Introduction to Artificial Intelligence

Lecture 17 – Learning

CS/CNS/EE 154

Andreas Krause

Announcement

- CS/CNS/EE 155 not offered next term
- Learning project sequence can be continued with
  - CS 141bc or
  - CS 144 and CS 145 or
  - CS 187 and CS 156a

Announcement 2

- Homework 3 due Wed Nov 24
- Project final implementation due Wed Dec 1

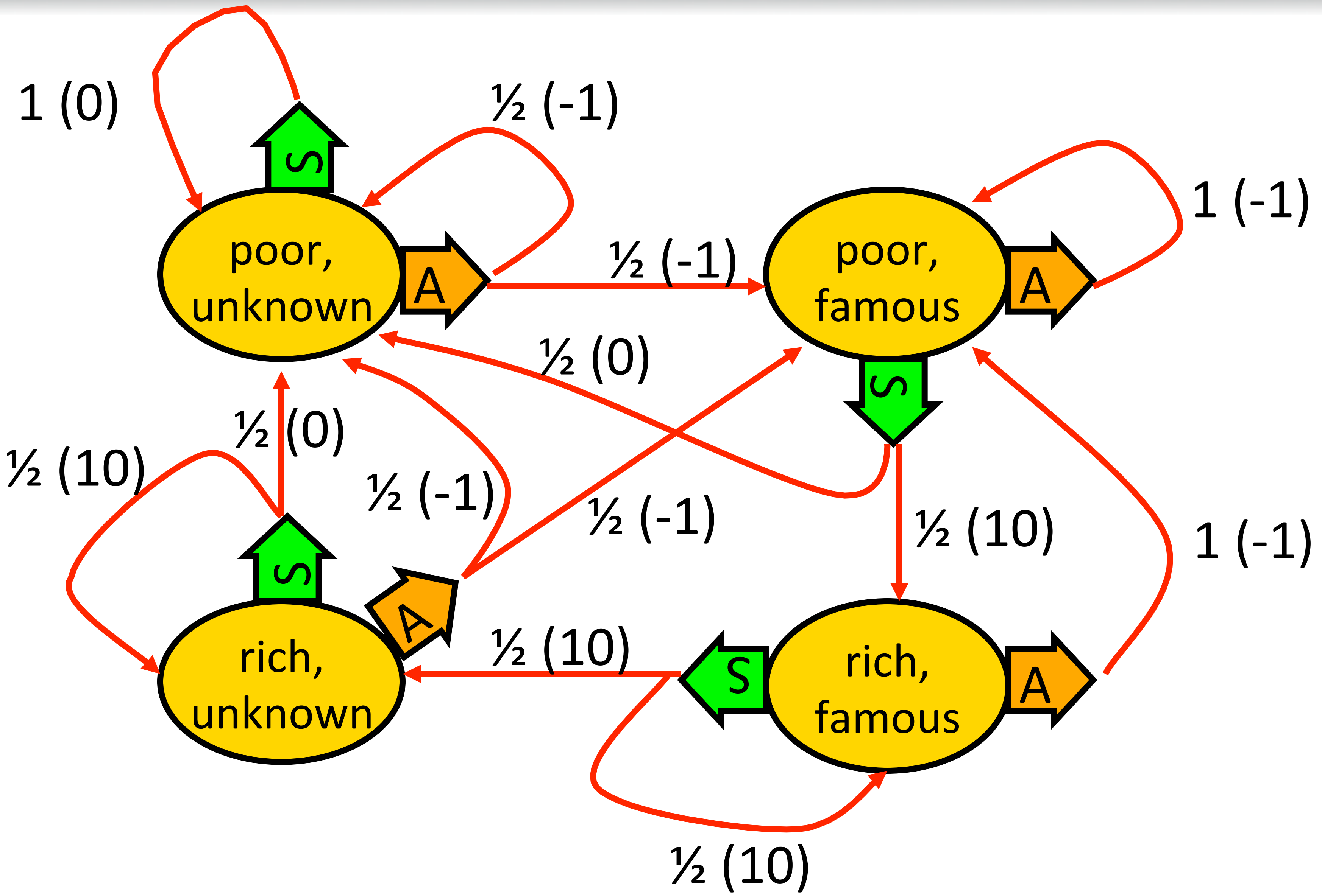
- Exam:
  - Take home, one day (date TBA)
  - Will have information session about exam next week

Markov Decision Processes

- An MDP has
  - A set of states $X = \{x_1,\ldots,x_n\}$ …
  - A set of actions $A = \{a_1,\ldots,a_m\}$
  - A reward function $r(x,a)$ [or random var. with mean $r(x,a)$]
  - Transition probabilities

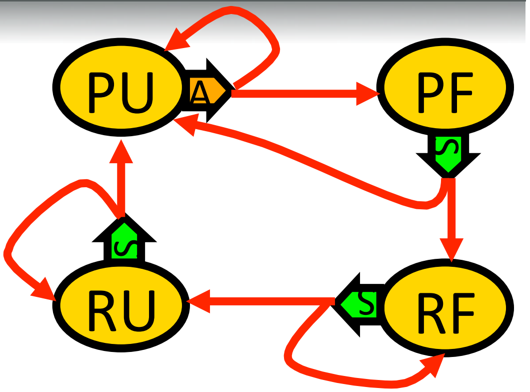
    $P(x'|x,a)$ = Prob(Next state = x' | Action a in state x)
- For now assume r and P are known!
- Want to choose actions to maximize reward

Becoming rich and famous



Planning in MDPs

- Deterministic policy $\pi: X \rightarrow A$
- Induces a **Markov chain**: $X_1,X_2,\ldots,X_t,\ldots$
  with transition probabilities



$$P(X_{t+1}=x' \mid X_t=x) = P(x' \mid x, \pi(x))$$

- Expected value $J(\pi) = E[$

$$\begin{aligned} & r(X_1,\pi(X_1)) \\ &+ \gamma\, r(X_2,\pi(X_2)) \\ &+ \gamma^2\, r(X_3,\pi(X_3)) \\ &+ \ldots \quad] \end{aligned}$$

Computing the value of a policy

- For fixed policy π and each state x, define **value function**

$$V^\pi(x) = J(\pi \mid \text{start in state } x) = \underline{r(x,\pi(x))} + E[\textstyle\sum_t \gamma^t\, r(X_t,\pi(X_t))]$$

Recursion: $V^\pi(x) = r(x,\pi(x)) + \gamma\, E[\sum_t \gamma^{t-1}\, r(X_t,\pi(X_t))]$

$$= r(x,\pi(x)) + \gamma \sum_{x'} P(x' \mid x,\pi(x))\, V^\pi(x')$$

and $J(\pi) = V^\pi(\underset{\uparrow \text{ start state}}{x_0})$

$[V^\pi(1) \dots V^\pi(n)]^T$

$[r(1,\pi(1)), \dots r(n,\pi(n))]^T$

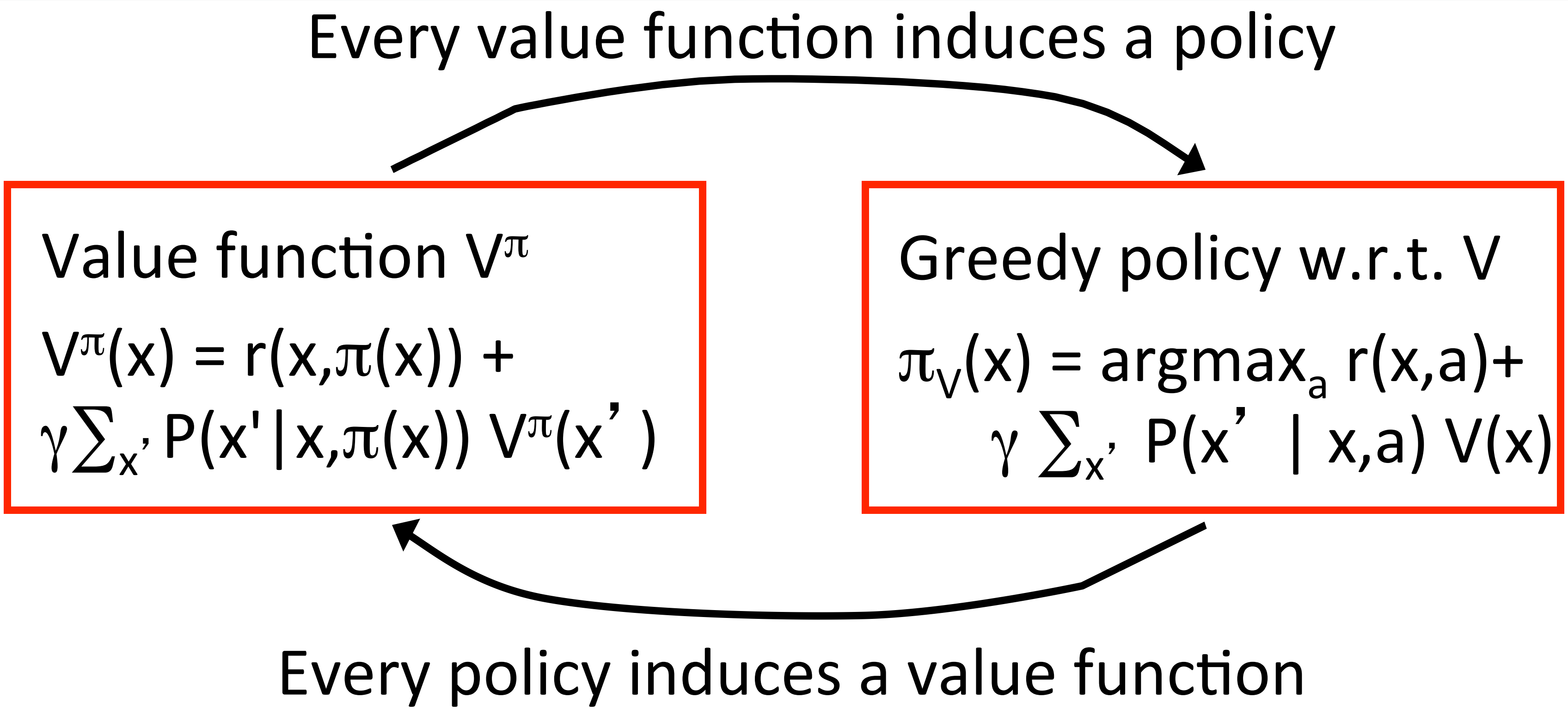
In matrix notation: $V^\pi = r + \gamma T V^\pi$

$$\begin{pmatrix} P(1 \mid 1,\pi(1)) & \dots & P(n \mid 1,\pi(1)) \\ \vdots & & \\ P(1 \mid n,\pi(n)) & \dots & P(n \mid n,\pi(n)) \end{pmatrix}$$

$\Rightarrow V^\pi = (I-\gamma T)^{-1} r$

➔ **Can compute V^π analytically, by matrix inversion! ☺**

Value functions and policies



Every value function induces a policy

Value function V^π

$V^\pi(x) = r(x,\pi(x)) + \gamma \sum_{x'} P(x'|x,\pi(x))\, V^\pi(x')$

Greedy policy w.r.t. V

$\pi_V(x) = \text{argmax}_a\, r(x,a) + \gamma \sum_{x'} P(x' \mid x,a)\, V(x)$

Every policy induces a value function

Thm: Policy optimal ⇔ greedy w.r.t. its induced value function

Policy iteration

- Start with a random policy π
- Until converged do:

```
        Compute value function $V_\pi (x)$
        Compute greedy policy $\pi_G$ w.r.t. $V_\pi$
        Set $\pi \leftarrow \pi_G$
```

- Guaranteed to
  - Monotonically improve $\forall t, x : V^{\pi_{t+1}}(x) \geq V^{\pi_t}(x)$
  - Converge to an optimal policy π^*
- Often performs really well!
- Not known whether it' s polynomial in |X| and |A|!

Value iteration

- Initialize $V_0(x) = \max_a r(x,a)$
- For t = 1 to 1

  For each x, a, let $Q_t(x,a) = r(x,a) + \gamma \sum_{x'} P(x' \mid x,a)\, V_{t-1}(x')$

  For each x let $V_t(x) = \max_a Q_t(x,a)$

  Break if $\max_x \left| V_t(x) - V_{t-1}(x) \right| \leq \varepsilon$

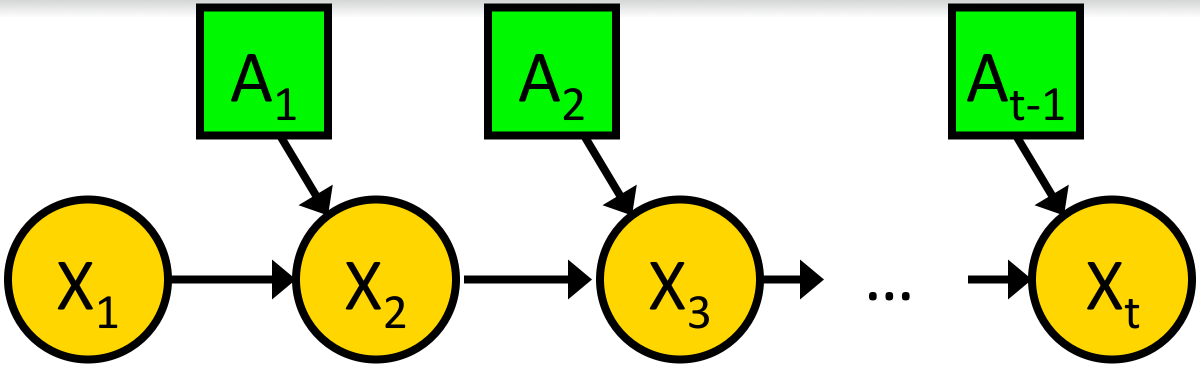
- Then choose greedy policy w.r.t. V_t

- **Guaranteed to converge to ε-optimal policy!**

Applications of MDPs

- Robot path planning (noisy actions)
- Elevator scheduling
- Manufactoring processes
- Network switching and routing
- AI in computer games
- …

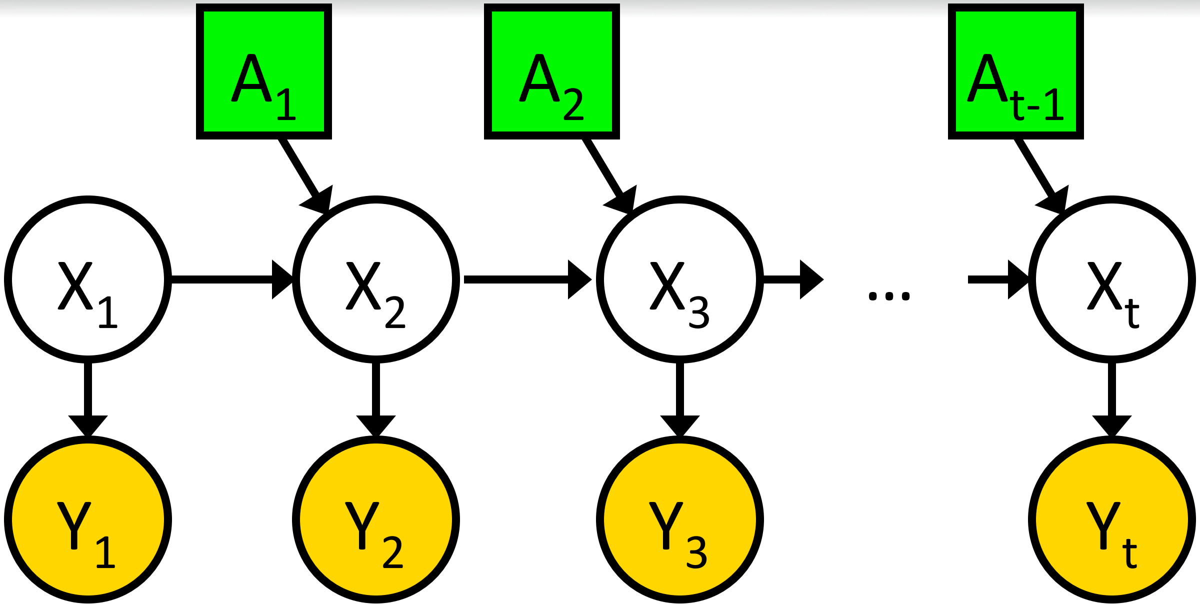
MDP = controlled Markov chain



Specify $P(X_{t+1} \mid X_t, A)$

- State fully observed at every time step
- Action A_t controls transition to X_{t+1}

POMDP = controlled HMM

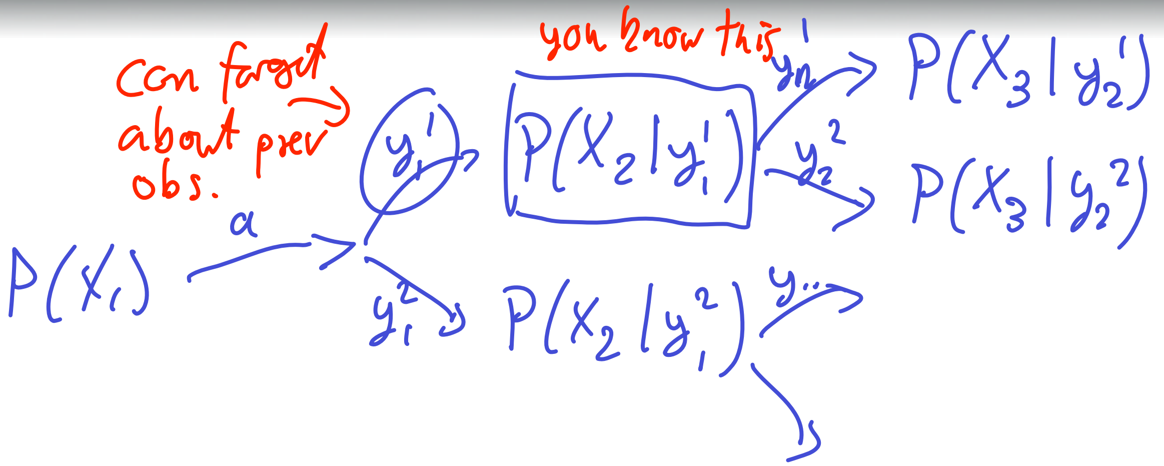


Specify $P(X_{t+1} \mid X_t, A_t)$

$P(Y_t \mid X_t)$

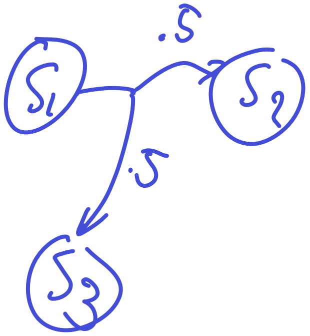
- Only obtain noisy observations Y_t of the hidden state X_t
- **Very powerful model! ☺**
- **Typically extremely intractable ☹**

POMDP = belief state MDP



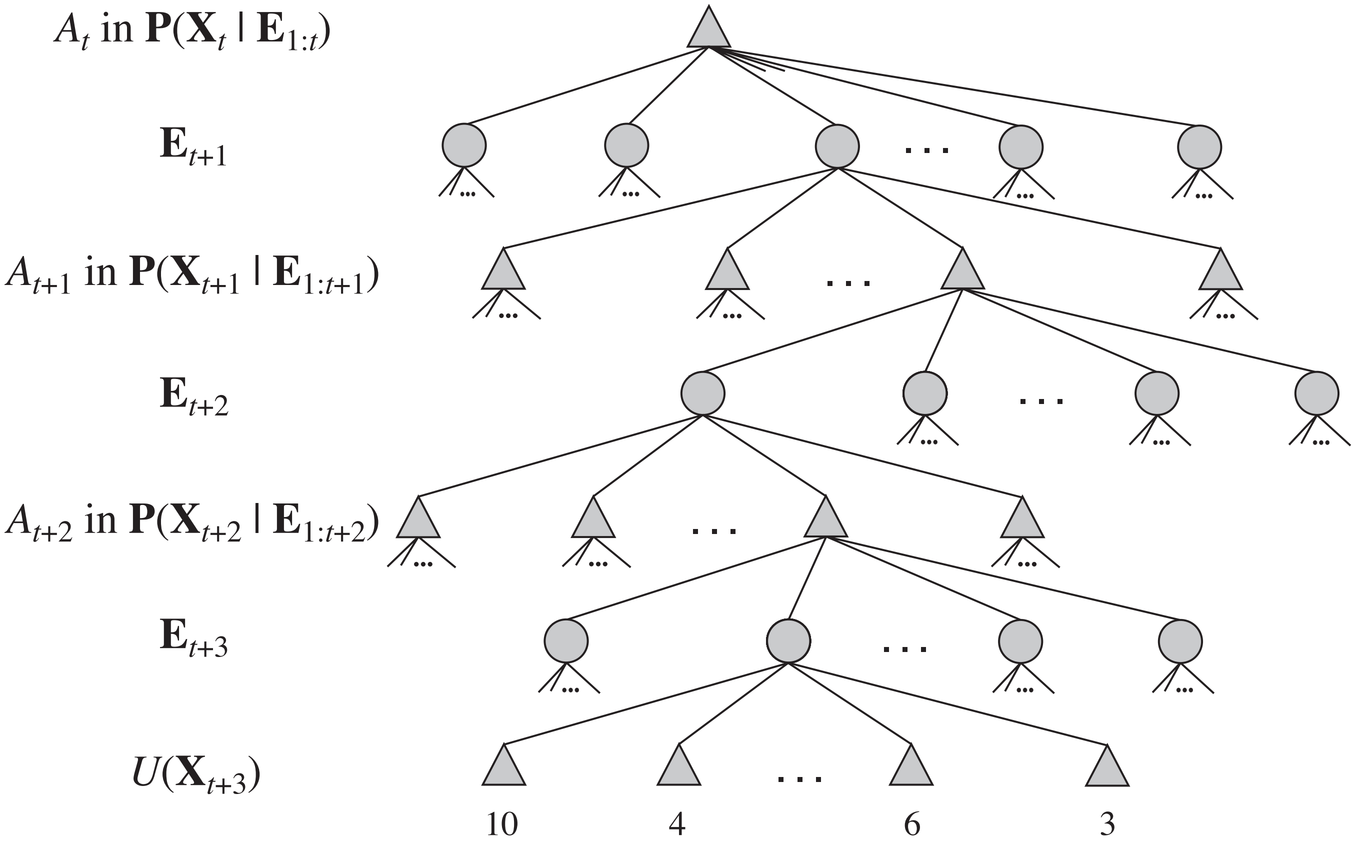
can forget about prev obs.

you know this!



Solving POMDPs

- For finite horizon T, set of reachable belief states is finite (but exponential in T)
- Can calculate optimal action using dynamic programming


A_t in $\mathbf{P}(\mathbf{X}_t \mid \mathbf{E}_{1:t})$
$\mathbf{E}_{t+1}$
A_{t+1} in $\mathbf{P}(\mathbf{X}_{t+1} \mid \mathbf{E}_{1:t+1})$
$\mathbf{E}_{t+2}$
A_{t+2} in $\mathbf{P}(\mathbf{X}_{t+2} \mid \mathbf{E}_{1:t+2})$
$\mathbf{E}_{t+3}$
$U(\mathbf{X}_{t+3})$
...
10
4
6
3

Approximate solutions to POMDPs

- Key idea: most belief states never reached

  ➔ Discretize the belief space by sampling

  ➔ Point based methods:

  - Point based value iteration (PBVI)
  - Point based policy iteration (PBPI)

- Alternative approach: Assume parametric functional form of policy
  - Policy gradient optimization

Learning

Learning

- So far, assumed that models were given to us
  - Bayesian network structure and CPDs
  - Transition/observation models for HMMs and KFs
  - Rewards and transition models for MDPs

- Next topic: Learn models from training data
  - This lecture: Learn parameters from i.i.d. data
  - Next lecture: Learning through exploration (reinforcement learning)

(Supervised) Learning

- Want to learn $f : \mathcal{X} \to \mathcal{Y}$

  X: Set up inputs (discrete, continuous)

  Y: Set of outputs

  from i.i.d. training examples

  $$(x_1, y_1), ..., (x_N, y_N) \sim P(X, Y)$$

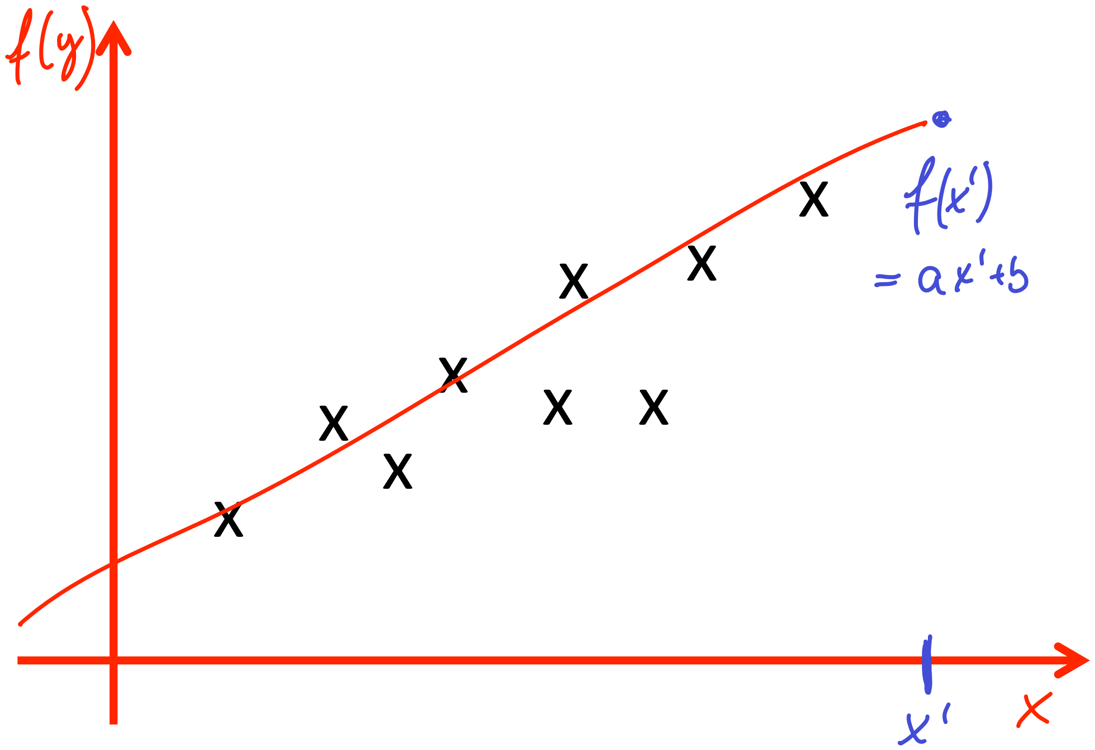
- Goal: minimize generalization error

  $$\mathbb{E}_{X,Y}[\ell(Y, f(X))]$$

  with loss function ℓ, for example:

  $$\ell(y, f(x)) = (y - f(x))^2$$

Linear regression



$$f(x) = a \cdot x + b$$

$$f(\tilde{x}) = w^T \tilde{x}$$

$$\tilde{x} = [1, x]$$

$$w_1 = b, \; w_2 = a$$

$$l(y'; f(x'; w)) = (y' - f(x'; w))^2$$

$$= (y' - w^T x)^2$$

Minimizing training error

- Would like to minimize generalization error

$$w^* = \arg\min_w \mathbb{E}_{X,Y}[(Y - w^T X)^2] = \int p(x,y)(y - w^T x)^2 \, dx \, dy$$

- Don't have access to P(X,Y)! Cannot evaluate generalization error
- Idea: Instead, minimize training error

$$\hat{w} = \arg\min_w \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

$$X = \begin{pmatrix} x_1 \\ \hline \\ \hline x_i \\ \hline \\ \hline \vdots \\ \hline \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \end{pmatrix}$$

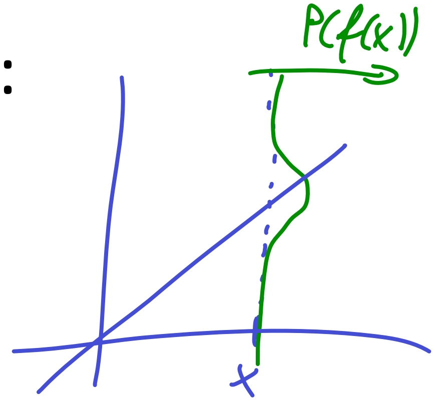
Closed form solution! $\hat{w} = (X^T X)^{-1} X^T y$

Least squares = MLE

- Least squares optimization

$$\hat{w} = \arg\min_{w} \frac{1}{N}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

- Equivalent probabilistic interpretation:



Sps, assume $Y \sim \mathcal{N}(w^T X ; \sigma^2)$

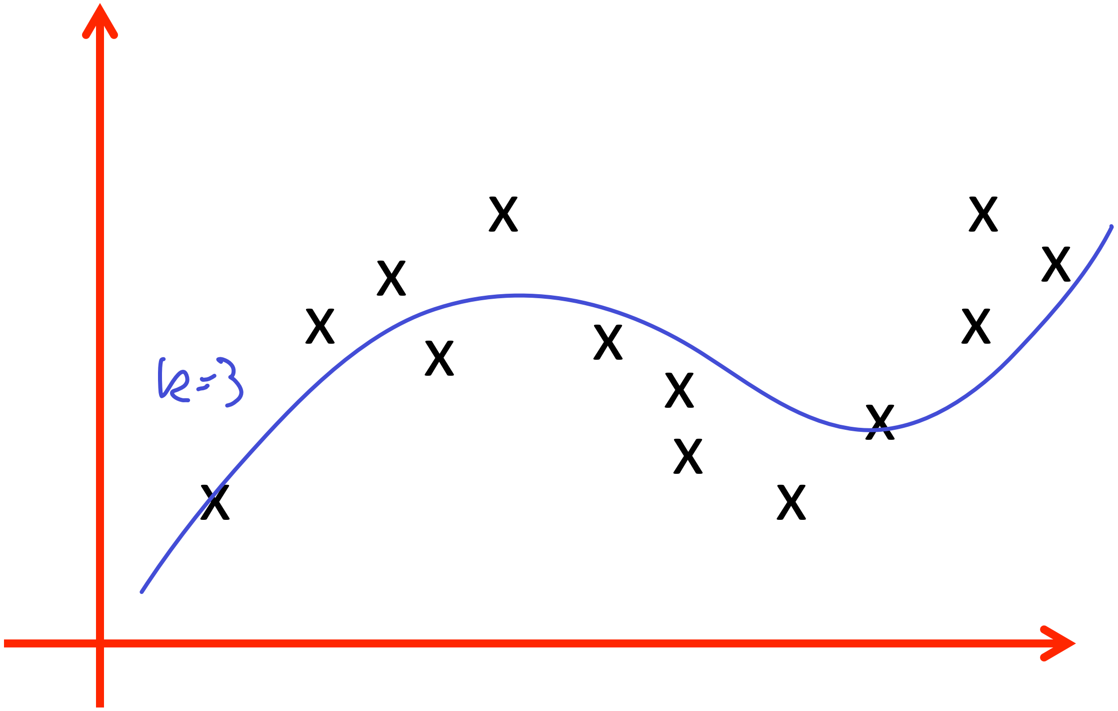
Then.

$$P(D|w) = \prod_{i=1}^{N} P(y_i | x_i, w)$$

$$\ln P(D|w) = \sum_{i=1}^{N} \ln \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - w^T x_i)^2}{\sigma^2}\right)$$

$$= \text{const} - \frac{1}{\sigma^2}\sum_{i=1}^{N} (y_i - w^T x_1)^2$$

Learning non-linear functions



Assume

$$f(x) = \sum_{i=0}^{k} w_i x^k$$

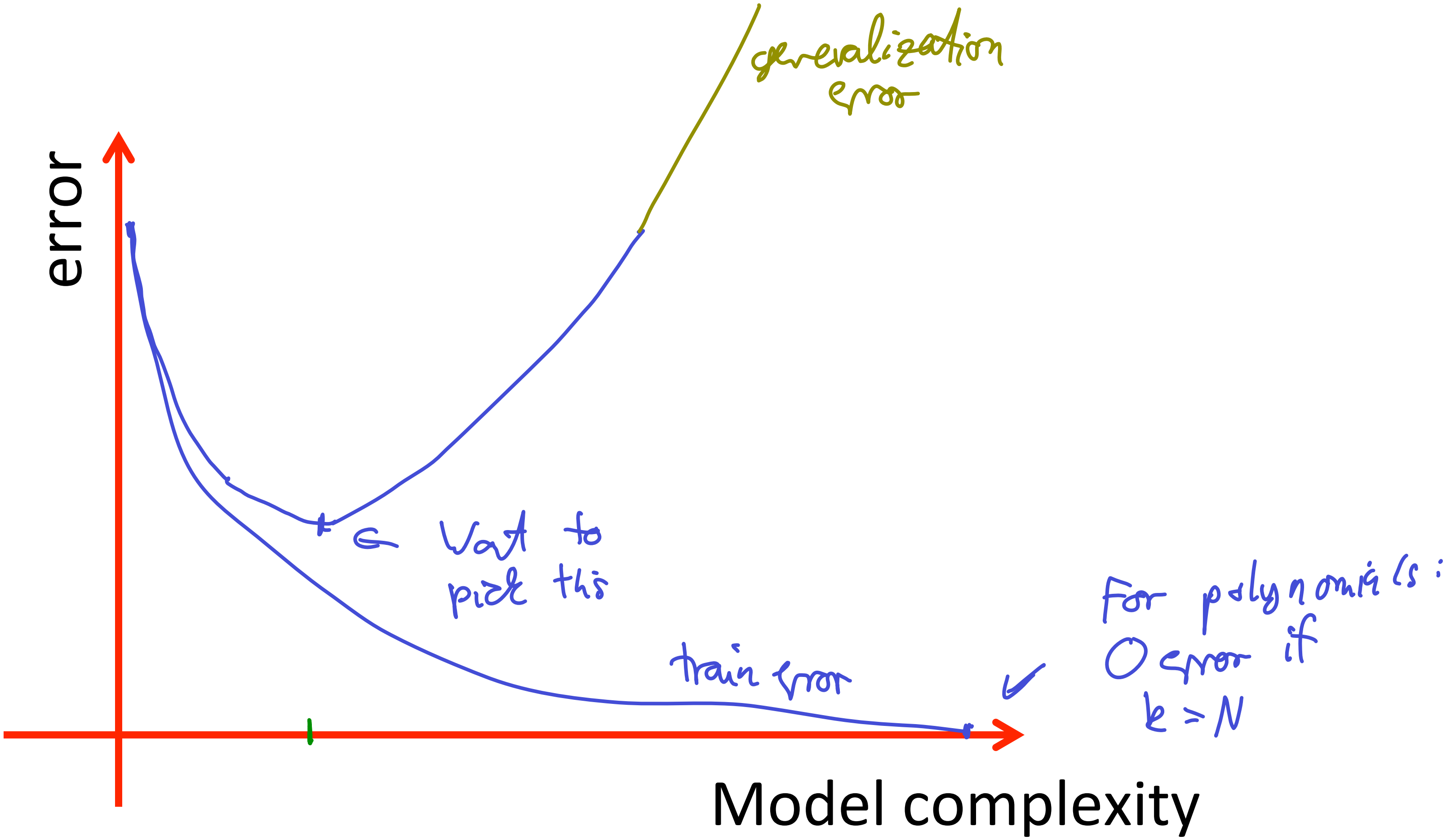
$$\tilde{x} = [1, x, x^2, x^3, \ldots x^k]$$

$$f(\tilde{x}) = w^T \tilde{x}$$

linear regression!

Overfitting

- Min. training error ≠ min. generalization error!



← Want to pick this

For polynomials: 0 error if k = N

Regularization

- Can avoid overfitting by penalizing "complex" functions (large weights)
- Occam's razor
  "The simplest explanation is more likely the correct one"

  ➔ Prior assumption about model complexity

entia non sunt multiplicanda praeter necessitatem

Regularization ≈ Posterior inference

- A priori, assume weights should be small
  ➔ need fewer bits to describe, simpler model

E.g.: $P(w) = \mathcal{N}(0; \lambda^2 \cdot I)$

$$\arg\max_w P(w \mid D) = \arg\max_w P(w) \cdot P(D \mid w)$$

$$= \arg\max_w \ln P(w) + \ln P(D \mid w)$$

$$= \arg\min_w \underbrace{\frac{1}{\sigma^2} \sum_{i=1}^{N} (y_i - w^T x_i)^2}_{\substack{\text{Log likelihood} \\ \| \\ \text{Fit to data}}} + \underbrace{\boxed{\lambda^2} \sum_{j=1}^{k} w_i^2}_{\substack{\text{log prior} \\ \| \\ \text{Complexity of model}}}$$

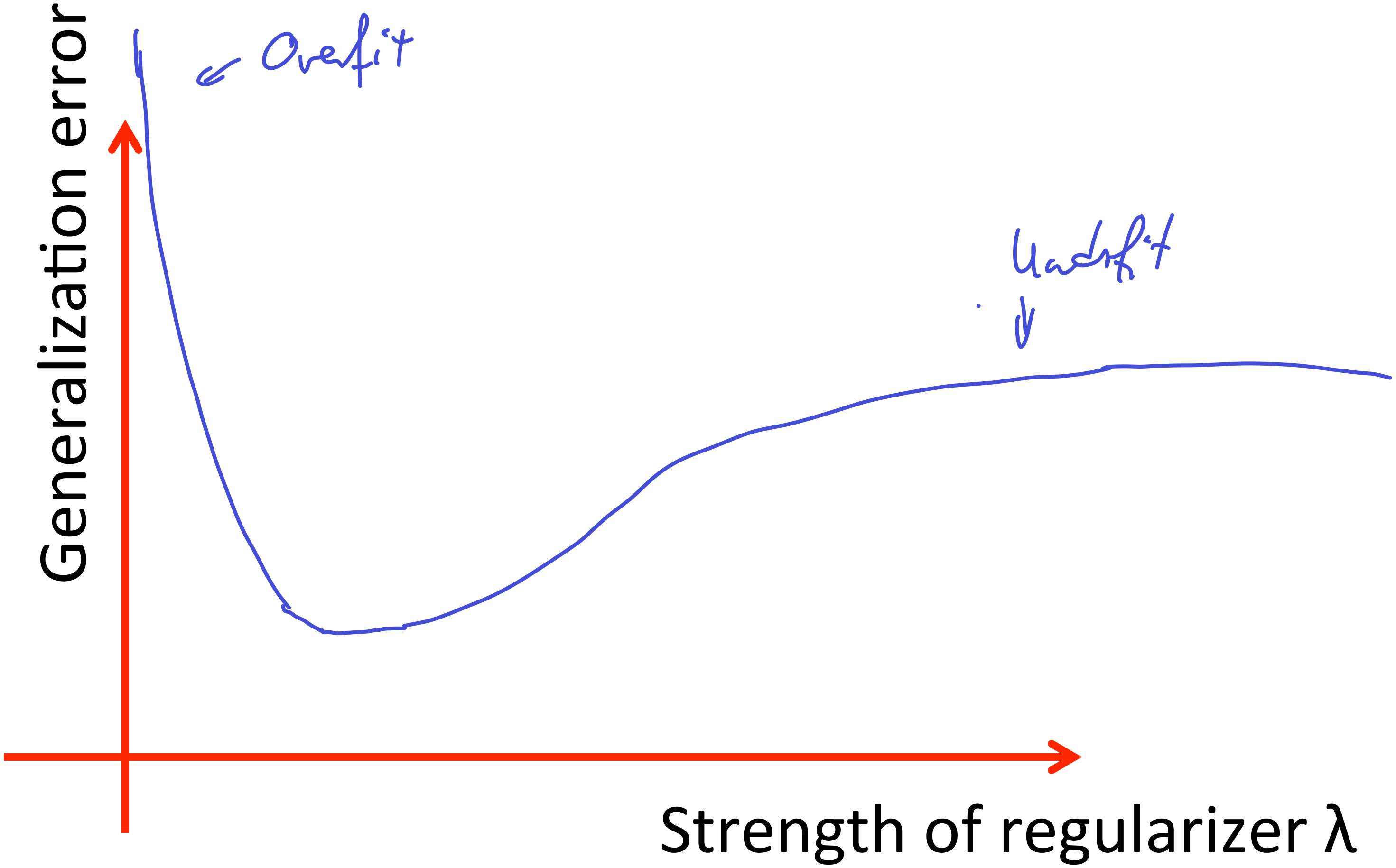
How should we choose λ?

Intuition: Bias variance tradeoff

- Too simple model:
  - Doesn't fit the data well
  - Biased solution
  - "Underfitting"
- Too complex model:
  - Highly sensitive to slight perturbations of the data
  - High variance solution
  - "Overfitting"
- Want to choose regularization to balance out bias and variance

Choosing the right regularizer

- How should we choose the regularization parameter?



Estimating regularization error

- Idea: Split data set into training and test set
- Optimize test set error instead of training set error!
- Is this a good idea?

Expected test error $\mathbb{E}_D\left[\min_\lambda \mathrm{Err}_{Test}(D,\lambda)\right]$

$\neq$

Generalization error $\min_\lambda \mathbb{E}_D\left[\mathrm{Err}_{Test}(D,\lambda)\right]$

Cross-validation

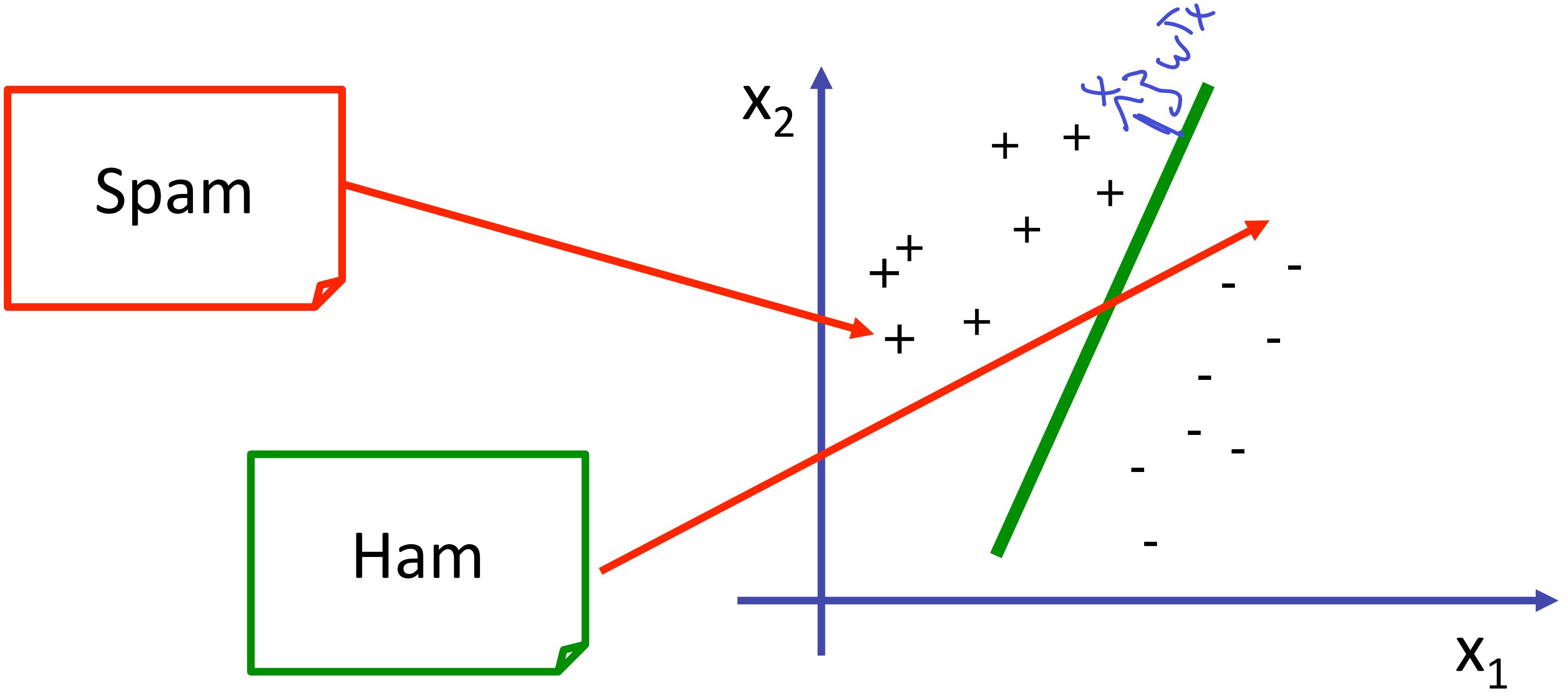
- May overfit if we optimize for fixed training set!
- Remedy: Cross-validation

D_1	D_2	…	D_k

- Split data set into k “folds”
- For each possible regularization parameter setting λ:
  - For i = 1:k
    - Train on all but i-th fold; calculate error E_i
  - Estimate generalization error for param. λ as $\frac{1}{k}\sum_i E_i$
- Can show that cross-validation error “nearly” unbiased!

Classification

- In classification, want to predict discrete label
- For example: binary linear classification $f: X \rightarrow \{+1, -1\}$



$$f(x;w) = \text{sign}(w_0 + w_1\, x_1 + w_2\, x_2)$$

linear classifier

0-1 loss

- Predict according to $f(x;w) = \mathrm{sign}(w^T x)$

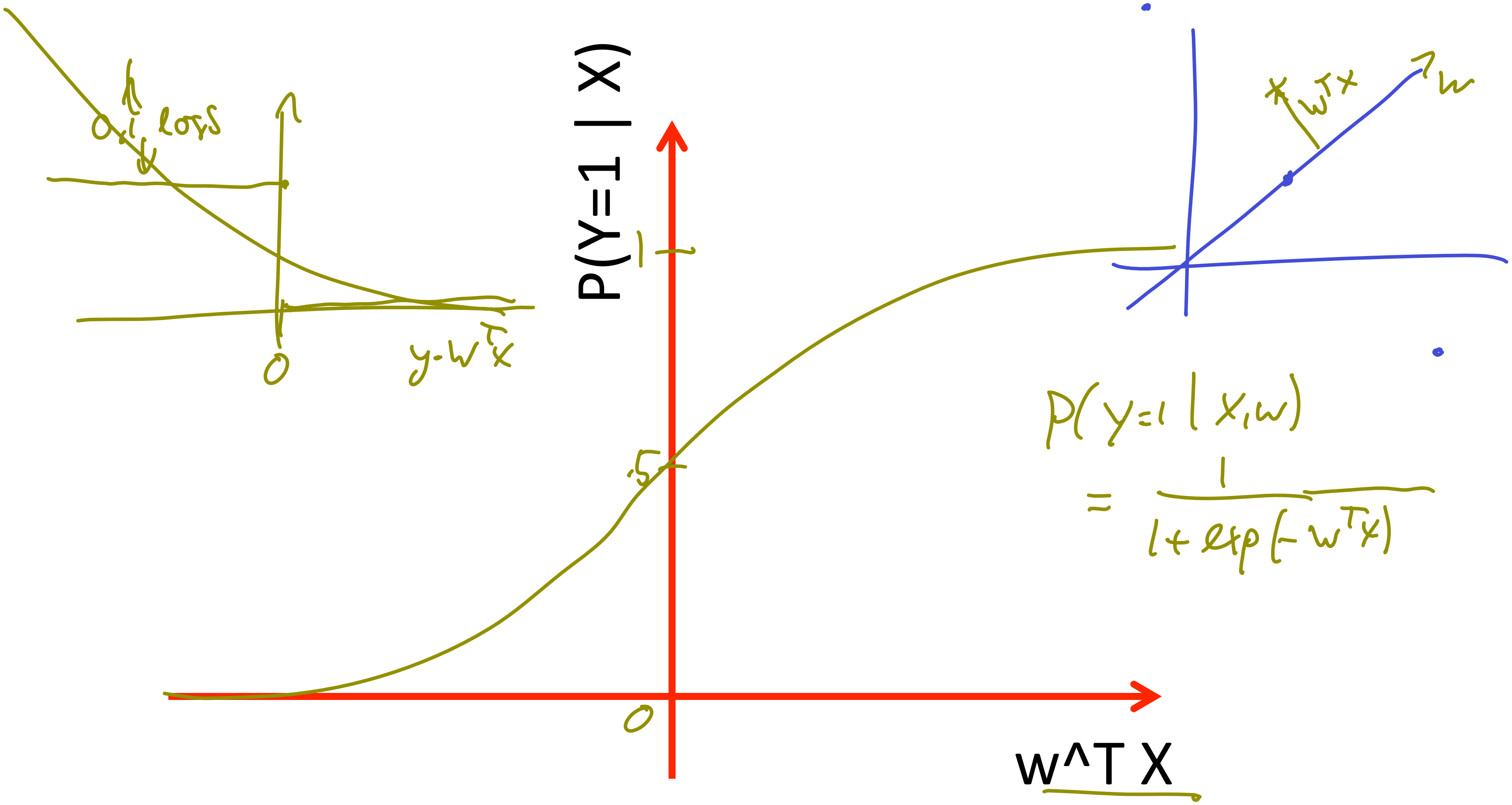
0-1 loss: $\ell(y, f(x;w)) = \begin{cases} 1 & \text{if } y \neq \mathrm{sign}(w^T x) \\ 0 & \text{otherwise} \end{cases}$

$$w^* \in \arg\min_{w} \sum_i \ell(y_i; f(x_i, w))$$

Non-diferentiable, non-convex !!

Logistic regression

- Key idea: Predict the probability of a label



$$P(y=1 \mid x, w) = \frac{1}{1+\exp(-w^T x)}$$

$$\ln P(y \mid x, w) = \underbrace{-\ln(1+\exp(-y \cdot w^T x))}_{\text{log loss}}$$

Logistic regression

- Maximize (conditional) likelihood

$$\ln P(D_Y \mid D_X, w) = \sum_{i=1}^{N} \ln P(y_i \mid x_i, w) = -\sum_{i=1}^{N} \log\Big(1 + \exp(-y_i w^T x_i)\Big)$$

- Convex, differentiable!
- Can find optimal weights w efficiently!

- Can regularize by putting prior on weights w (exactly as in linear regression)