UCI University of California, Irvine

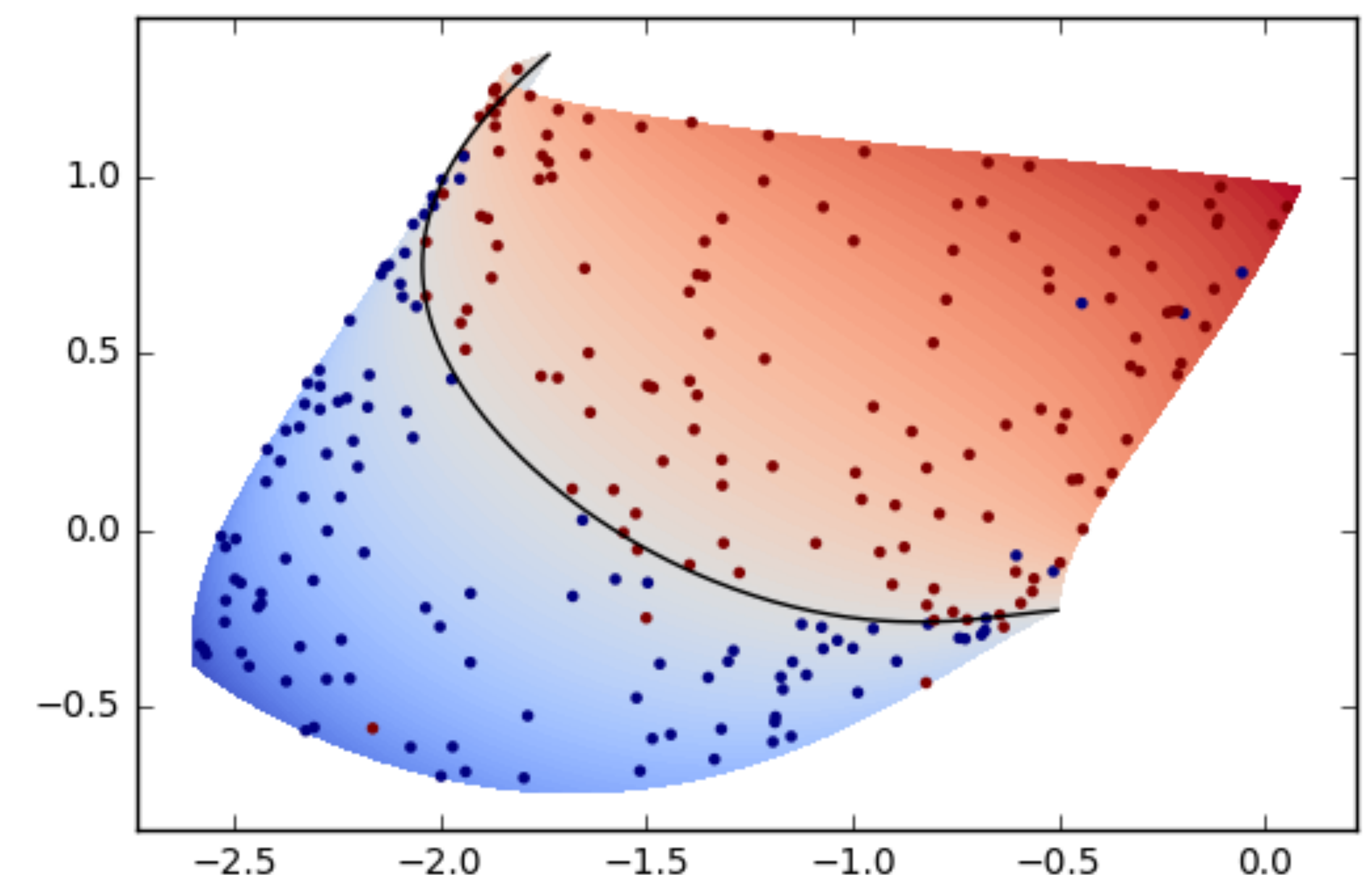
# CS 273A: Machine Learning

## Fall 2021

# Lecture 18: 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



# Logistics

project

- Final report due next Thursday, Dec 9

evaluations

- Course evaluations due end-of-week (before Monday)

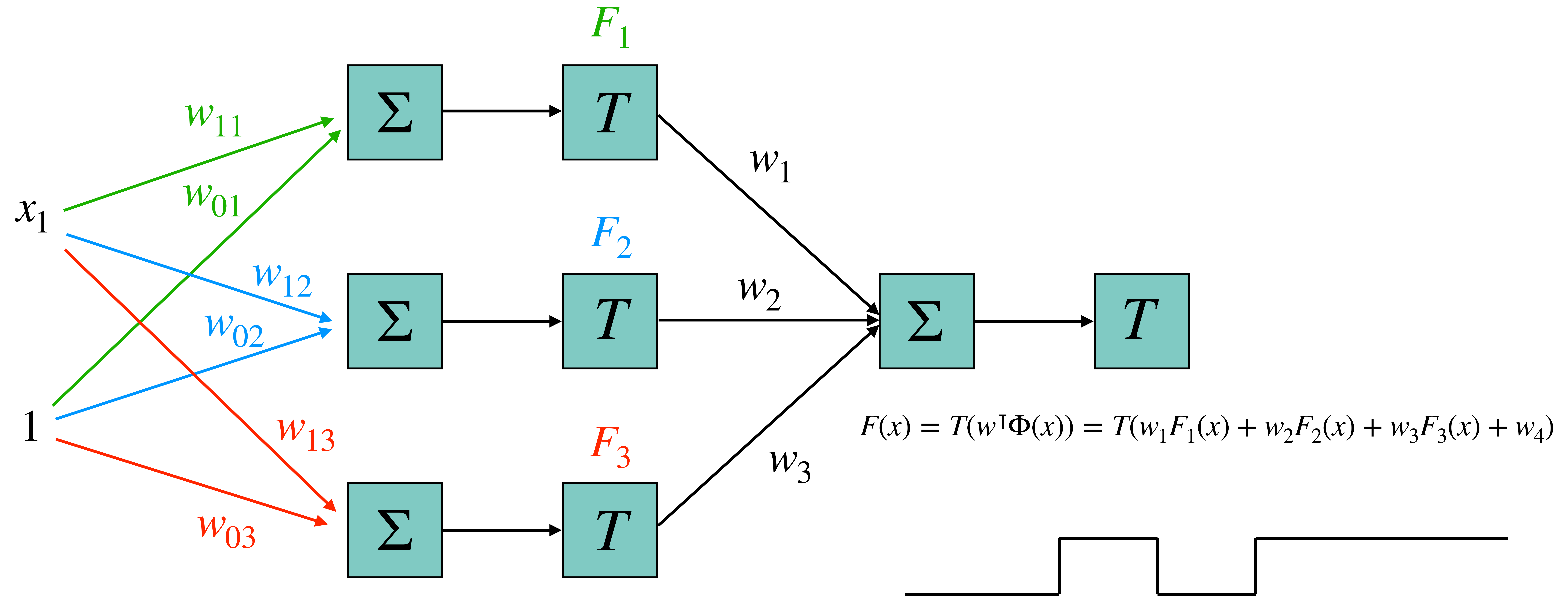
# Exam Logistics

- Format:
  - Time: Tuesday, December 7, 10:30am–12:30
  - Location: ICS 174 (in person)
  - Should be doable in 90–100 minutes
- 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

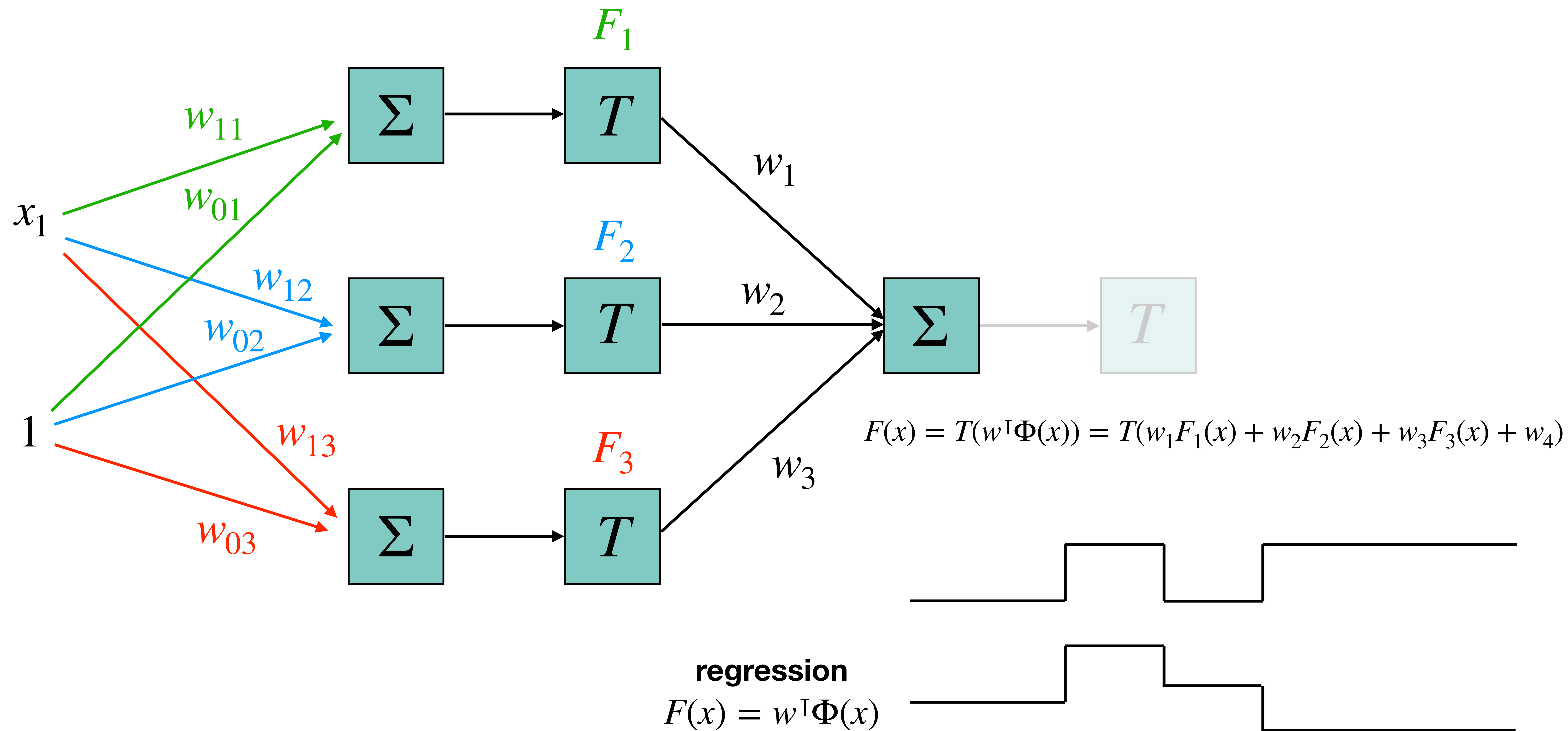
# Exam suggestions

- 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 Ed)
- 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
  - Turn in your calculation sheet(s)
    - They won't be graded, but can be used for regrading

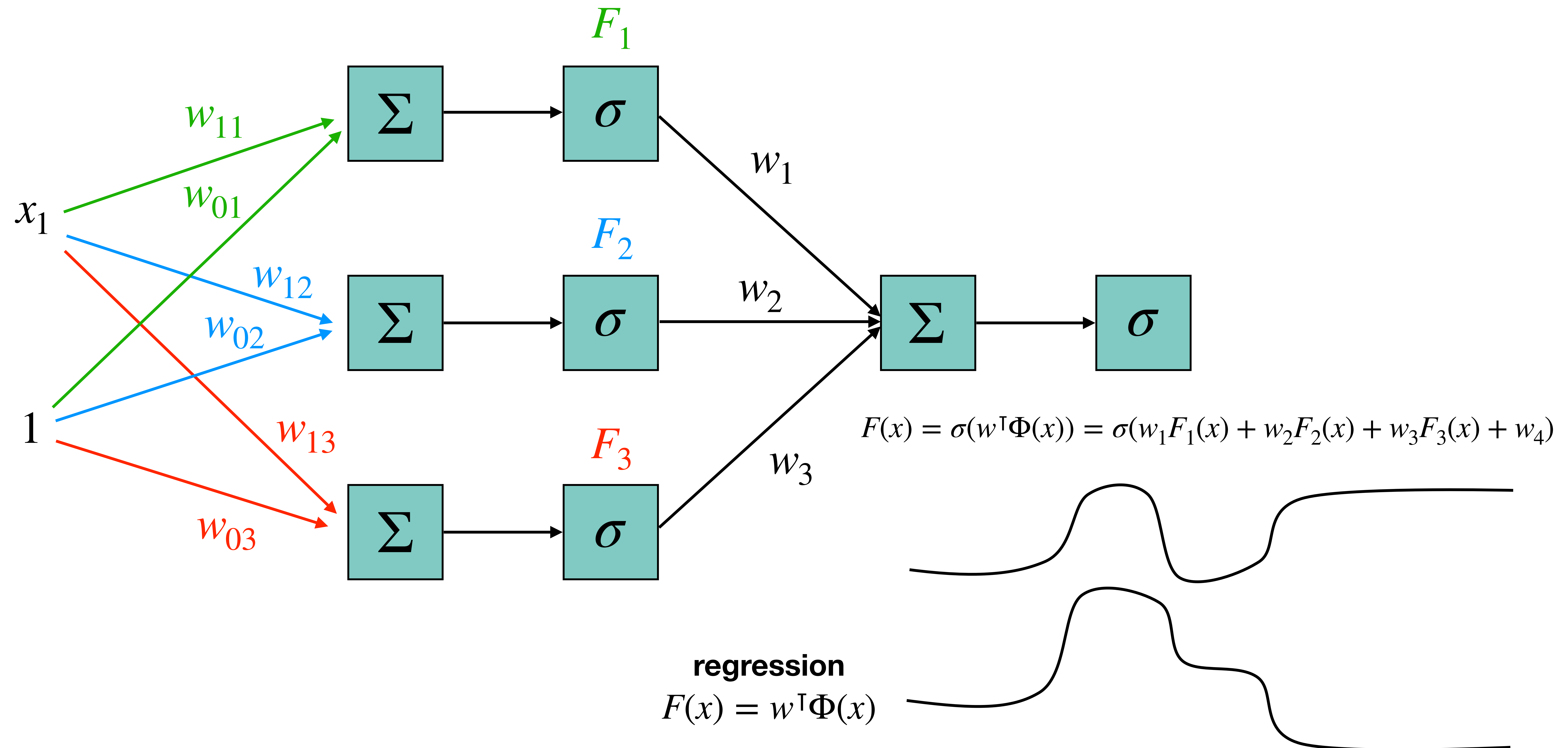
# Multi-Layer Perceptron (MLP)



# Multi-Layer Perceptron (MLP)

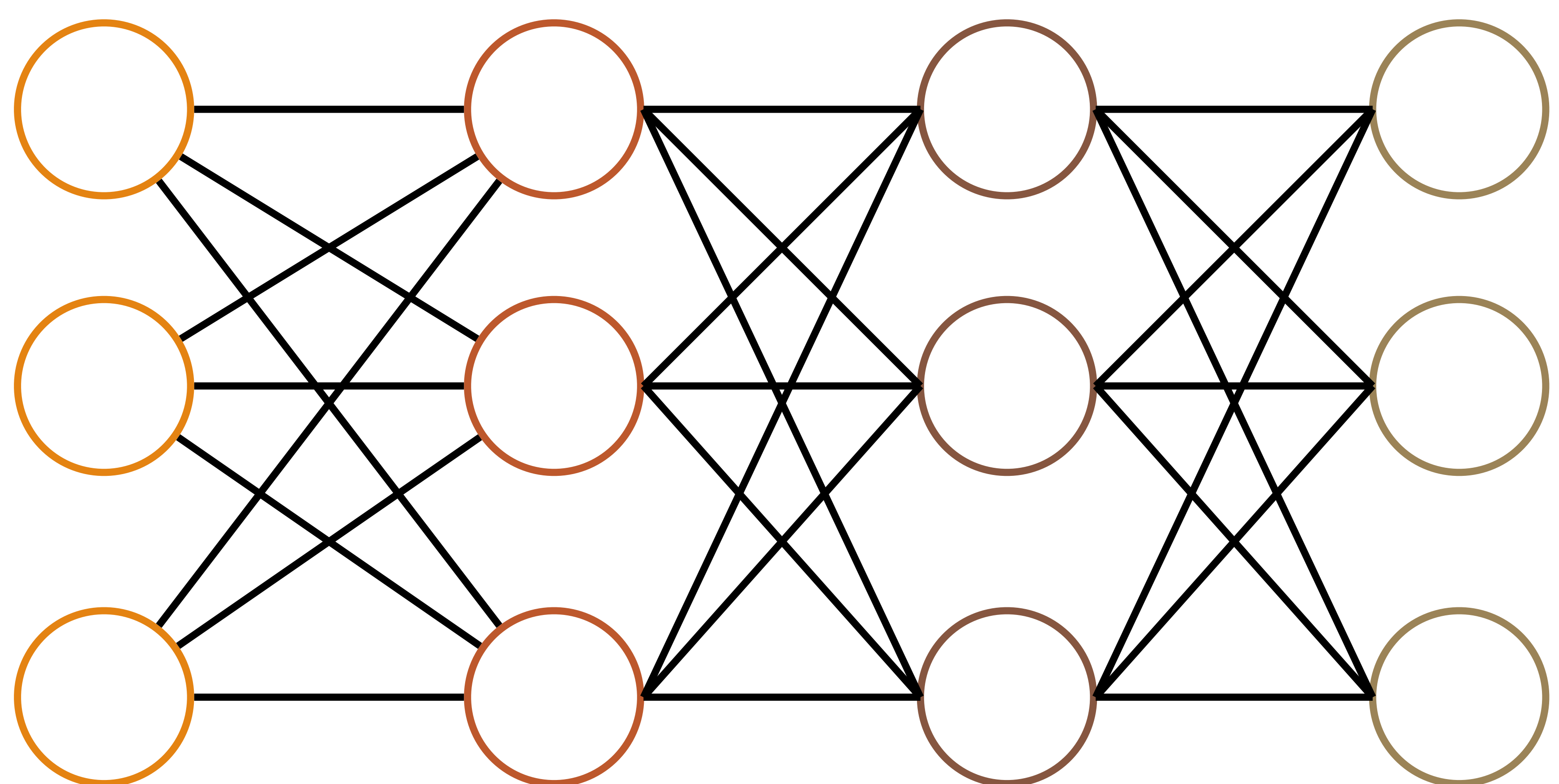


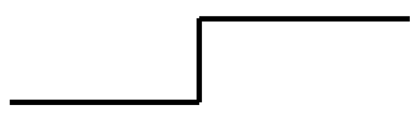
# Multi-Layer Perceptron (MLP)

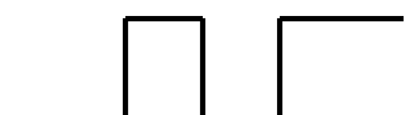


# Deep Neural Networks (DNNs)

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





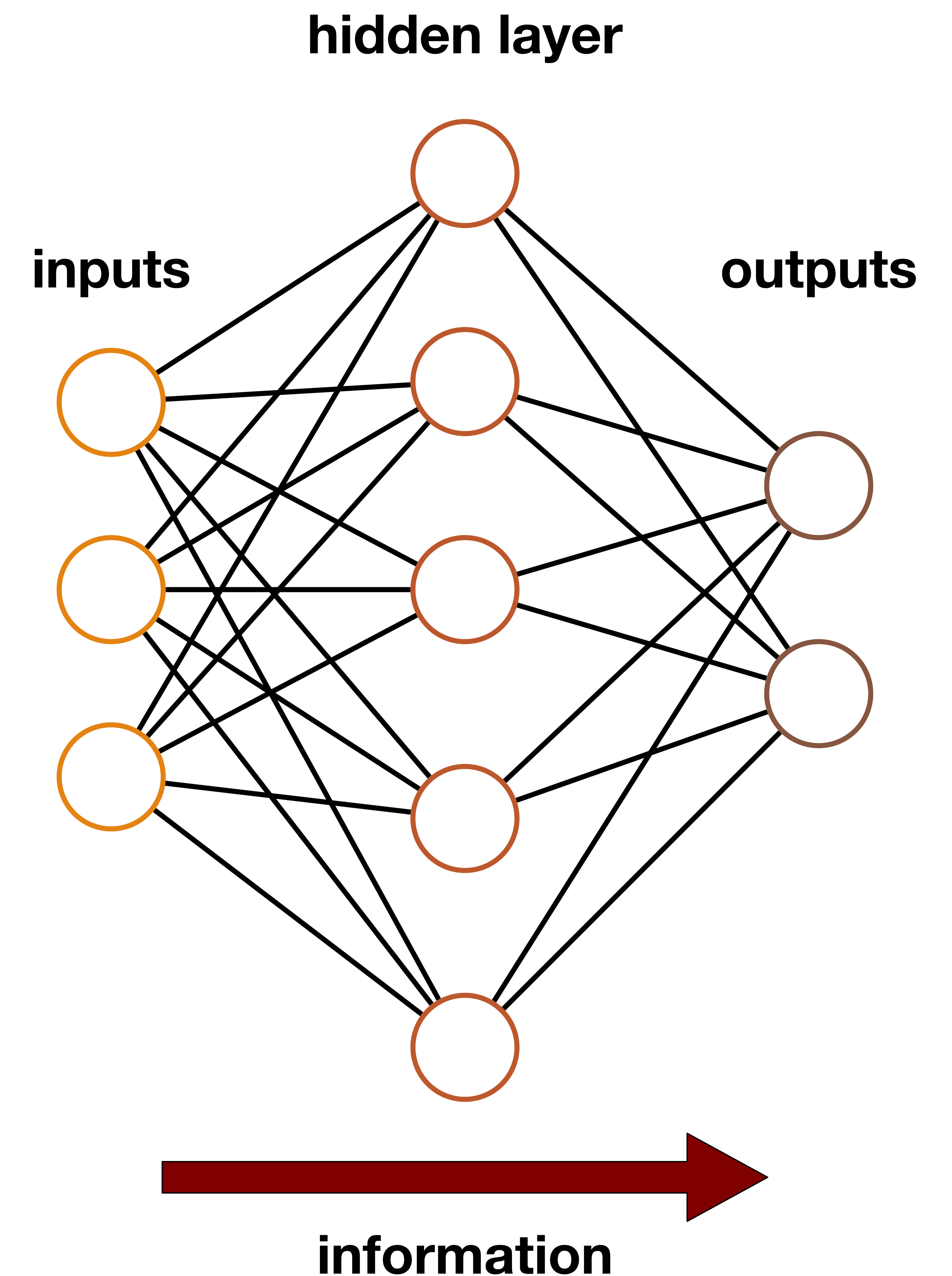
```
r1 = w[0].T @ x + b[0]    # linear response
h1 = sig(r1)              # activation function

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

                          # ...
```

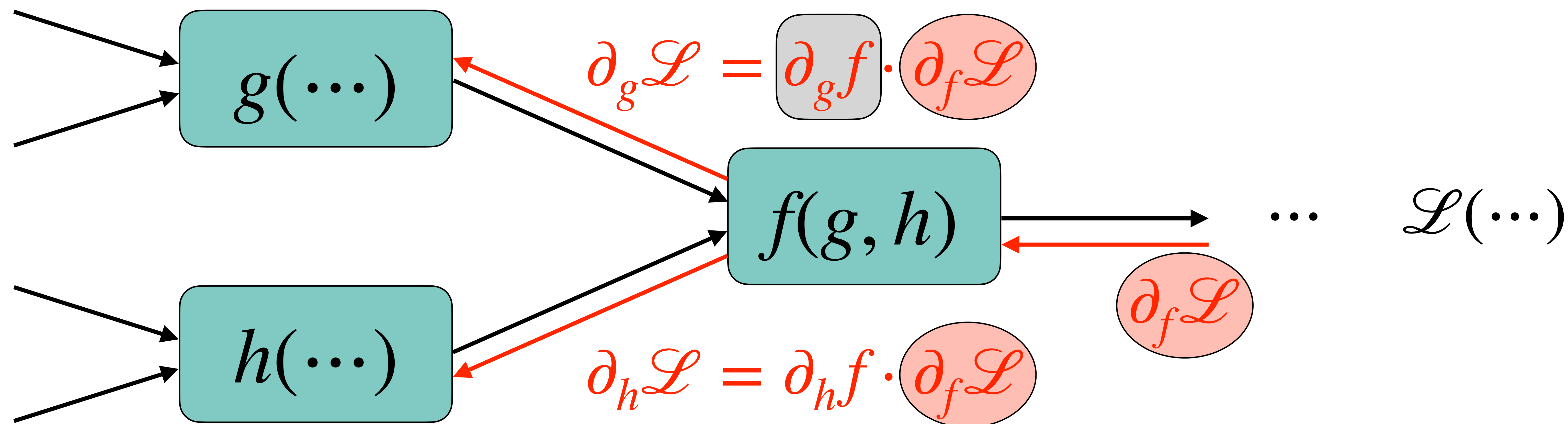
# Feed-forward (FF) networks

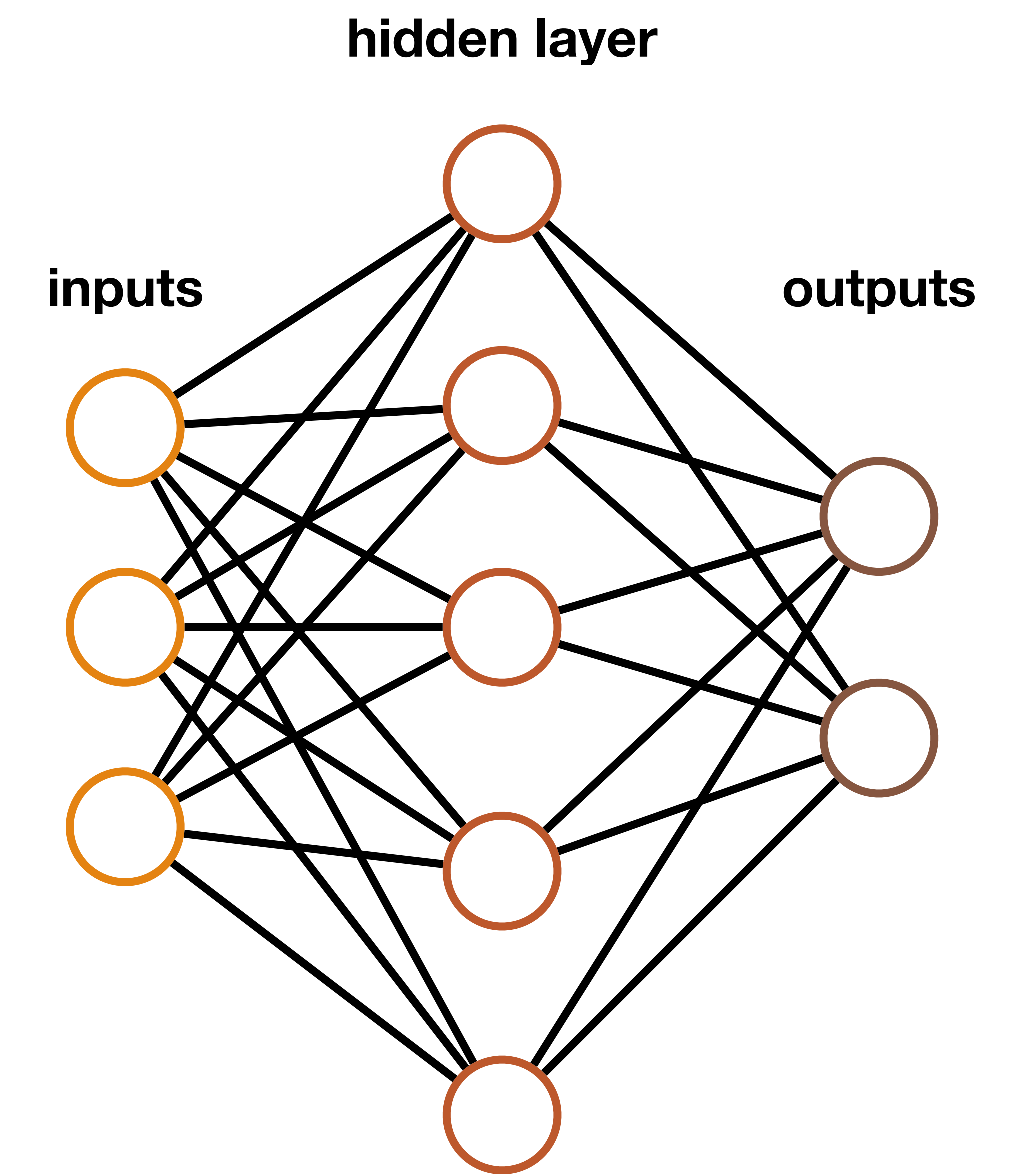
- Information flow in feed-forward (FF) networks:
  - Inputs → shallow layers → deeper layers → outputs
  - Alternative: recurrent NNs (information loops back)
- Multiple outputs $\implies$ efficiency:
  - Shared parameters, less data, less computation
- Multi-class classification:
  - One-hot labels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



# Gradient computation

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



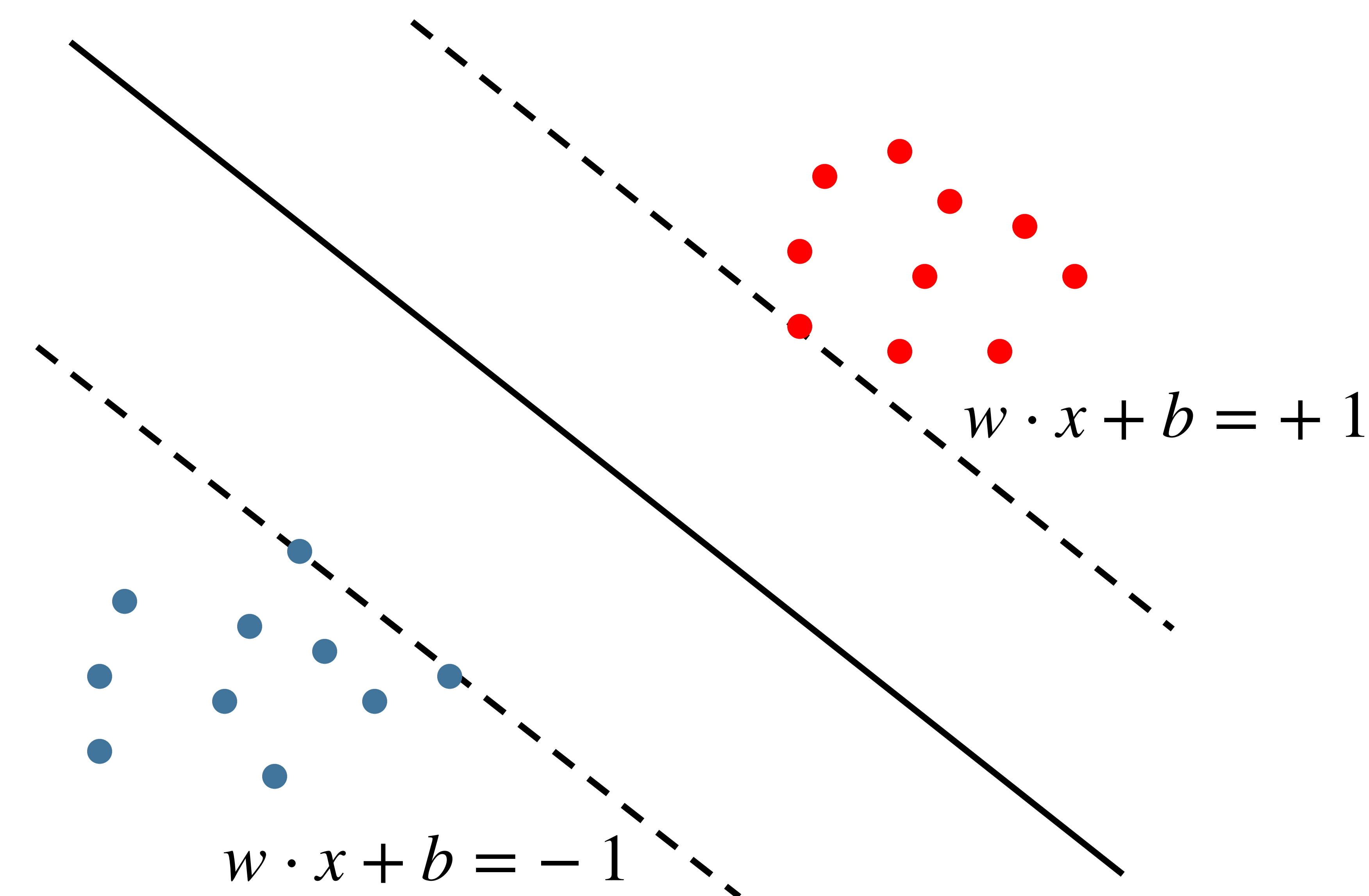

**example:** $f(g,h) = \sigma(g+h) \implies \partial_g f = f(1-f)$

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

- Backpropagation = chain rule + dynamic programming to avoid repetitions

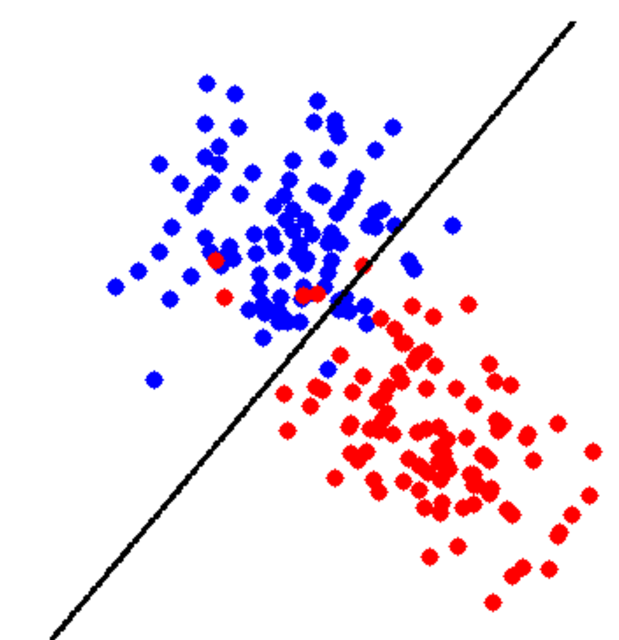
# Maximizing the margin

- 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: $\max_{0 \le \lambda \le R} \sum_j \left( \lambda_j - \frac{1}{2} \sum_k \lambda_j \lambda_k y^{(j)} y^{(k)} K(x^{(j)}, x^{(k)}) \right)$ s.t. $\sum_j \lambda_j y^{(j)} = 0$
- Learned parameters = $\lambda$ ($m$ parameters)
  - But also need to store all support vectors (having $\lambda_j > 0$)
- Prediction: $\hat{y}(x) = \operatorname{sign}(w \cdot \Phi(x))$

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

# Bagging

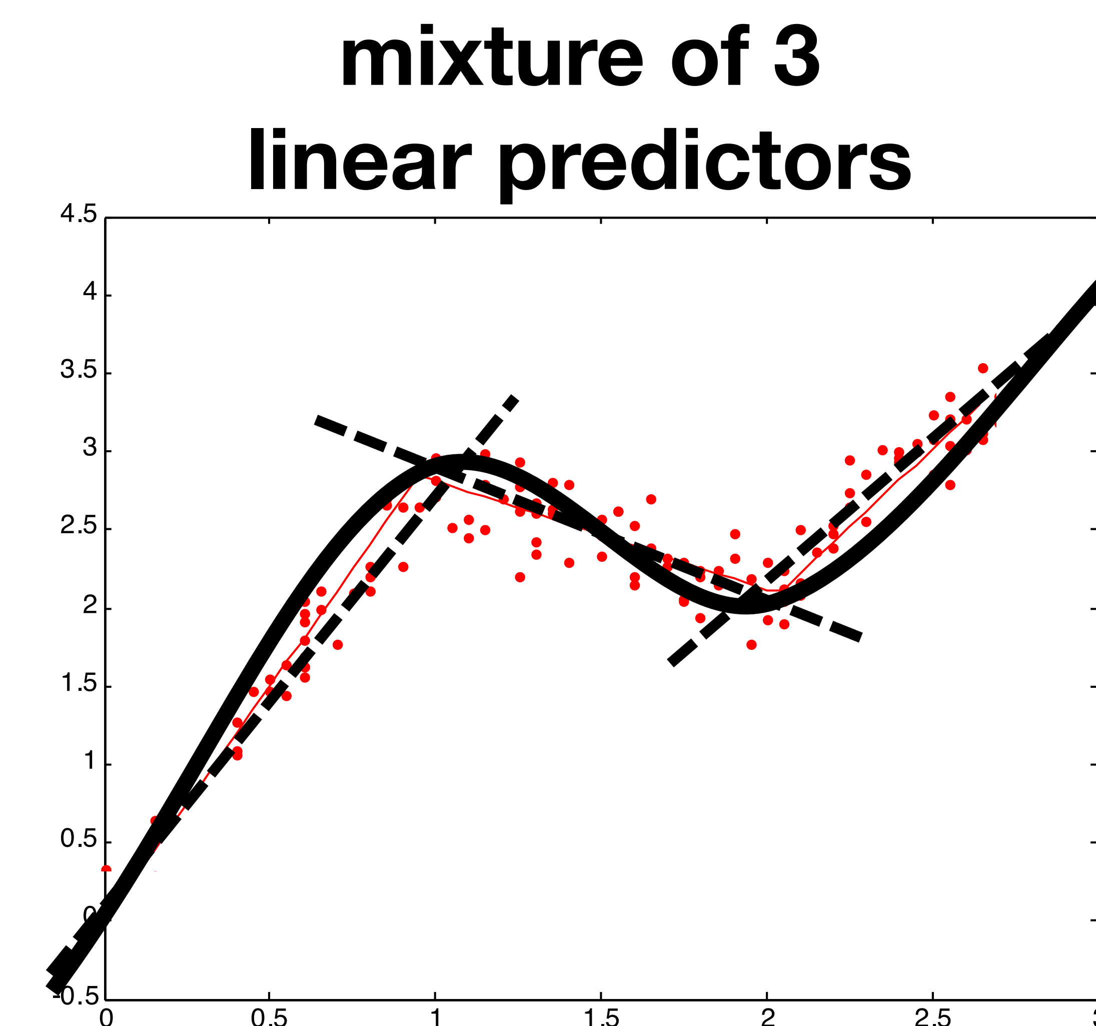
- Bagging = bootstrap aggregating:
  - Resample $K$ datasets $\mathscr{D}_1, \ldots, \mathscr{D}_K$ of size $b$
  - Train $K$ models $\theta_1, \ldots, \theta_K$ on each dataset
  - Regression: output $f_\theta : x \mapsto \frac{1}{K} \sum_k f_{\theta_k}(x)$
  - Classification: output $f_\theta : x \mapsto \text{majority}\{f_{\theta_k}(x)\}$
- 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): $\hat{y}(x) = f_\theta(\hat{y}_1(x), \ldots, \hat{y}_K(x))$
  - $f_\theta$ trained on held out data = validation of which model should be trusted
  - $f_\theta$ linear $\Longrightarrow$ weighted committee, with learned weights

# Mixture of Experts (MoE)

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



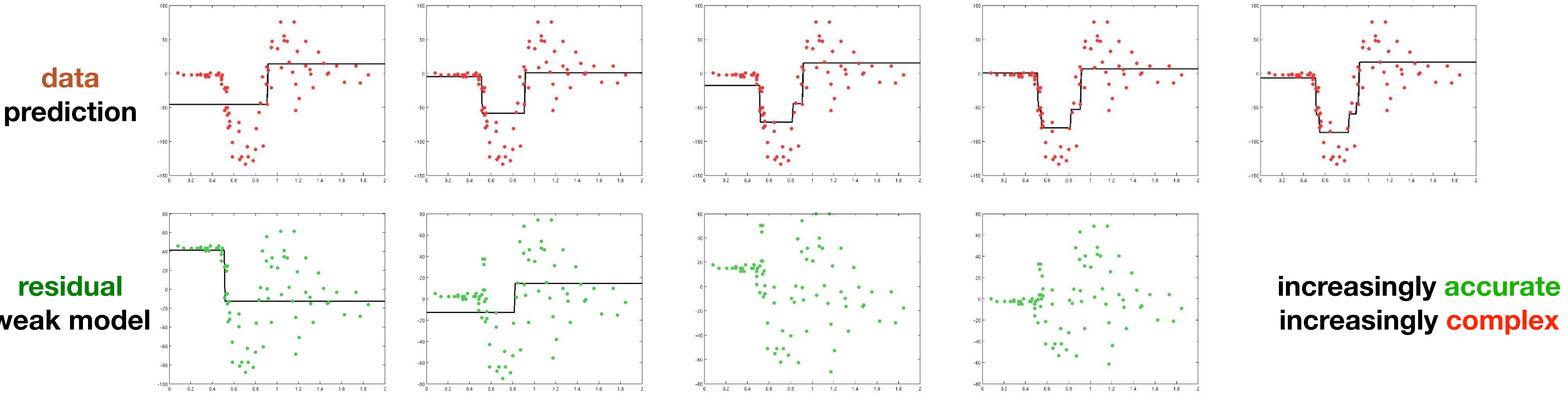

- Can we predict which model will perform well?
  - Learn a predictor $\alpha_\phi(k \mid x)$
    - E.g., multilogistic regression (softmax) $\alpha_\phi(k \mid x) = \frac{\exp(\phi_k \cdot x)}{\sum_{k'} \exp(\phi_{k'} \cdot x)}$
- Loss, experts, weights differentiable $\Longrightarrow$ end-to-end gradient-based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

# Random Forests

- Bagging over decision trees: which feature at root?
  - Much data $\Longrightarrow$ max info gain stable across data samples
  - Little diversity among models $\Longrightarrow$ 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 $\Longrightarrow$ 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 $\hat{y}_k = \hat{y}_{k-1} + f_k(x)$

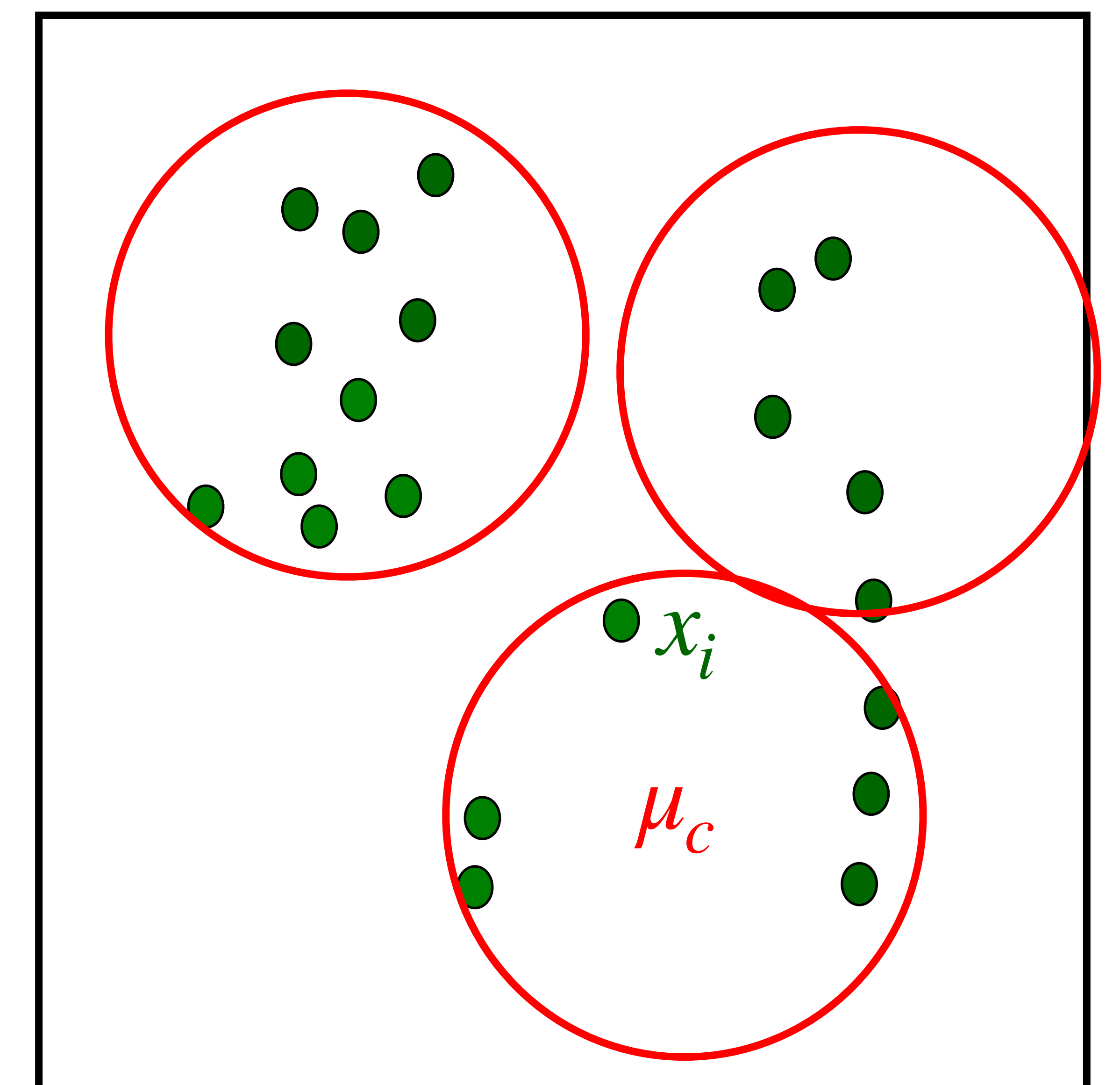


# AdaBoost

- 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 $\hat{y}(x) = \operatorname{sign} \sum_k \alpha_k f_k(x)$

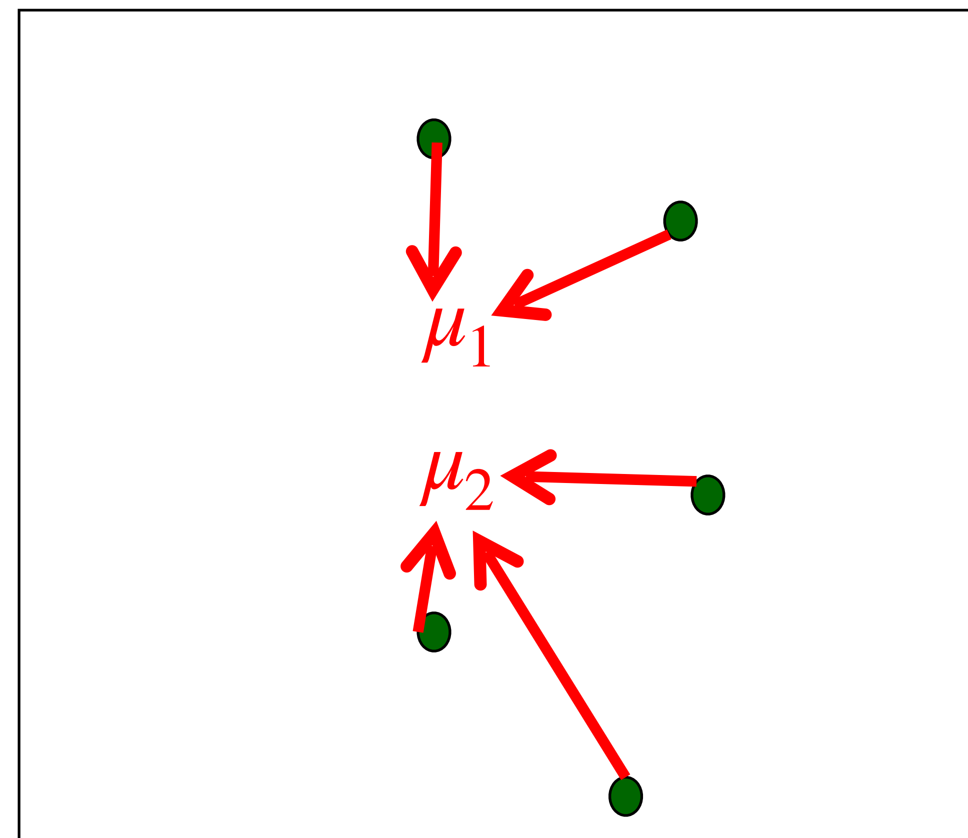
# $k$-Means

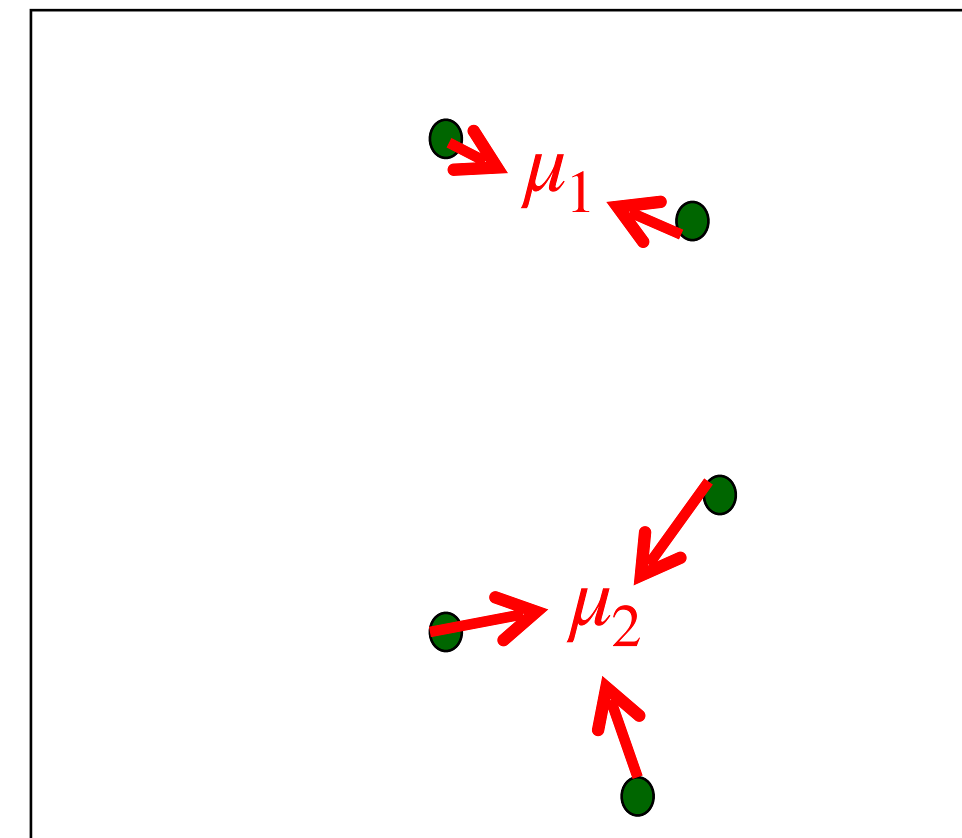
- Simple clustering algorithm
- Repeat:
  - Update the clustering = assignment of data points to clusters
  - Update the cluster's representation to match the assigned points
- Notation:
  - $x_i$ = data point in the dataset
  - $k$ = number of clusters
  - $\mu_c$ = representation of cluster $c$



# $k$-Means

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



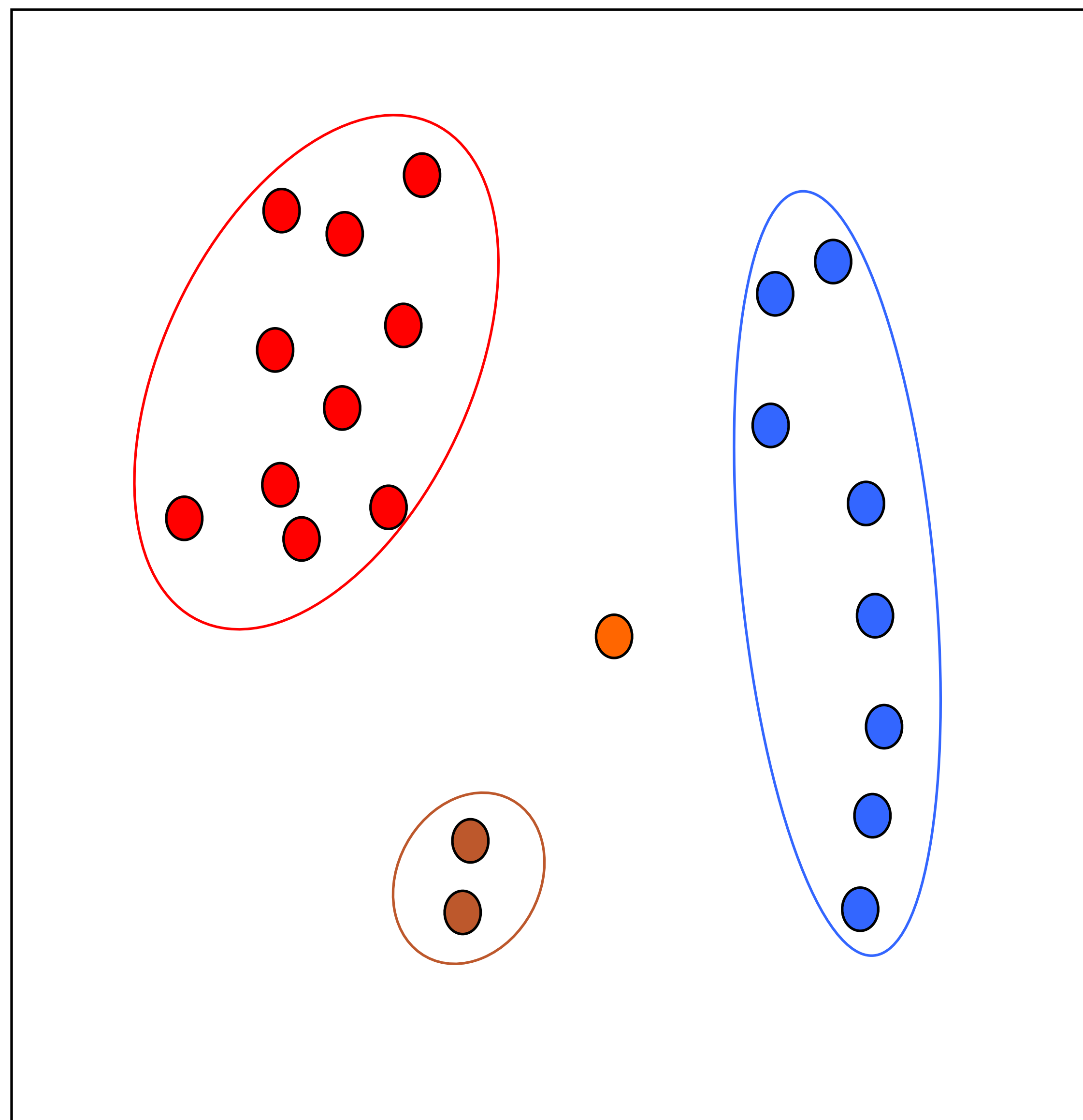
# Hierarchical agglomerative clustering

- 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 $\implies O(m^2 \log m)$

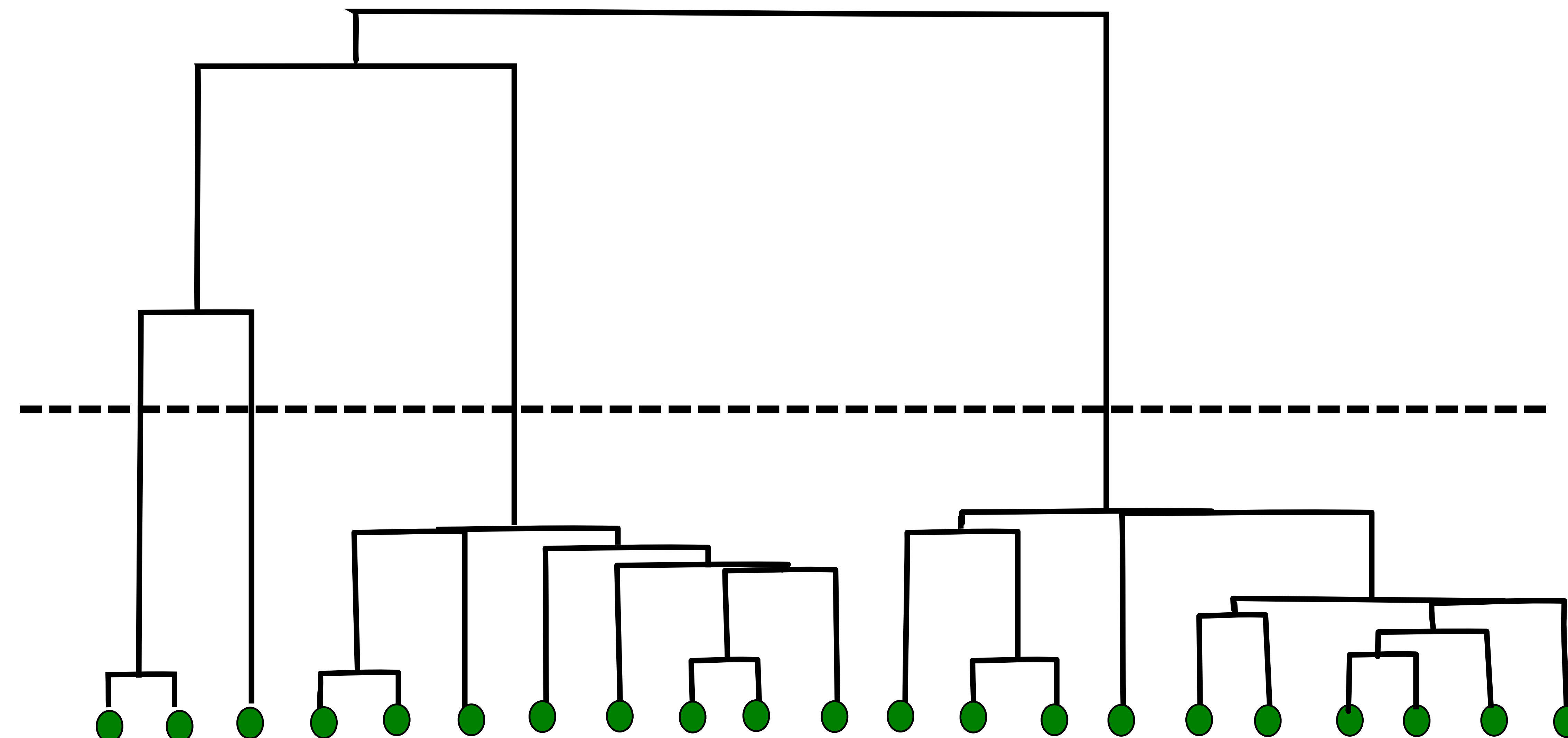
# From dendrogram to clusters

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

**data**



**dendrogram**

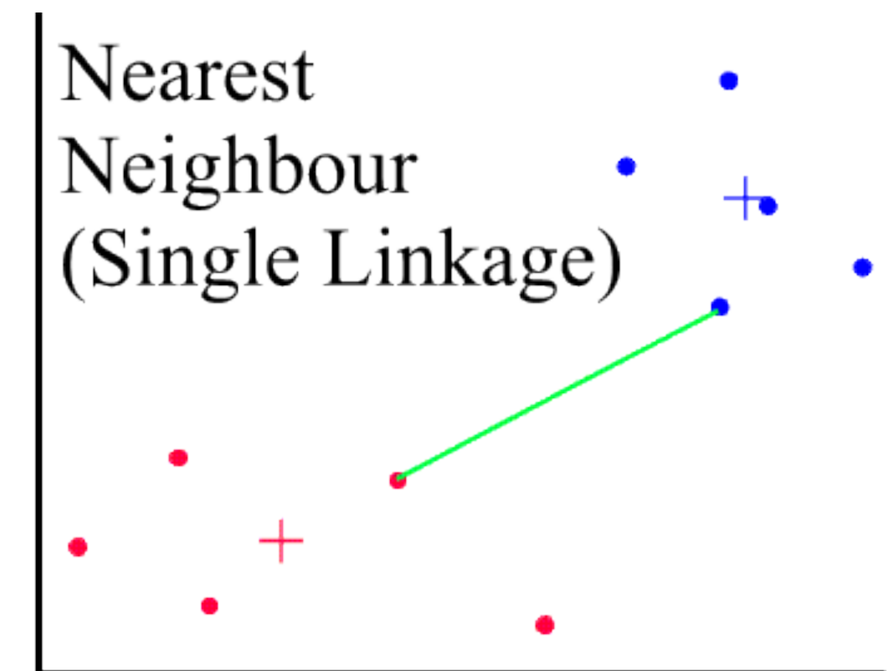


  - For a given $k$, or a given level of dissimilarity

# Distance measures

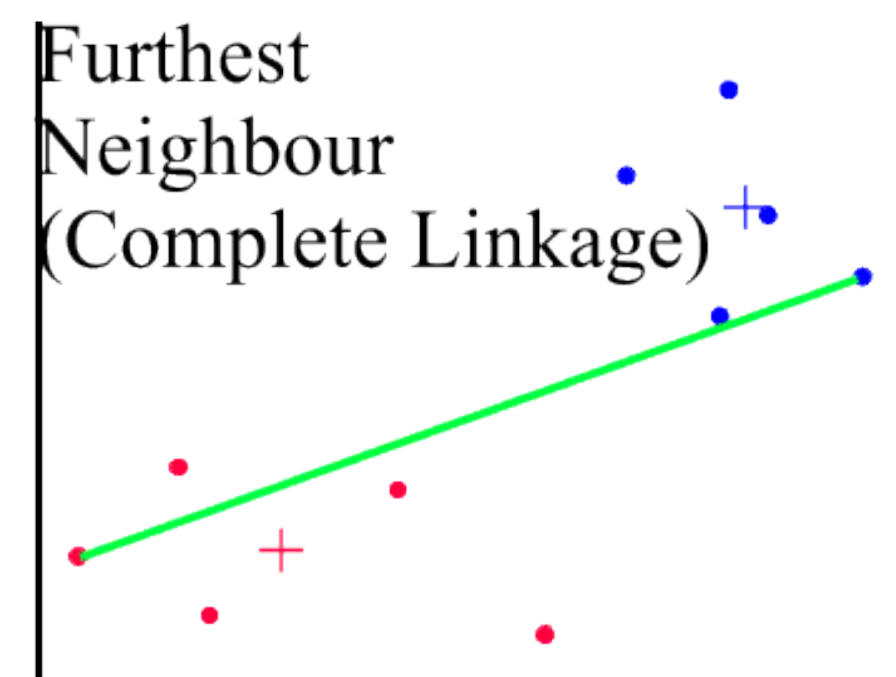
- $d_{\min}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|^2$

produces minimum spanning tree



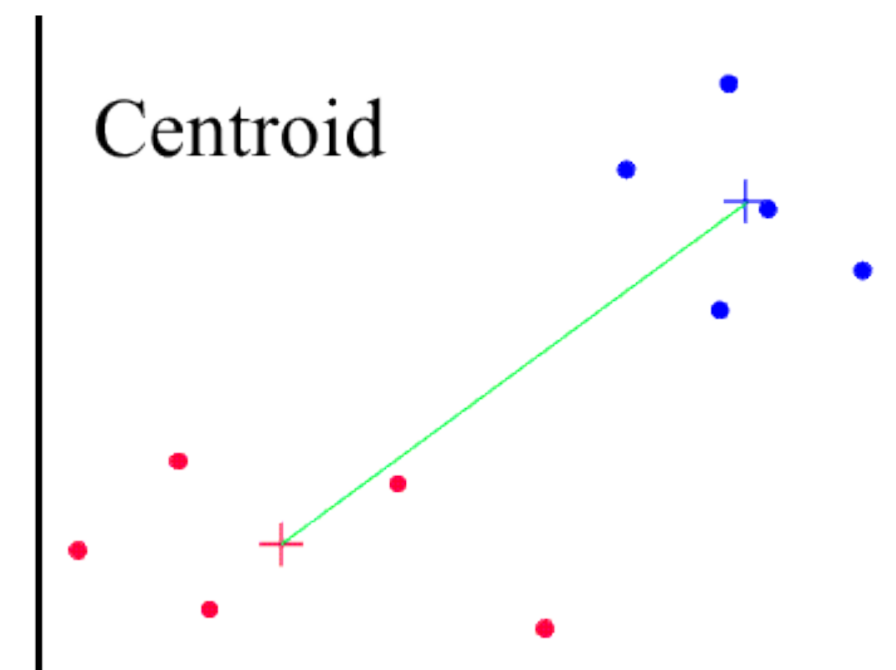

- $d_{\max}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|^2$

avoids elongated clusters




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

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



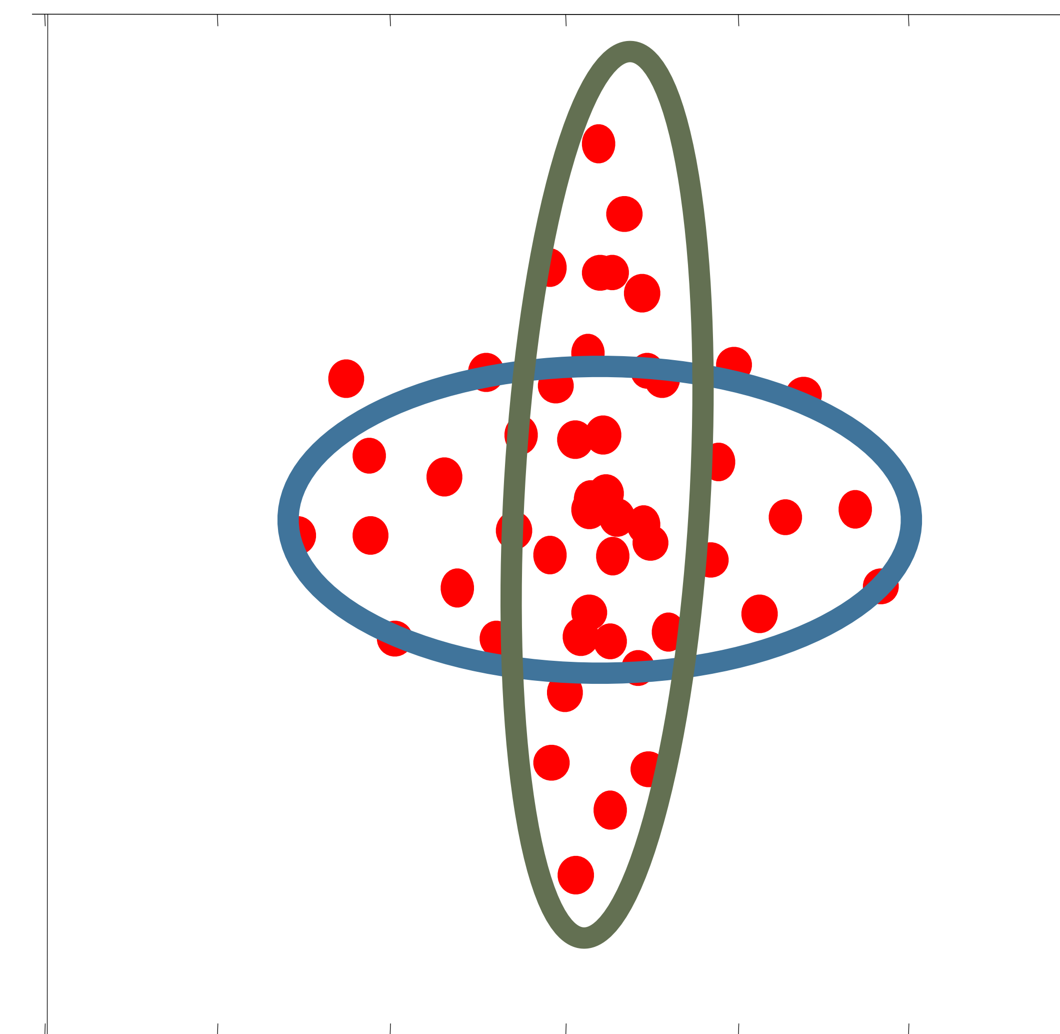

- Important property: iterative computation

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

# Gaussian Mixture Models (GMMs)

- Each cluster is modeled by a Gaussian $p(x \mid c) = \mathcal{N}(x; \mu_c, \Sigma_c)$
  - $\Sigma_c$ allows non-isotropic clusters $\implies$ 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**
**$\implies$ can be any of them**



# Training GMMs

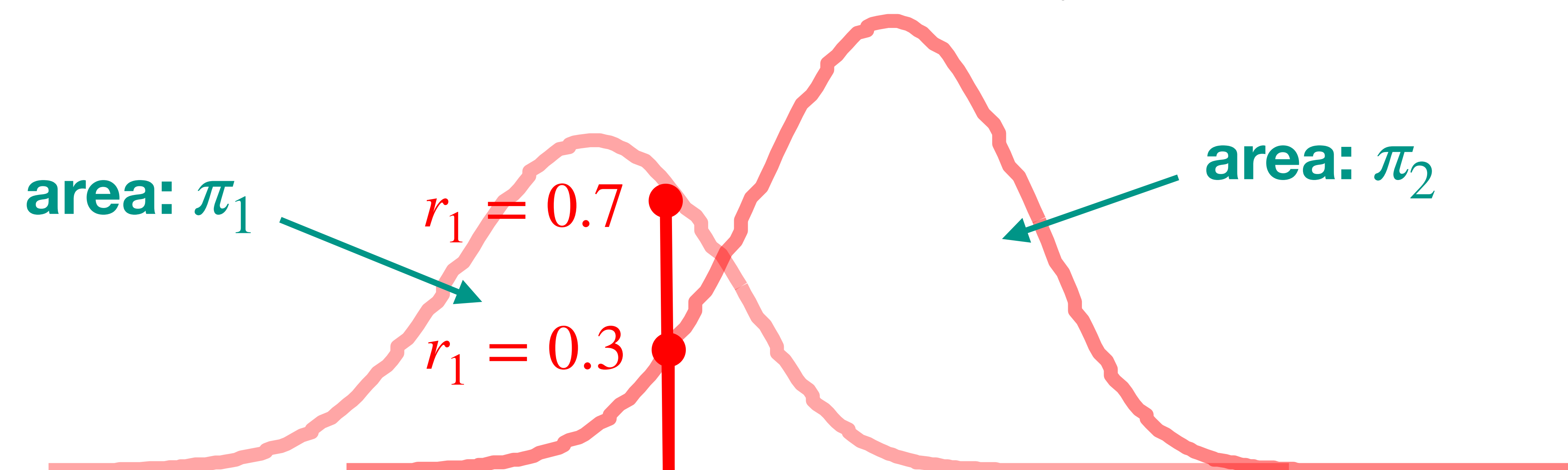
- Compare to $k$-Means:
  - Assign data points to clusters $z_i$
  - Update each cluster's parameters $\mu_c$
- A "soft" version of $k$-Means: Expectation–Maximization (EM) algorithm
  - Find a "soft" assignment $p(c \mid x)$
  - Update model parameters $p(c)$, $p(x \mid c)$
- The EM algorithm is extremely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

# Expectation–Maximization: E-step

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

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

  - 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 $c$

$$\pi_c = \frac{m_c}{m}$$

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

weighted mean of data in cluster $c$

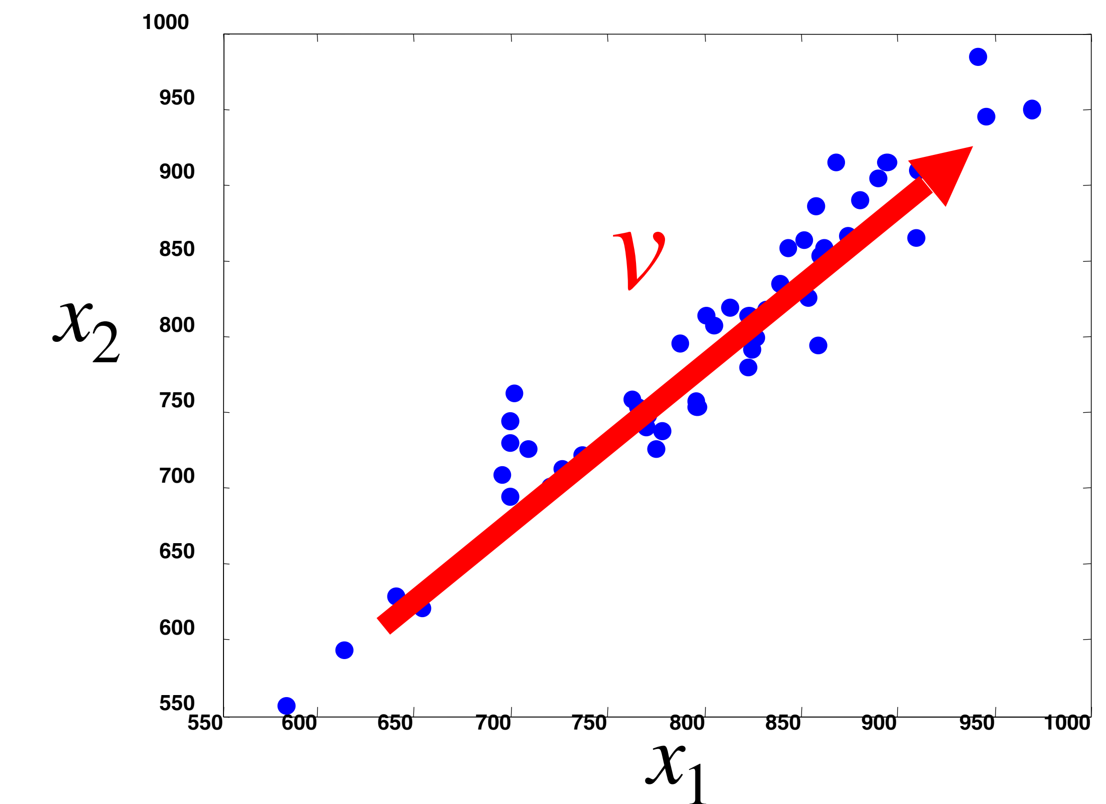
$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic}(x_i - \mu_c)(x_i - \mu_c)^\intercal$$

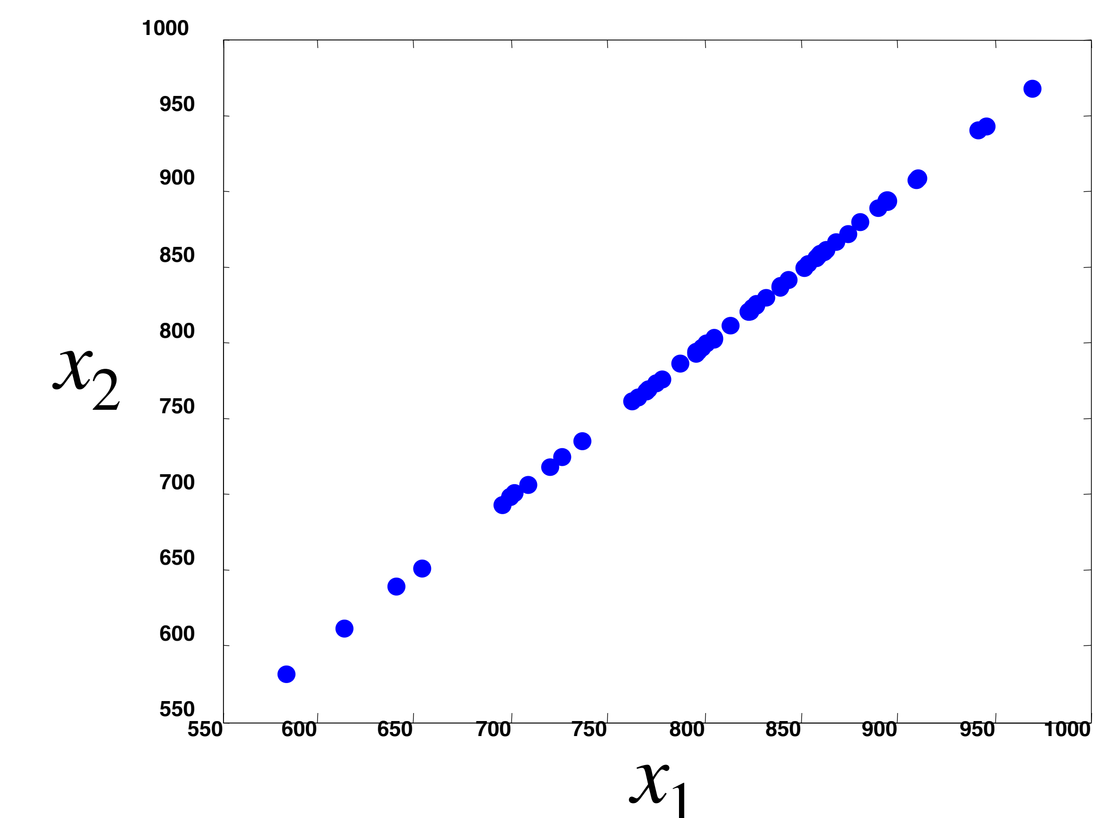
weighted covariance of data in cluster $c$

# 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 $v$
    - $z = \arg\min \|x - zv\|^2 \implies z = \frac{x^\intercal v}{v^\intercal v}$

projection of $x$ on $v$



# 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
  - $\Sigma = X^\intercal X = VD^\intercal U^\intercal UDV^\intercal = V(D^\intercal D)V^\intercal$

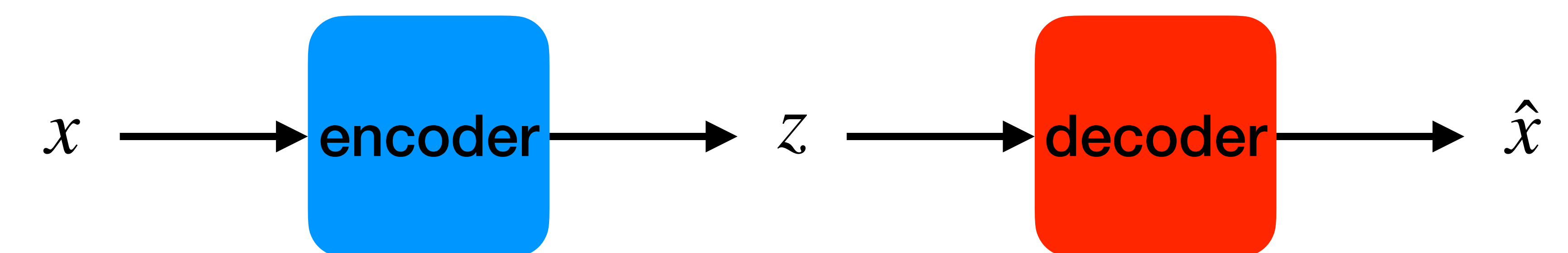
$$\underset{m \times n}{X} = \underset{m \times m}{U} \cdot \underset{m \times n}{D} \cdot \underset{n \times n}{V^\intercal}$$

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

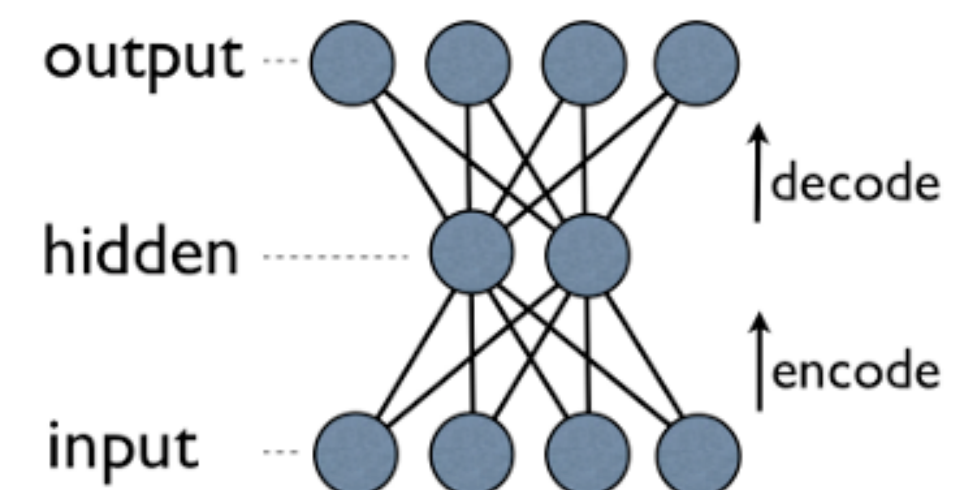
- $UD$ matrix gives coefficients to reconstruct data: $x_i = U_{i,1}D_{1,1}v_1 + U_{i,2}D_{2,2}v_2 + \cdots$
  - We can truncate this after top $k$ singular values (square root of eigenvalues)

# Nonlinear latent spaces

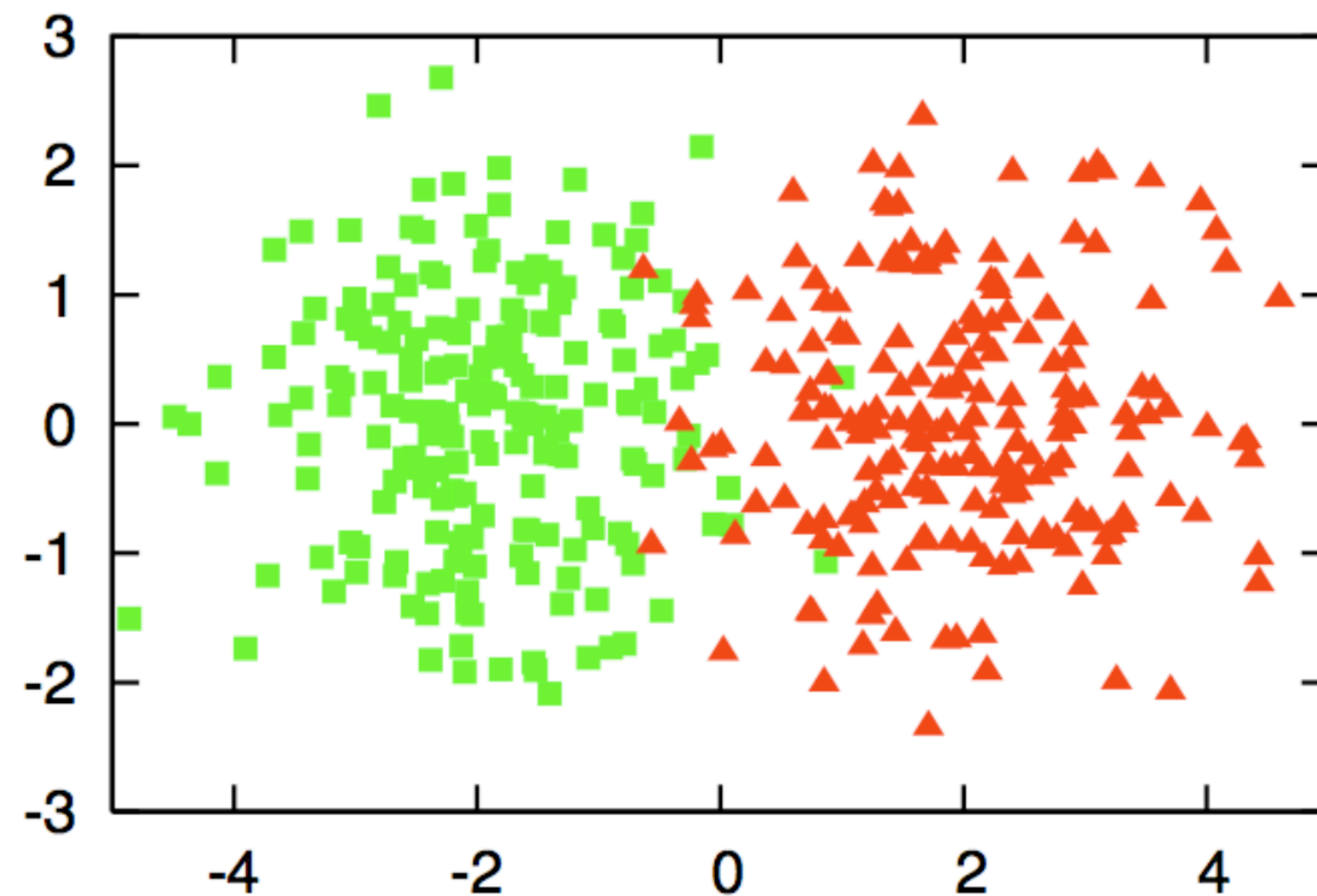
- Latent-space representation = represent $x_i$ as $z_i$
  - Usually more succinct, less noisy
  - Preserves most (interesting) information on $x_i \implies$ can reconstruct $\hat{x}_i \approx x_i$
  - Auto-encoder = encode $x \to z$, decode $z \to \hat{x}$

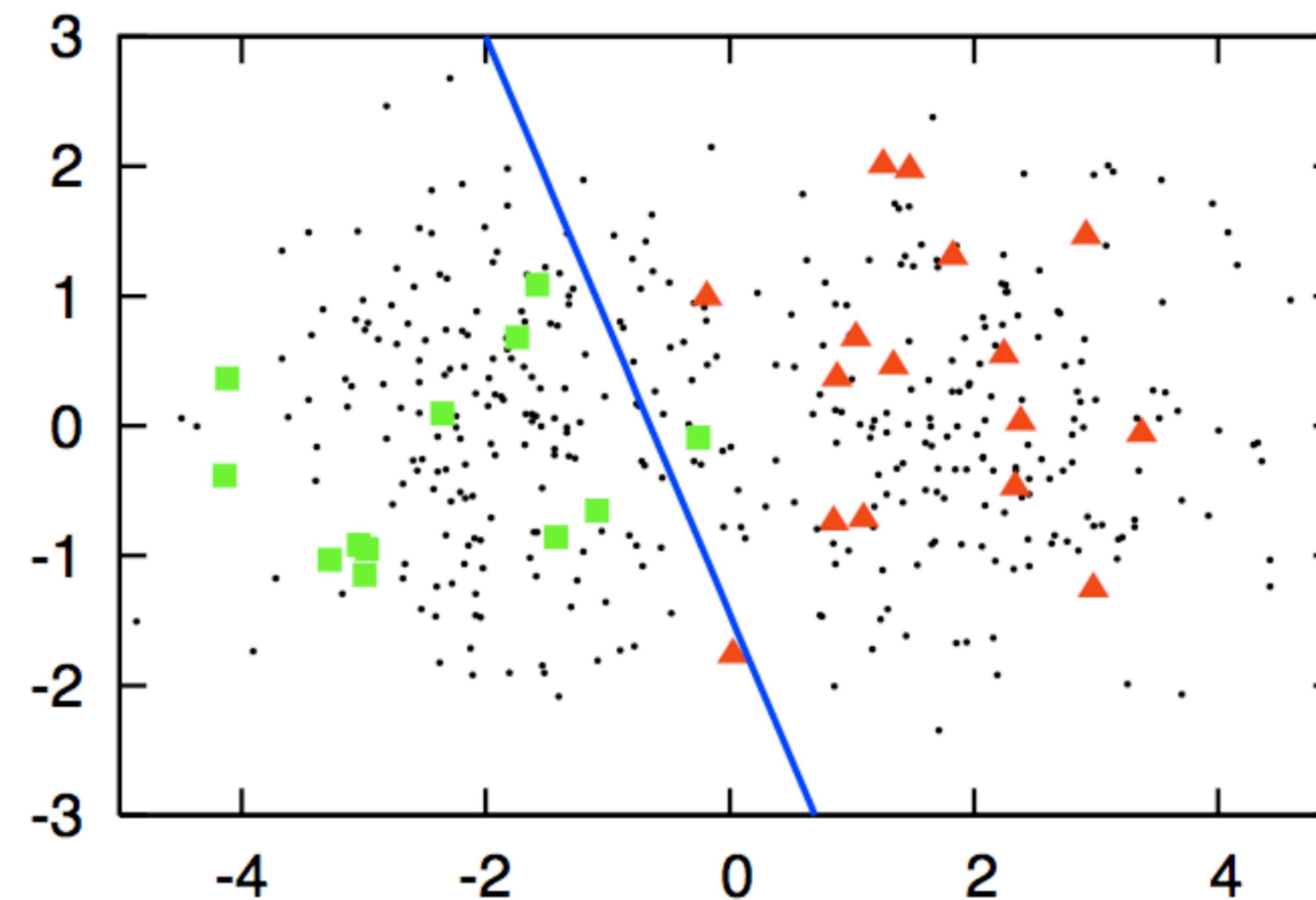


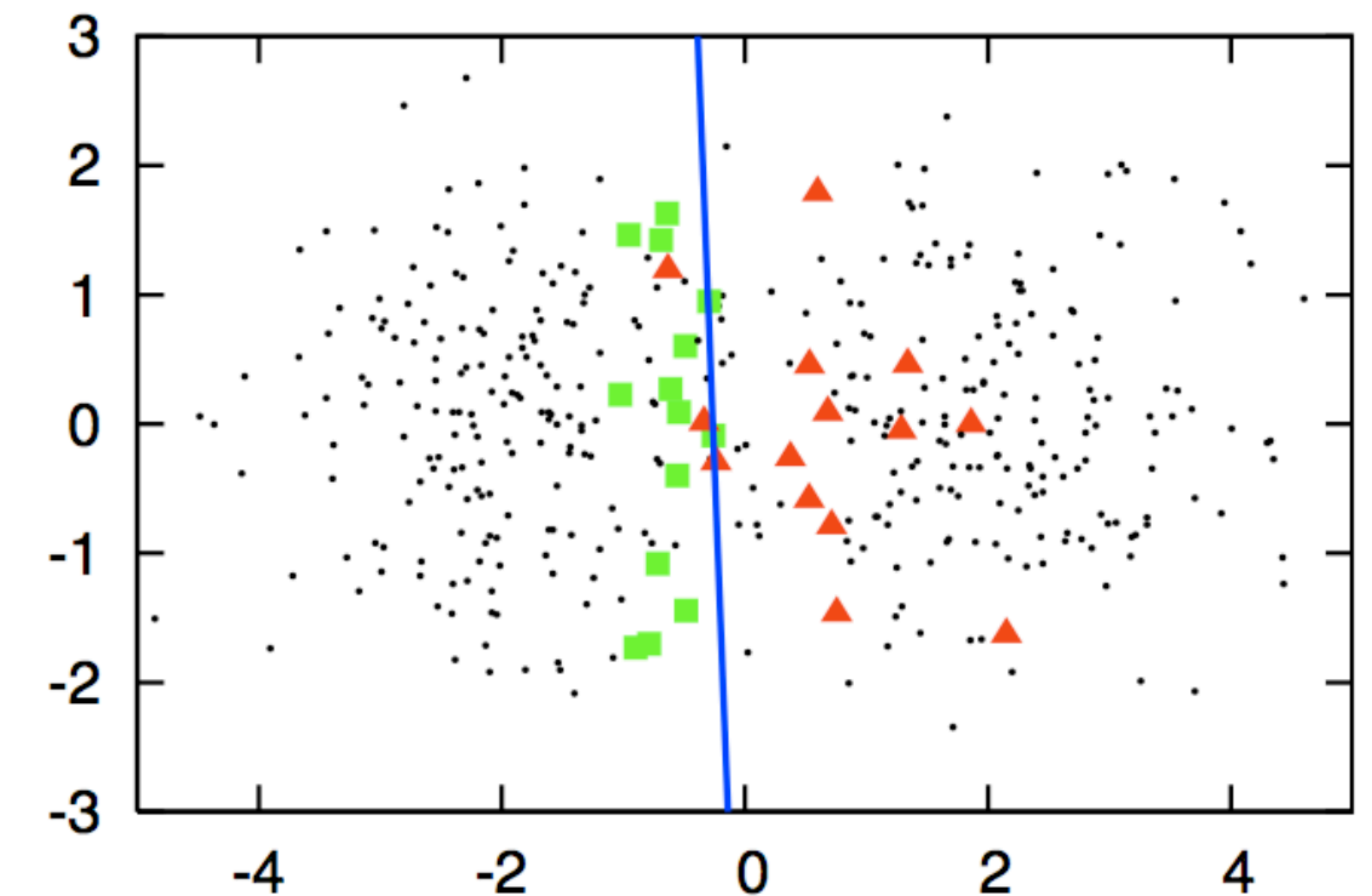

- Linear latent-space representation:
  - Encode: $Z = XV_{\leq k} = (UDV^\intercal V)_{\leq k} = U_{\leq k}D_{\leq k}$; Decode: $X \approx ZV_{\leq k}^\intercal$
- Nonlinear: e.g., encoder + decoder are neural networks
  - Restrict $z$ to be shorter than $x \implies$ requires succinctness



# Why active learning?



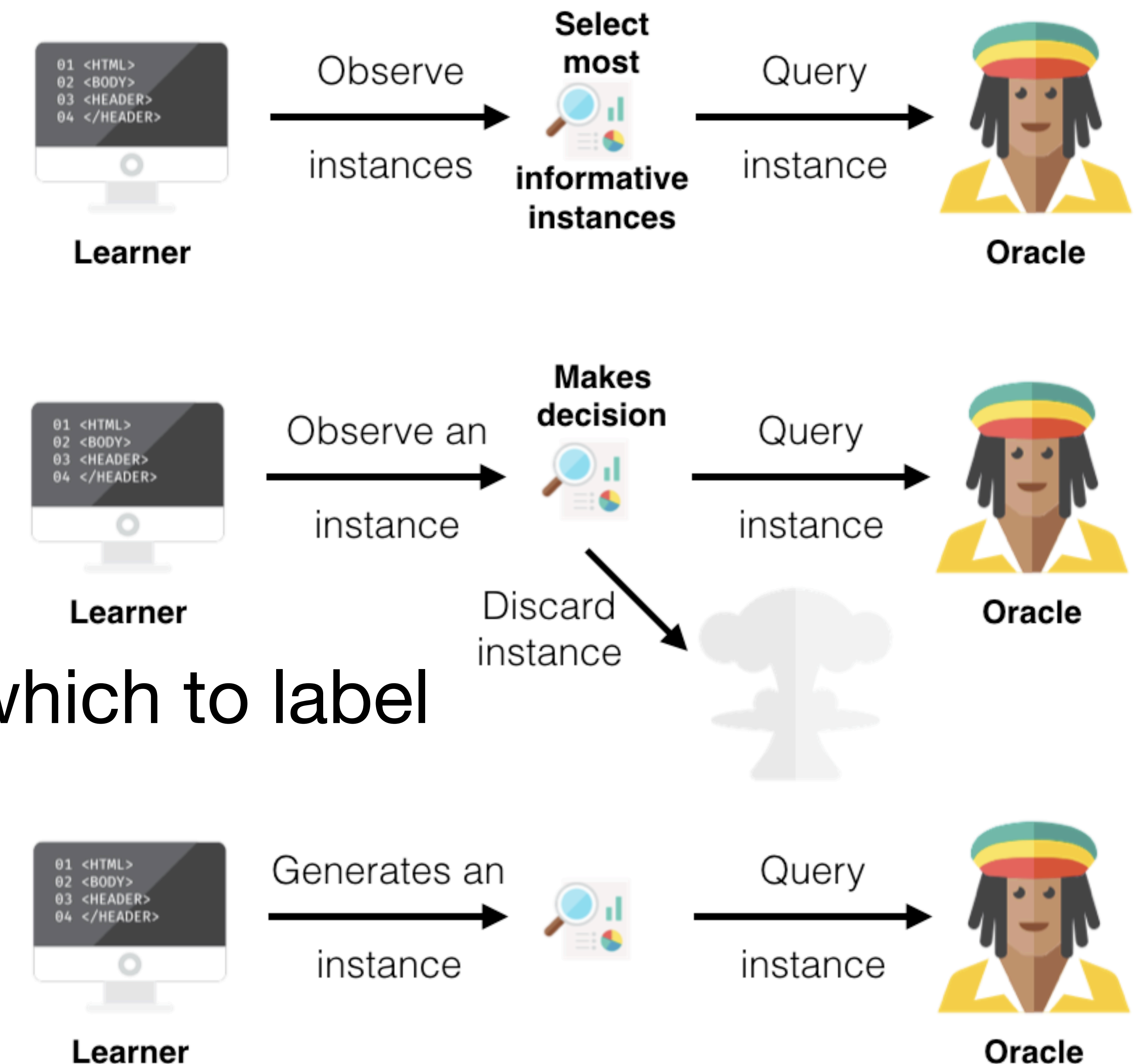




Source: https://www.datacamp.com/community/tutorials/active-learning

- 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?")



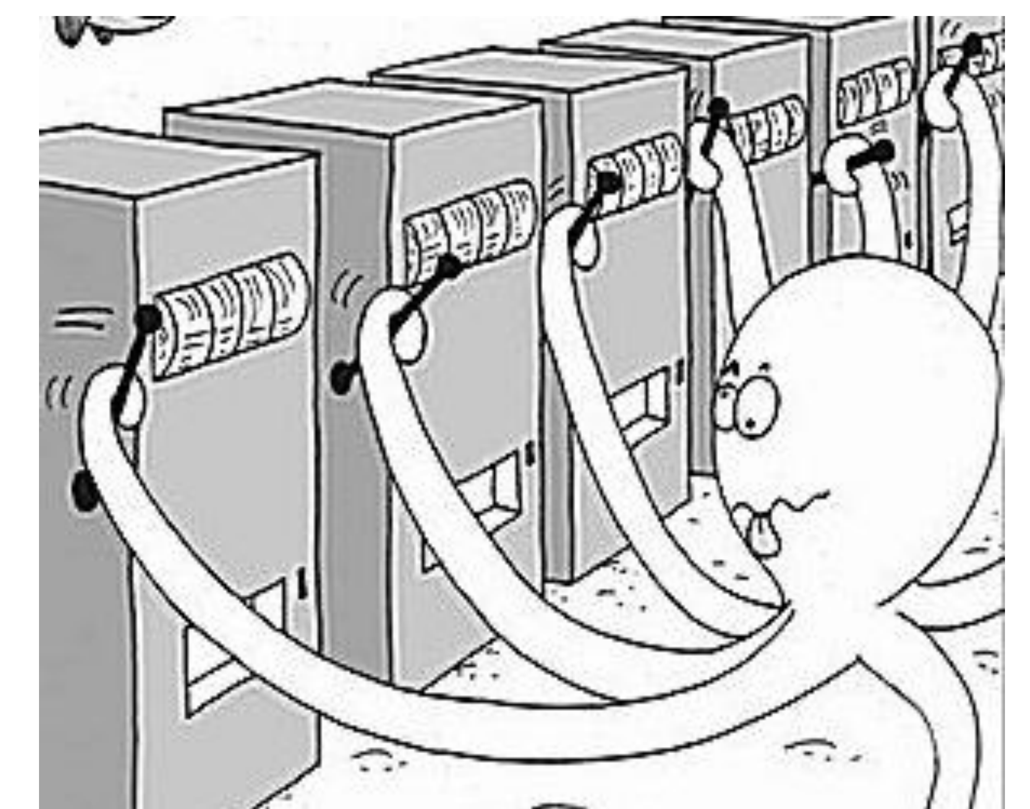

Source: https://www.datacamp.com/community/tutorials/active-learning

# 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**

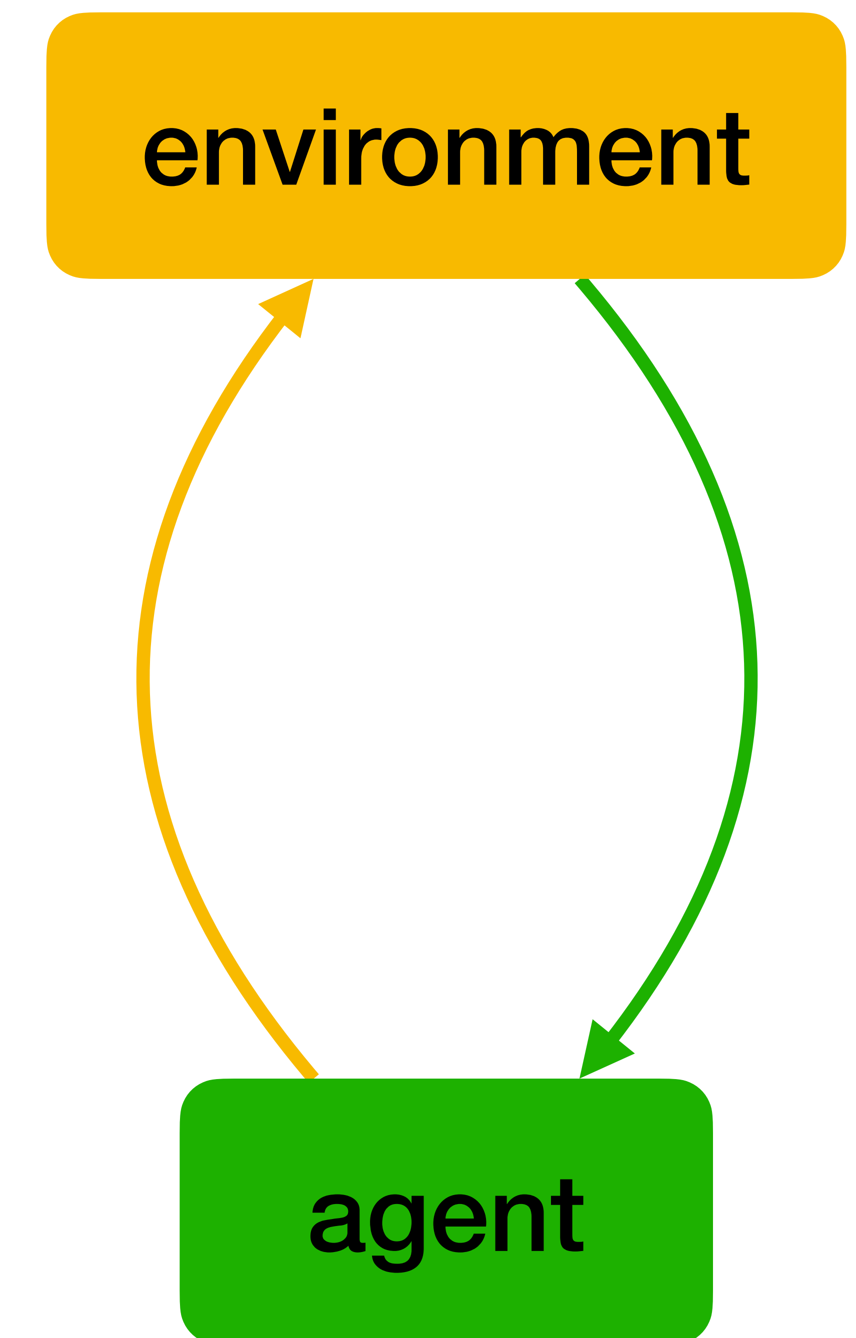


# Optimism under uncertainty

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

# Markov Decision Process (MDP)

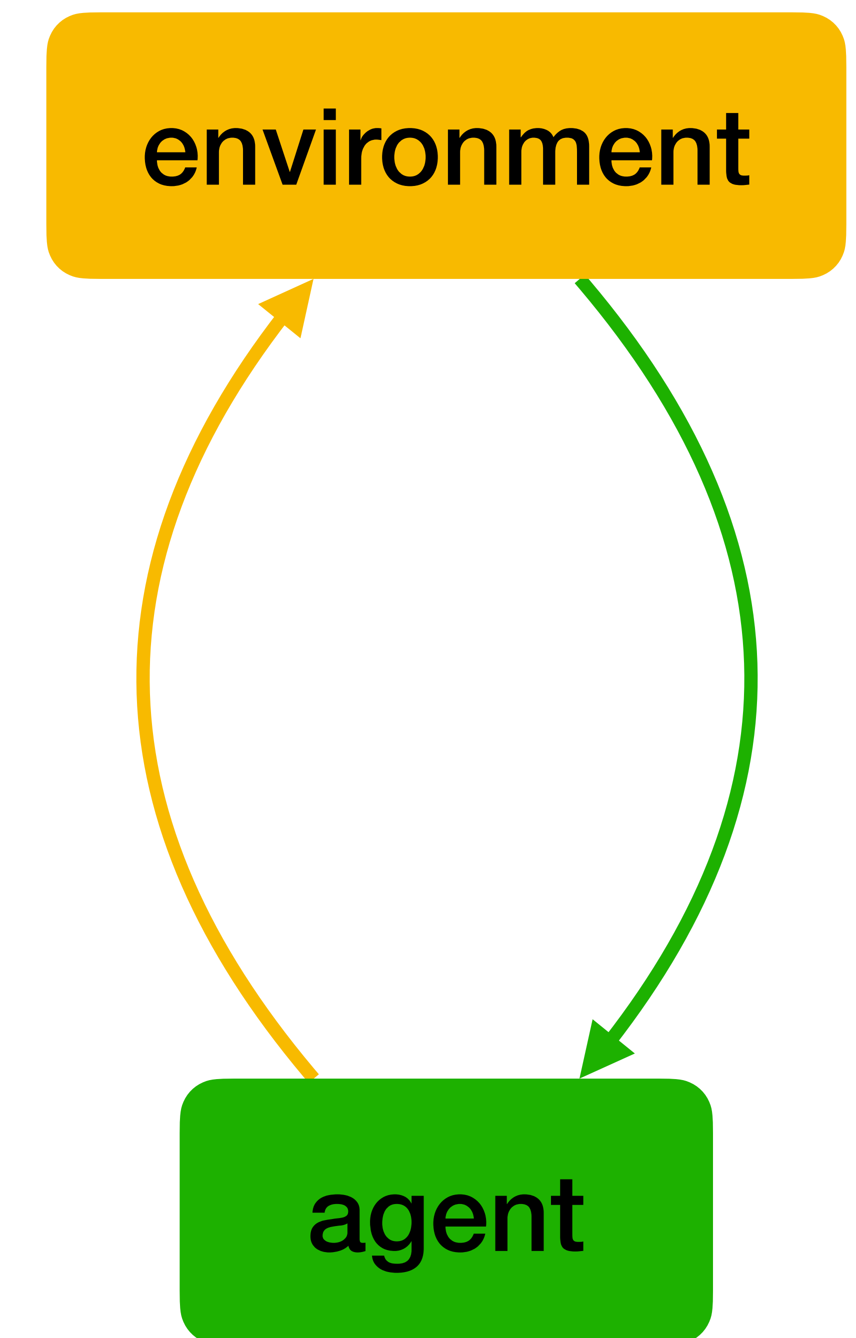
- Model of environment
  - $S$ = set of states
  - $A$ = set of actions
  - $p(s' \mid s, a)$ = probability that $s_{t+1} = s'$, if $s_t = s$ and $a_t = a$



# Trajectories

- The agent's behavior iteratively uses (rolls out) the policy
- Trajectory: $\xi = (s_1, a_1, s_2, a_2, \ldots, s_{T+1})$
- MDP + policy induce distribution over trajectories

$$p_\pi(\xi) = p(s_1)\pi(a_1 \,|\, s_1)p(s_2 \,|\, s_1, a_1)\cdots\pi(a_T \,|\, s_T)p(s_{T+1} \,|\, s_T, a_T)$$
$$= p(s_1)\prod_{t=1}^{T}\pi(a_t \,|\, s_t)p(s_{t+1} \,|\, s_t, a_t)$$



# Learning from Demonstrations (LfD)

- Teacher provides demonstration trajectories $\mathscr{D} = \{\xi^{(1)}, \ldots, \xi^{(m)}\}$
- Learner trains a policy $\pi_\theta$ to minimize a loss $\mathscr{L}(\theta)$
- For example, negative log-likelihood (NLL):

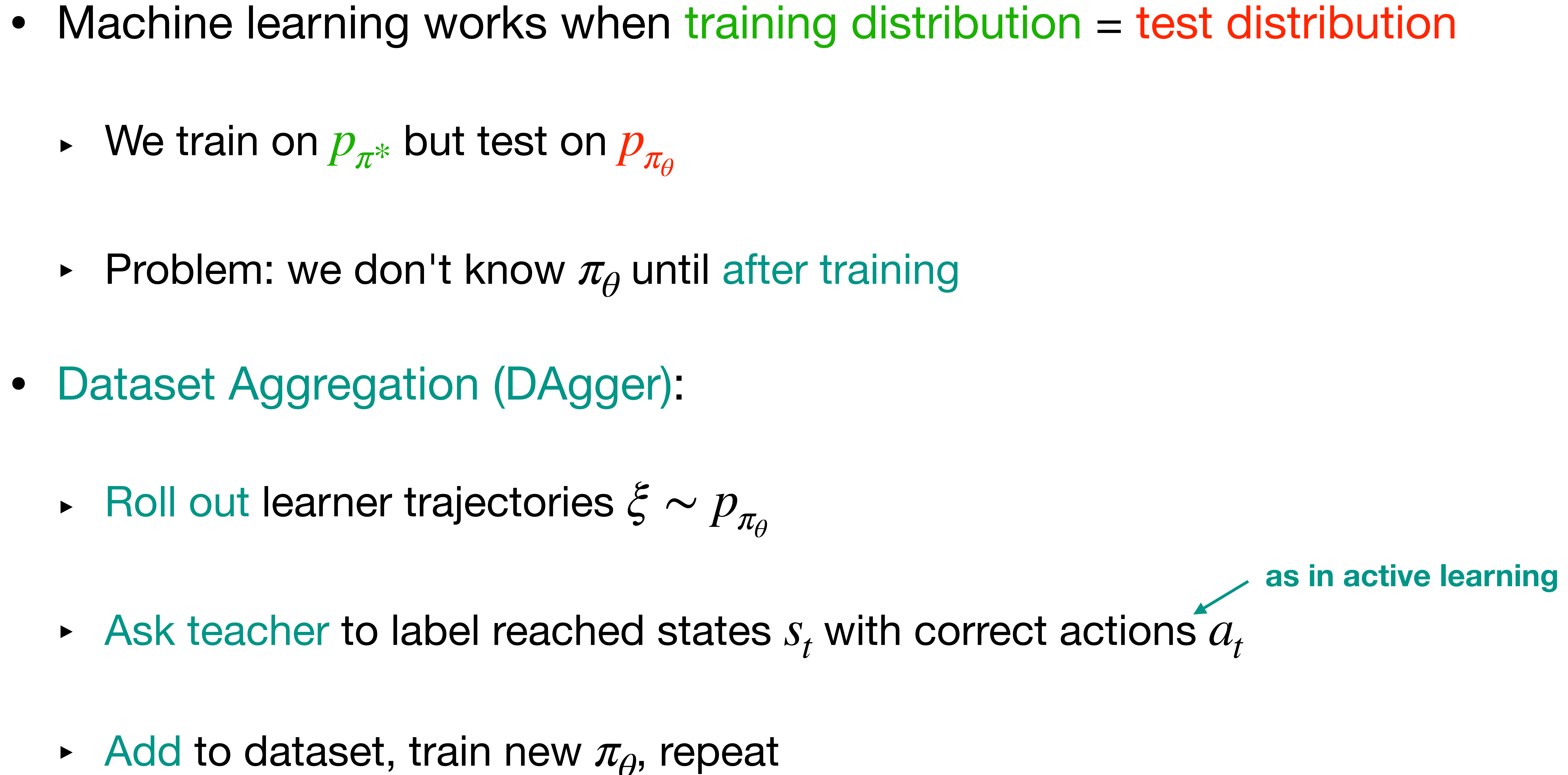
$$\arg\min_\theta \mathscr{L}_\theta(\xi) = \arg\min_\theta(-\log p_\theta(\xi))$$

$$= \arg\max_\theta \left( \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t) + \log p(s_{t+1} \mid s_t, a_t) \right)$$

$$= \arg\max_\theta \sum_{t=1}^{T} \log \pi_\theta(a_t \mid s_t)$$

**model-free**
**= no need to know the environment dynamics** $p$

# Gathering experience



# Returns

- Return = total reward = $R = \sum_t \gamma^t r(s_t, a_t)$
  - Summarize reward sequence $r_t = r(s_t, a_t)$ as single number to be maximized
- Discount factor $\gamma \in [0,1]$
  - Higher weight to short-term rewards (and costs) than long-term
  - Good mathematical properties:
    - Assures convergence, simplifies algorithms, reduces variance
  - Vaguely economically motivated (inflation)

# Optimal policy

- If we know we will use $\pi$ in the future, what should we do now?
  - Greedy policy: $\pi^*(s_t) = \arg\max_{a_t} Q_\pi(s_t, a_t)$
  - In stochastic notation: $\pi^*(a_t | s_t) = 1$ for the greedy action
- If we have a guess for the action-value function $Q(s, a)$
  - Then $V(s) = \max_a Q(s, a)$ is a value function of a better policy
- This gives us a policy improvement step
  - Can be put together with policy evaluation $Q(s, a) \to r + \gamma V(s')$

# Putting it all together: Deep Q Learning (DQN)