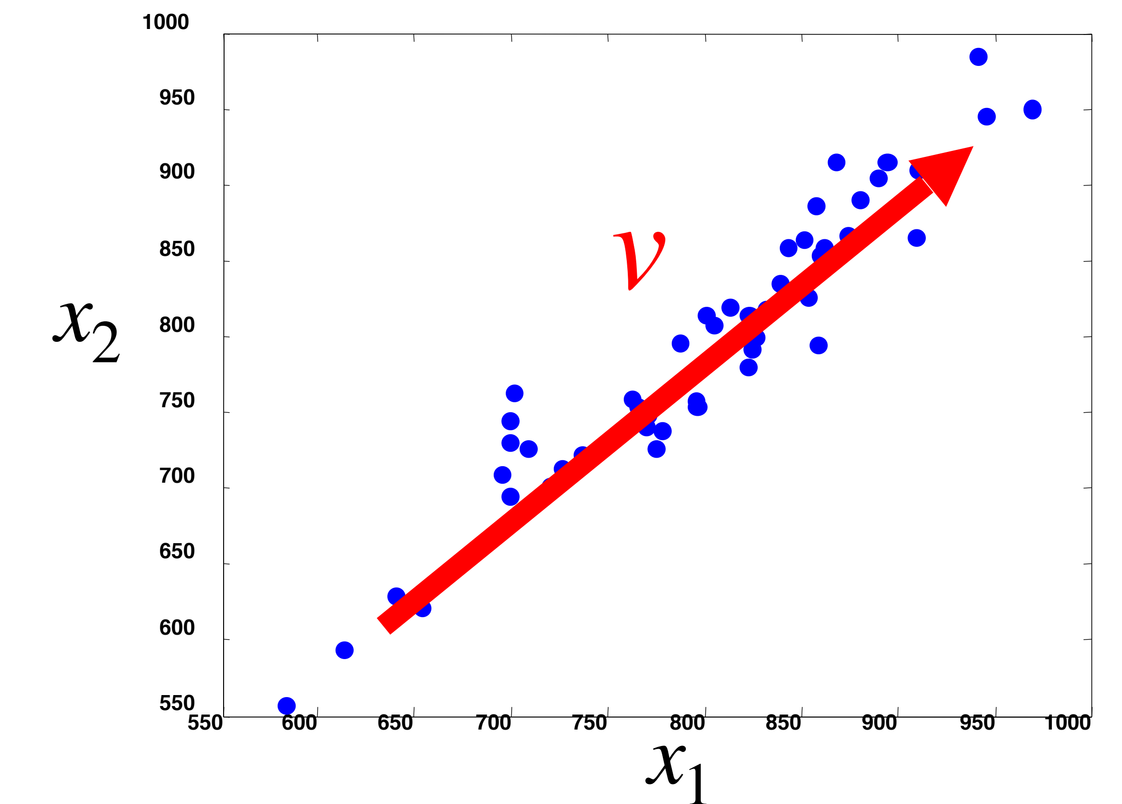
weighted mean of data in cluster c

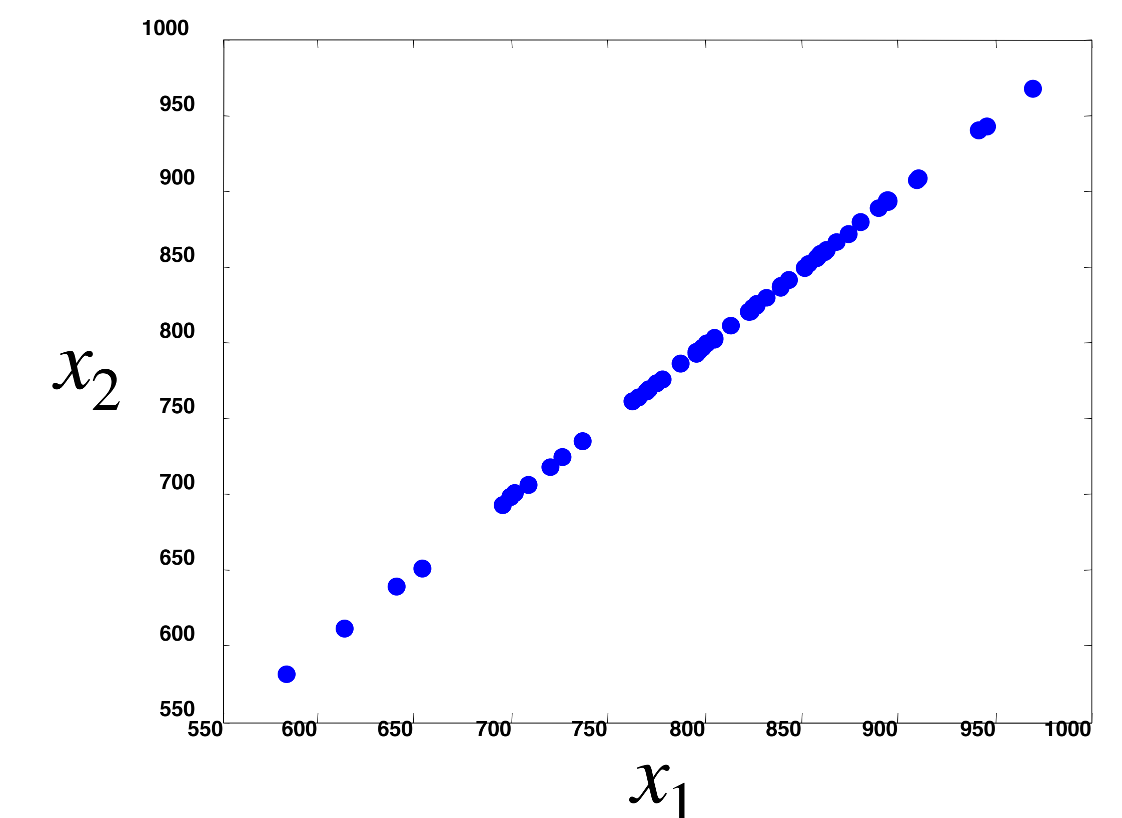
$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic}(x_i - \mu_c)(x_i - \mu_c)^\intercal$$

weighted covariance of data in cluster c

Dimensionality reduction: 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
    - $\implies z = x \cdot v$





Singular Value Decomposition (SVD)

- Alternative method for finding covariance eigenvectors
  - Has many other uses
- Singular Value Decomposition (SVD): $X = UDV^\intercal$
  - U and V (left- and right singular vectors) are orthogonal: $U^\intercal U = I$, $V^\intercal V = I$
  - D (singular values) is rectangular-diagonal
  - $\Sigma = X^\intercal X = VD^\intercal U^\intercal UDV^\intercal = V(D^\intercal D)V^\intercal$

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

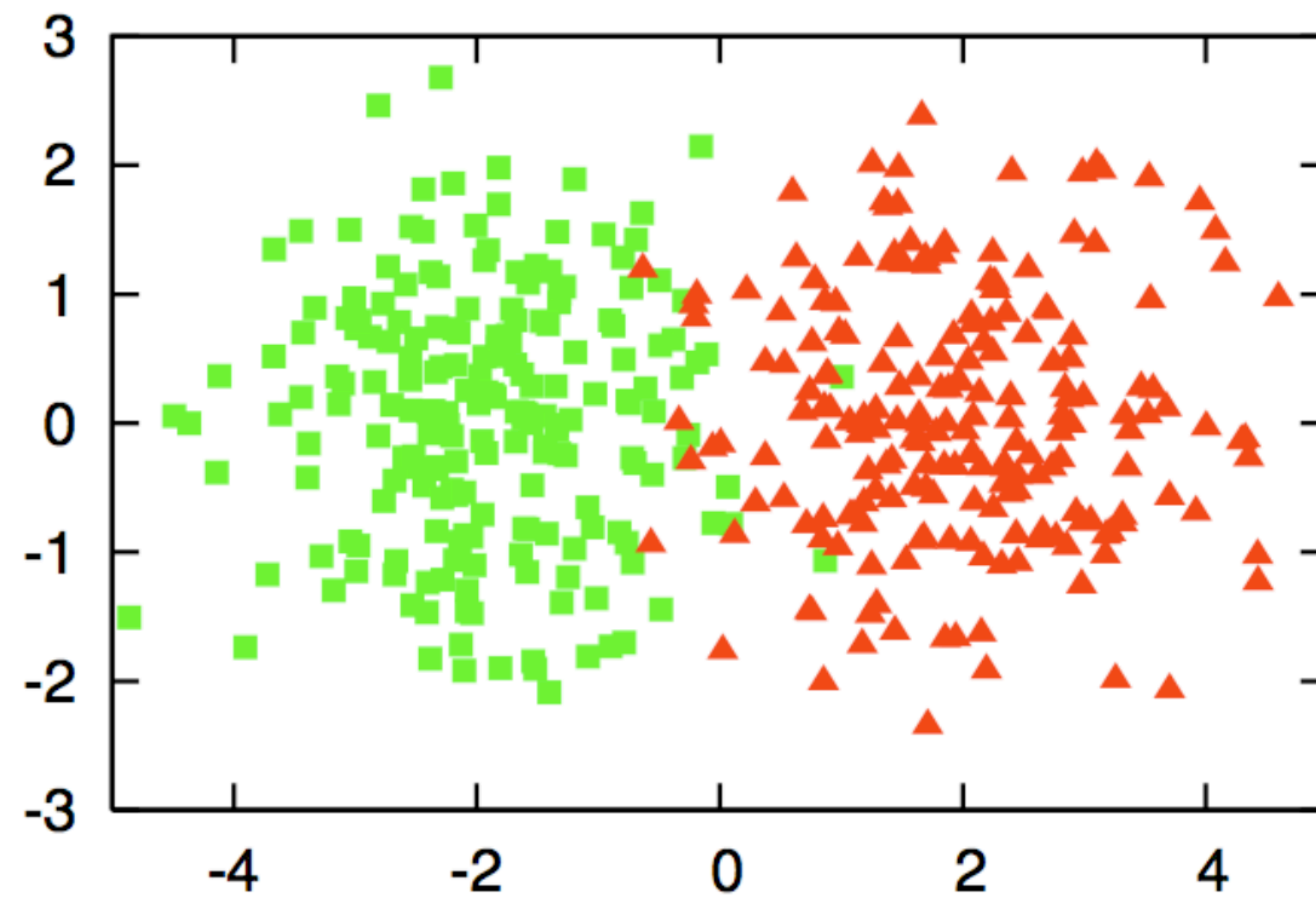
- UD matrix gives coefficients to reconstruct data: $x_i = U_{i,1}D_{1,1}v_1 + U_{i,2}D_{2,2}v_2 + \cdots$
  - We can truncate this after top k singular values (square root of eigenvalues)

Latent-space models: extensions

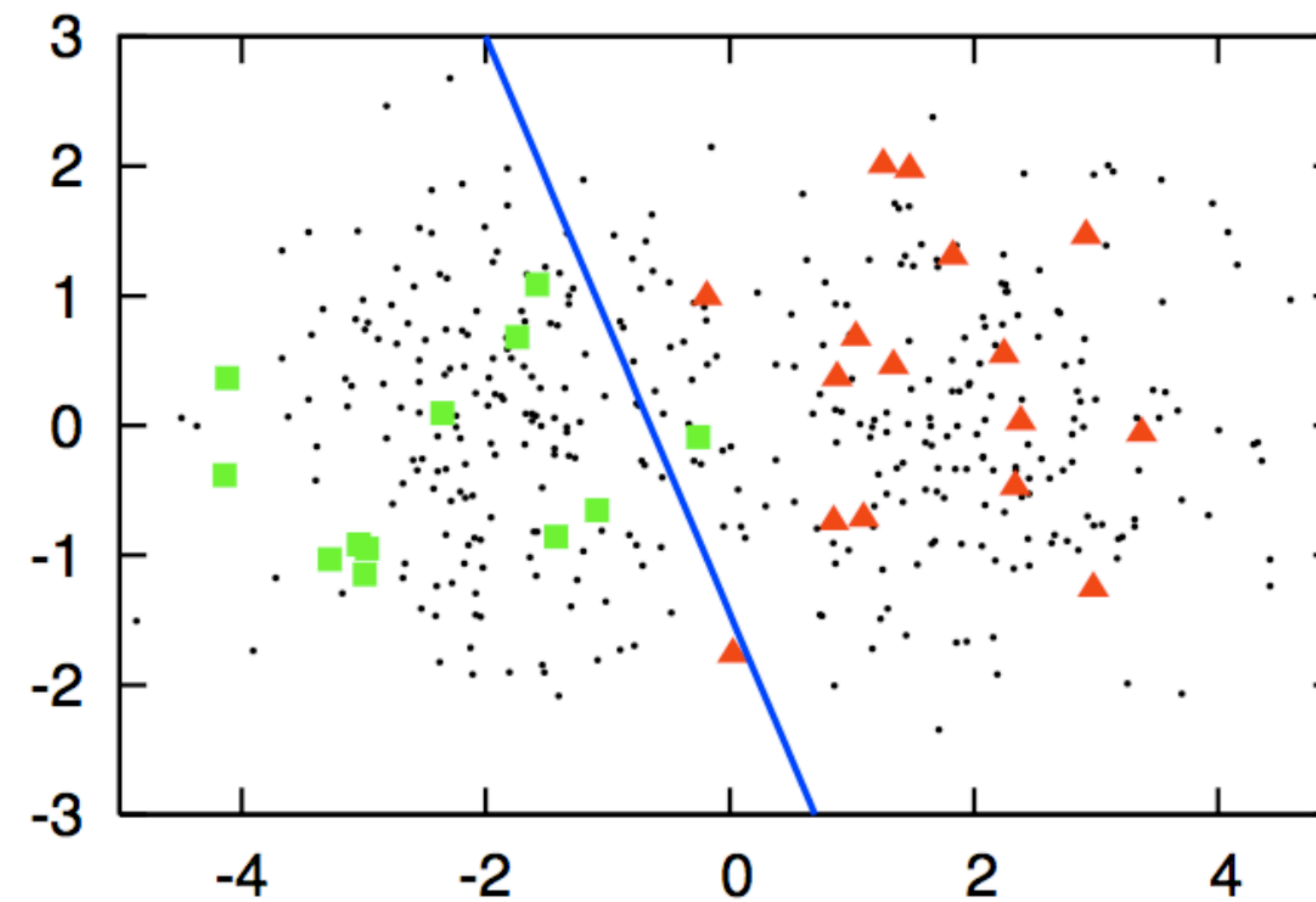
- Add lower-order terms: $r_{mu} \approx \mu + b_m + b_u + \sum_k U_{mk} V_{ku}$
  - μ = overall average rating (affected by user interface etc.)
  - $b_m + b_u$ = item and user biases
- Can add non-linearity (saturating?)
- Gradient decent on loss, e.g. MSE : $\mathscr{L}(U, V) = \sum_{u,m} \left(X_{mu} - \sum_k U_{mk} V_{ku} \right)^2$
- Train using a gradient-based optimizer

Why active learning?

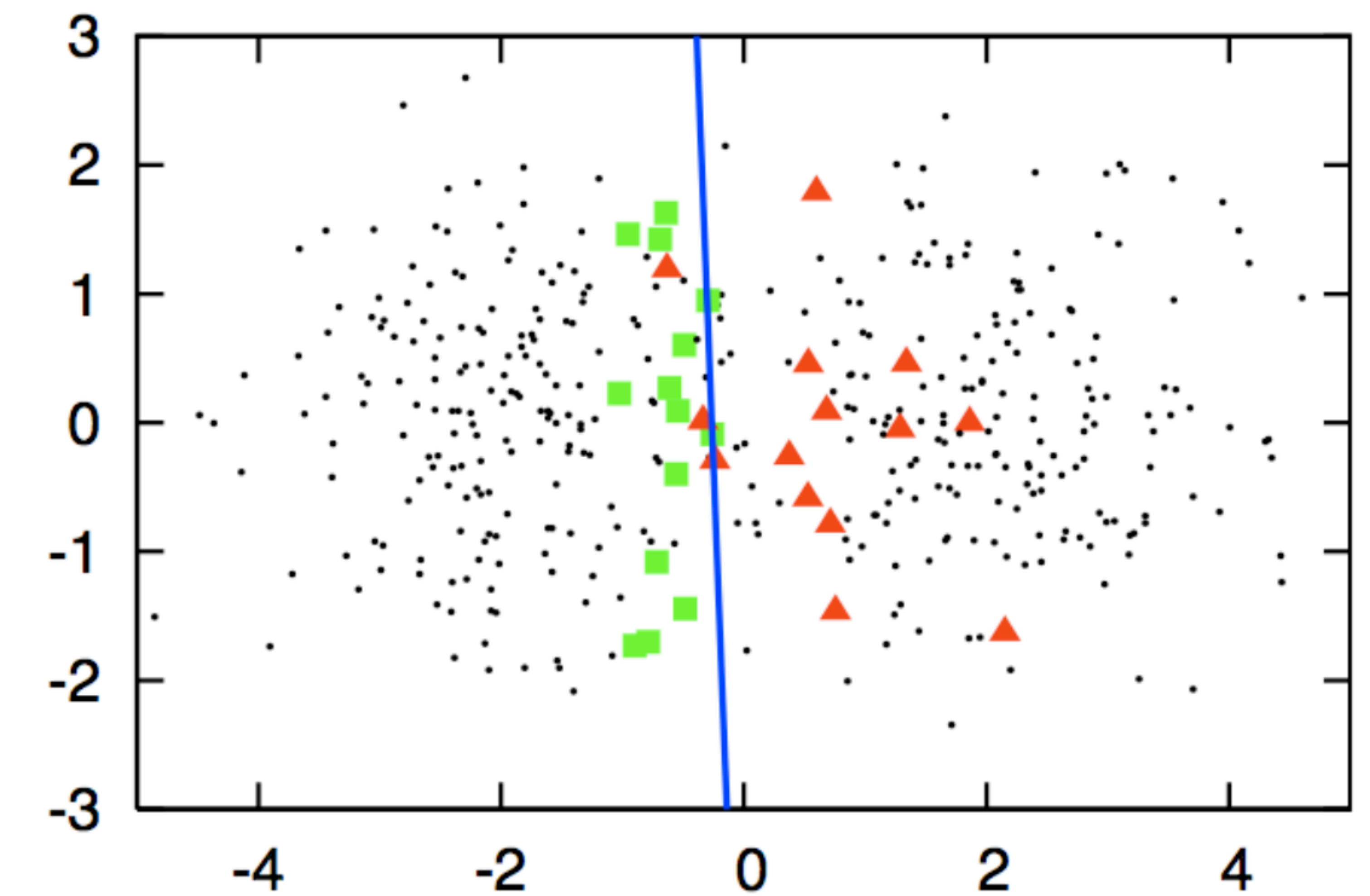
full labeled data (unavailable)



SVM on random sample of labeled data



SVM on selected sample of labeled data

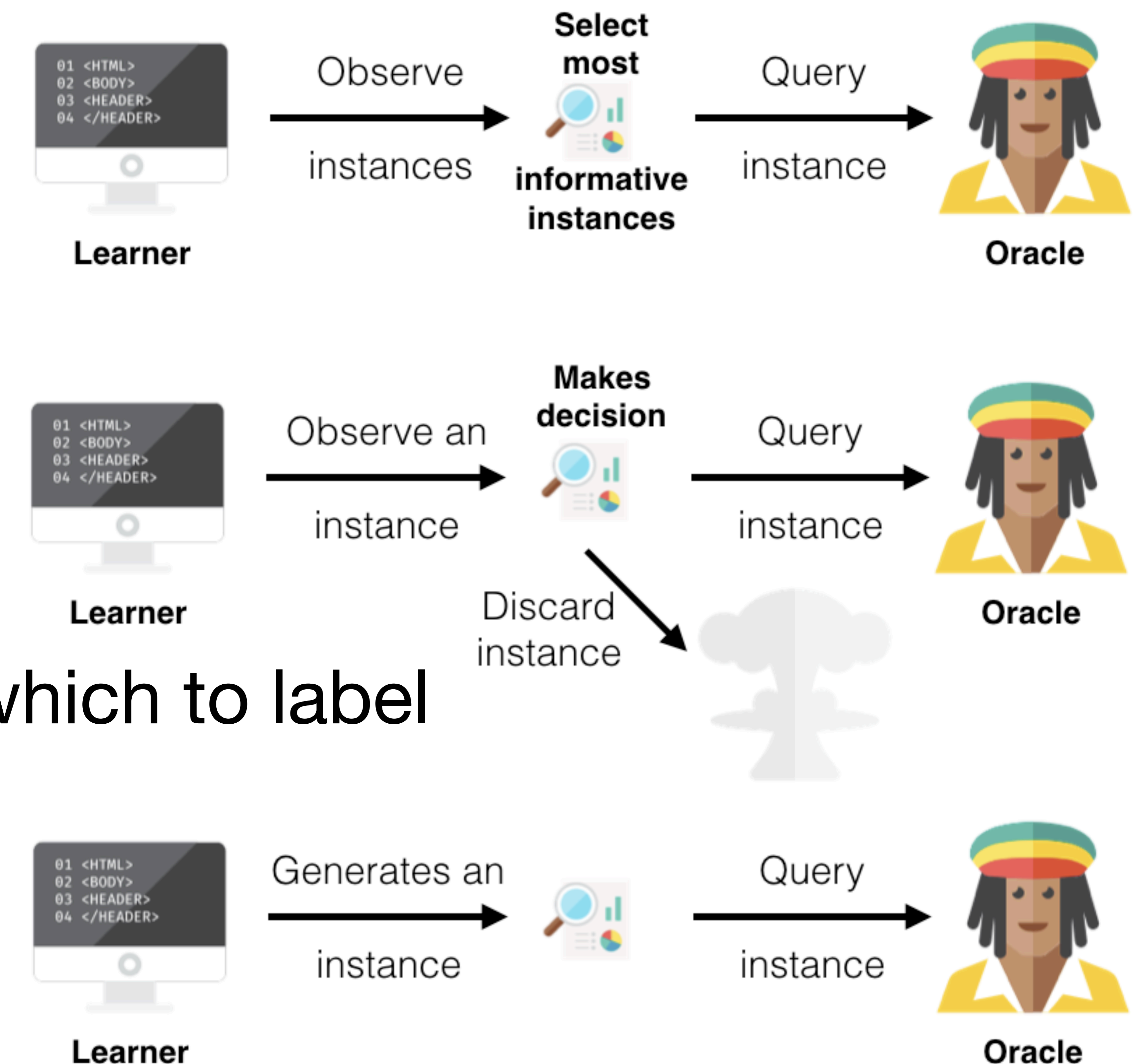


Source: https://www.datacamp.com/community/tutorials/active-learning

- 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
    - $\Longrightarrow$ May be harder for teacher to label ("is this synthesized image a dog or a cat?")

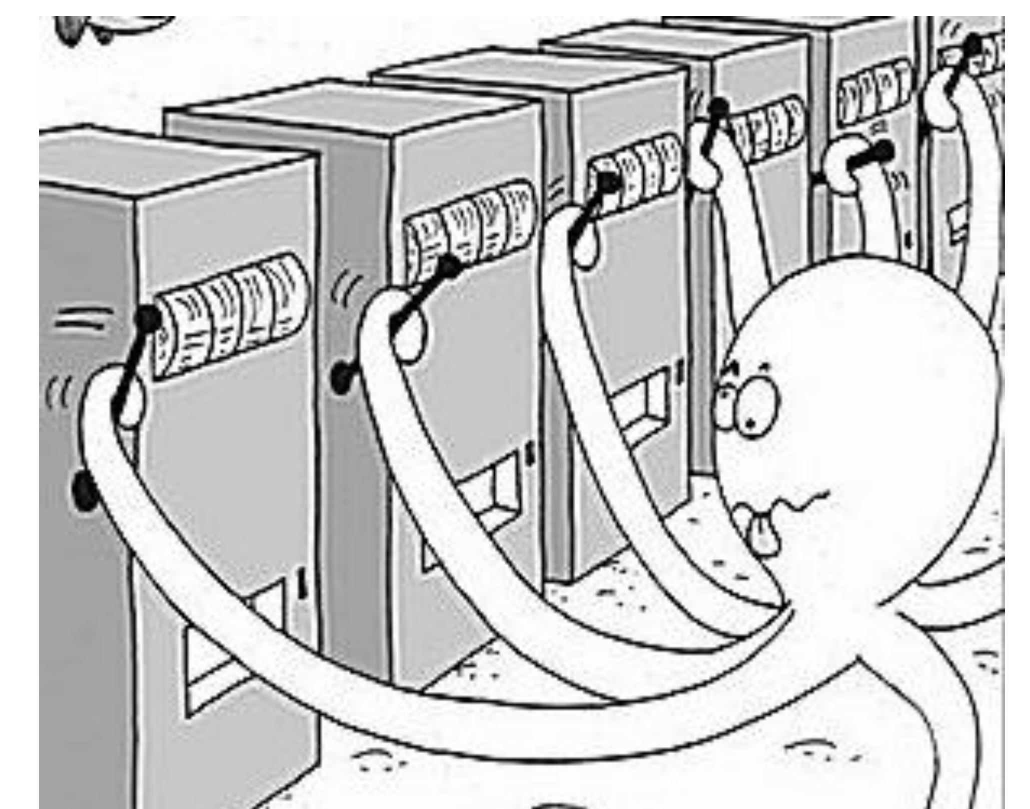


Multi-Armed Bandits (MABs)

- 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

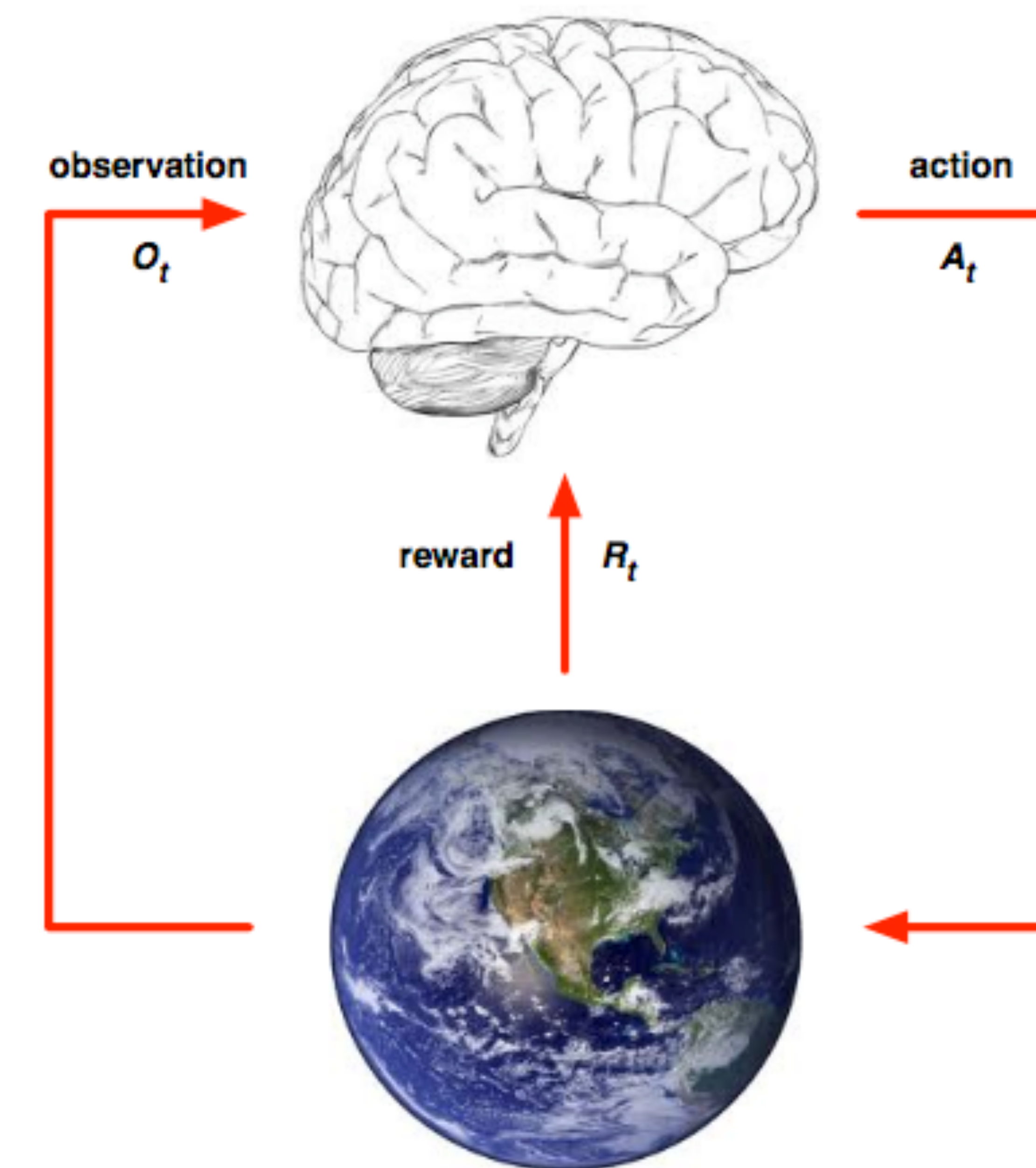


Optimism under uncertainty

- 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 $\Longrightarrow$ let c grow
  - But not too fast, or we fail to exploit what we do know
- Regret: $\rho(T) = O(\log T)$, provably optimal

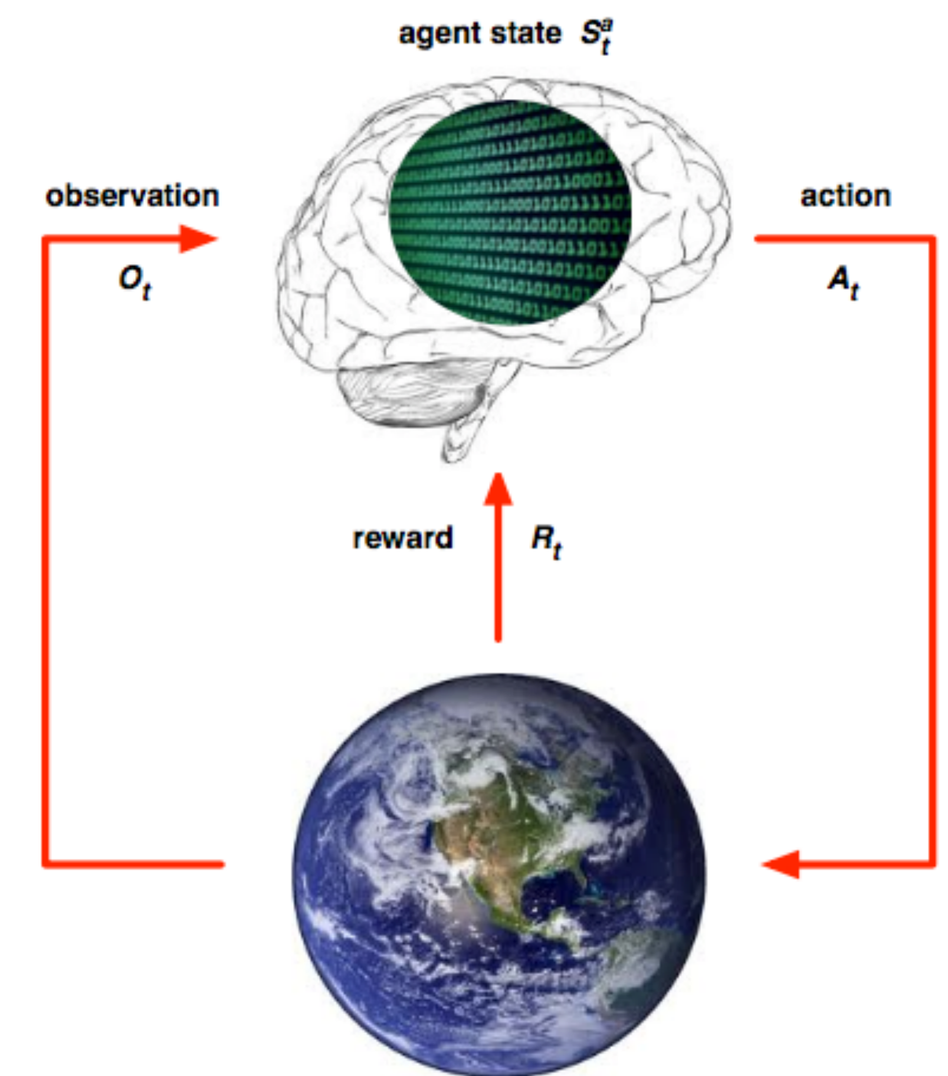
Agent–environment interface

- Environment
  - Executes the action → changes its state
  - Generates next observation
  - Supervisor: reveals the reward
- Agent
  - Policy decides on next action $\pi(a_t | x_t)$
  - Context can be full state $x_t = s_t$
    - Or any summary of observable history $x_t = f(h_t)$



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 $x_t = f(h_t)$



Markov Property

"The future is independent of the past given the present"

Definition

A state S_t is *Markov* if and only if

$$\mathbb{P}[S_{t+1} \mid S_t] = \mathbb{P}[S_{t+1} \mid S_1, \dots, S_t]$$

- The state captures all relevant information from the history
- Once the state is known, the history may be thrown away
- i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \textit{from} \overset{\textit{to}}{\begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}}$$

where each row of the matrix sums to 1.

Return as expected future reward

Definition

The *return* G_t is the total discounted reward from time-step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

- The *discount* $\gamma \in [0, 1]$ is the present value of future rewards
- The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$.
- This values immediate reward above delayed reward.
  - γ close to 0 leads to "myopic" evaluation
  - γ close to 1 leads to "far-sighted" evaluation

Markov Decision Process

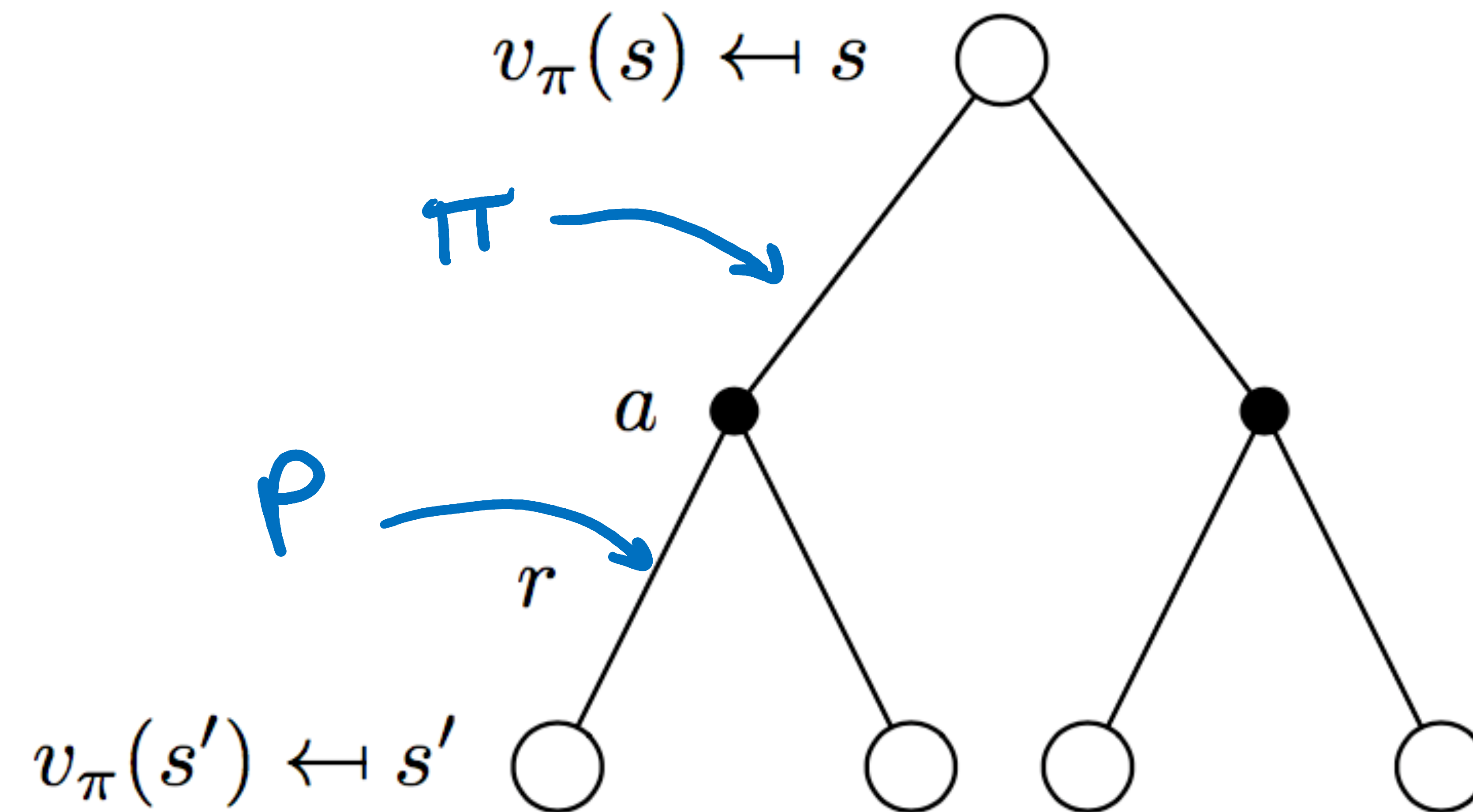
A Markov decision process (MDP) is a Markov reward process with decisions. It is an *environment* in which all states are Markov.

Definition

A *Markov Decision Process* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'}^{a} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$
- γ is a discount factor $\gamma \in [0, 1]$.

Bellman Expected Equation, V



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

Bellman Exp Eq: Matrix Form

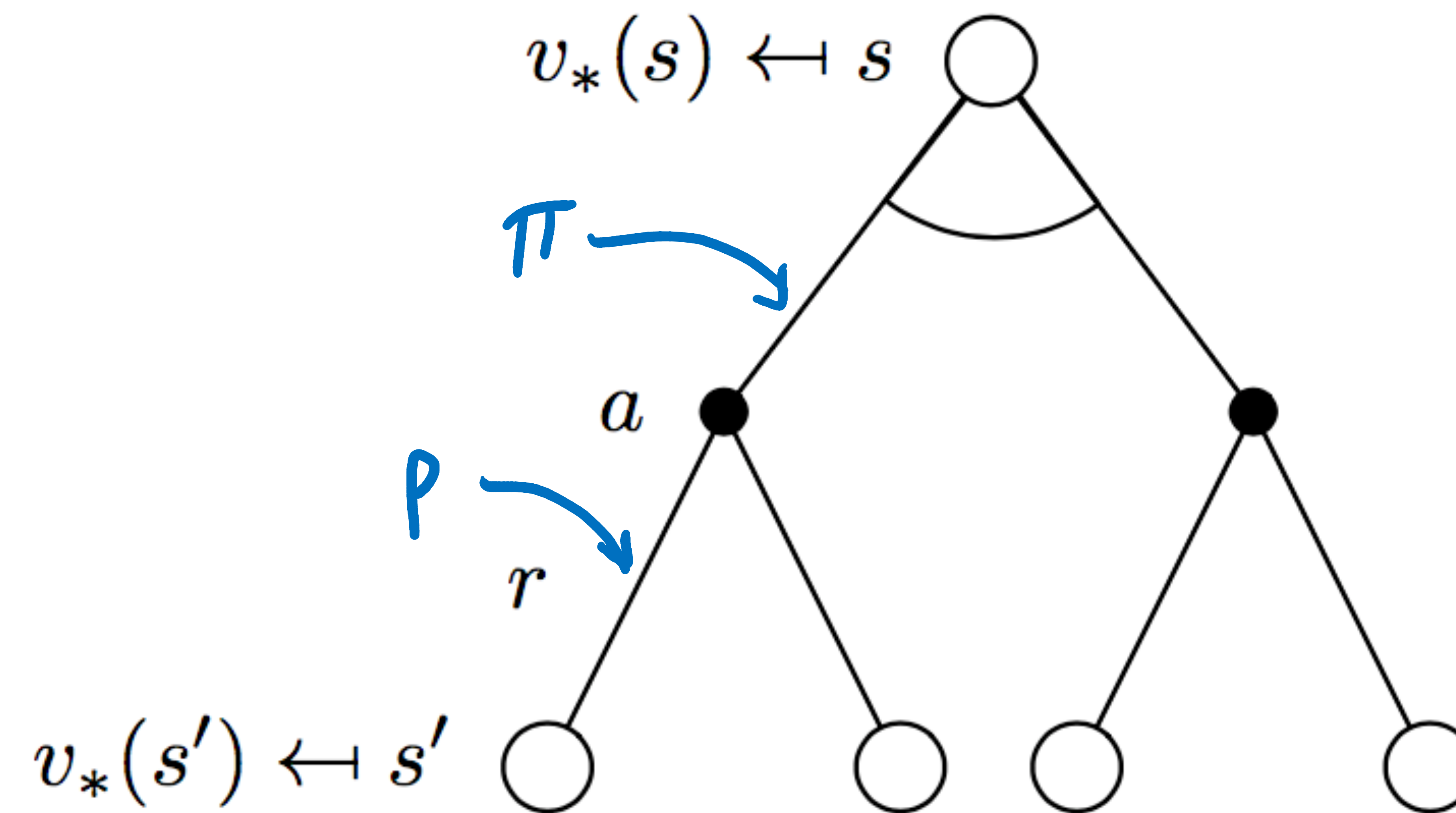
The Bellman expectation equation can be expressed concisely using the induced MRP,

$$v_\pi = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi v_\pi$$

with direct solution

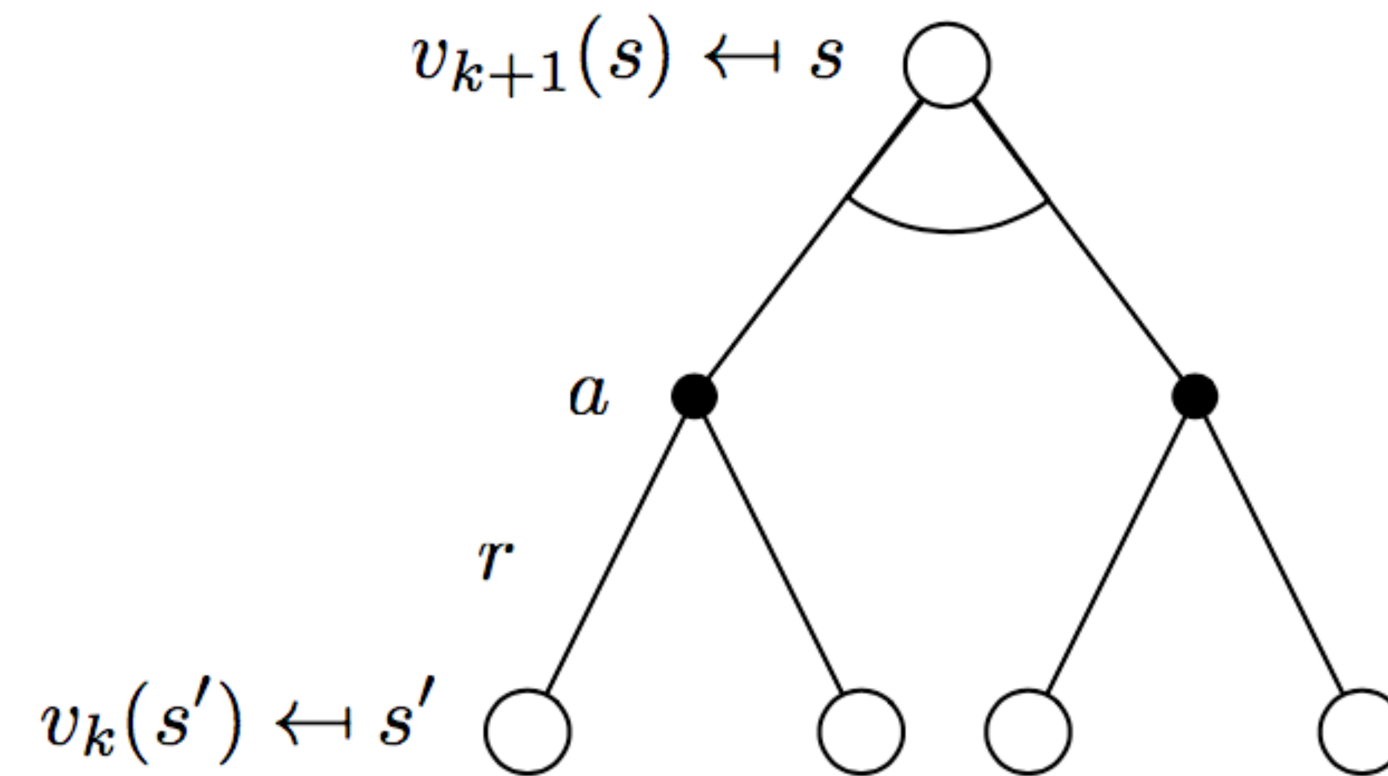
$$v_\pi = (I - \gamma \mathcal{P}^\pi)^{-1} \mathcal{R}^\pi$$

Bellman Optimality Eq, V



$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s')$$

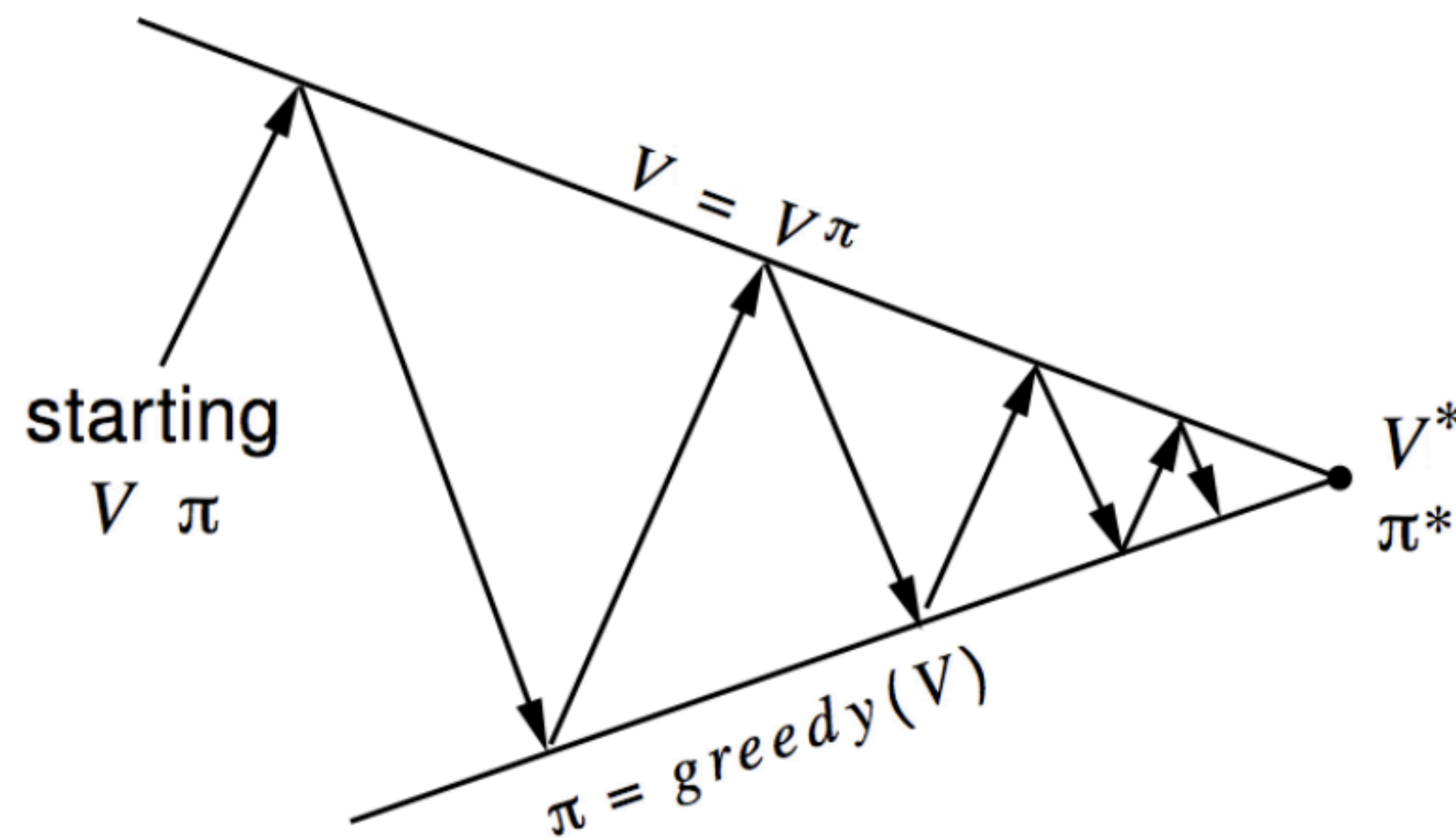
Value Iteration

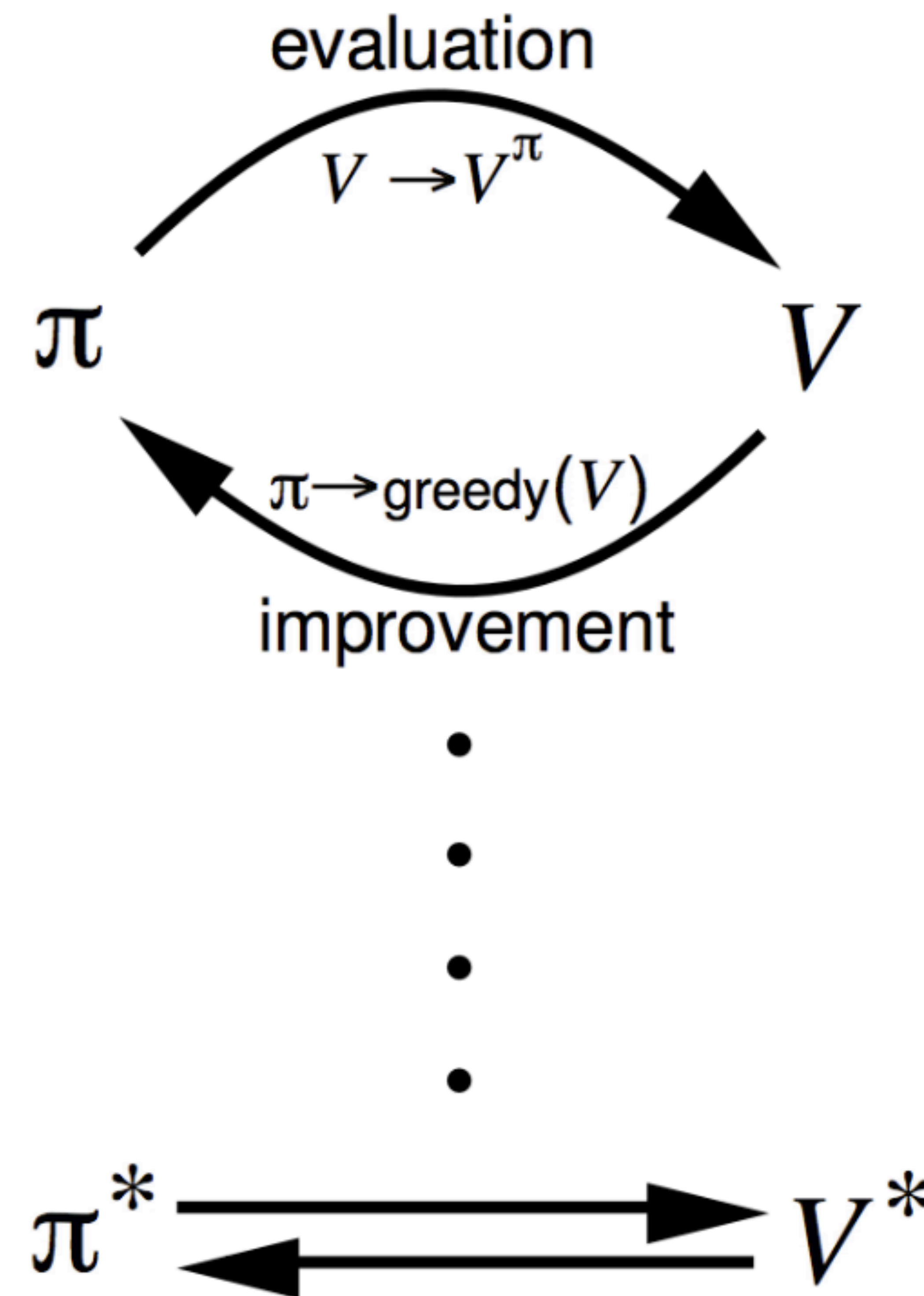


$$v_{k+1}(s) = \max_{a \in \mathcal{A}} \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}_{k+1} = \max_{a \in \mathcal{A}} \boldsymbol{\mathcal{R}}^{\mathbf{a}} + \gamma \boldsymbol{\mathcal{P}}^{\mathbf{a}} \mathbf{v}_k$$

Policy Iteration





Policy evaluation Estimate v_π
Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$
Greedy policy improvement

MC and TD

- Goal: learn v_π online from experience under policy π
- Incremental every-visit Monte-Carlo
  - Update value $V(S_t)$ toward *actual* return G_t

$$V(S_t) \leftarrow V(S_t) + \alpha\left(G_t - V(S_t)\right)$$

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward *estimated* return $R_{t+1} + \gamma V(S_{t+1})$

$$V(S_t) \leftarrow V(S_t) + \alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$$

  - $R_{t+1} + \gamma V(S_{t+1})$ is called the *TD target*
  - $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called the *TD error*