南方科技大学

MAT8034: Machine Learning

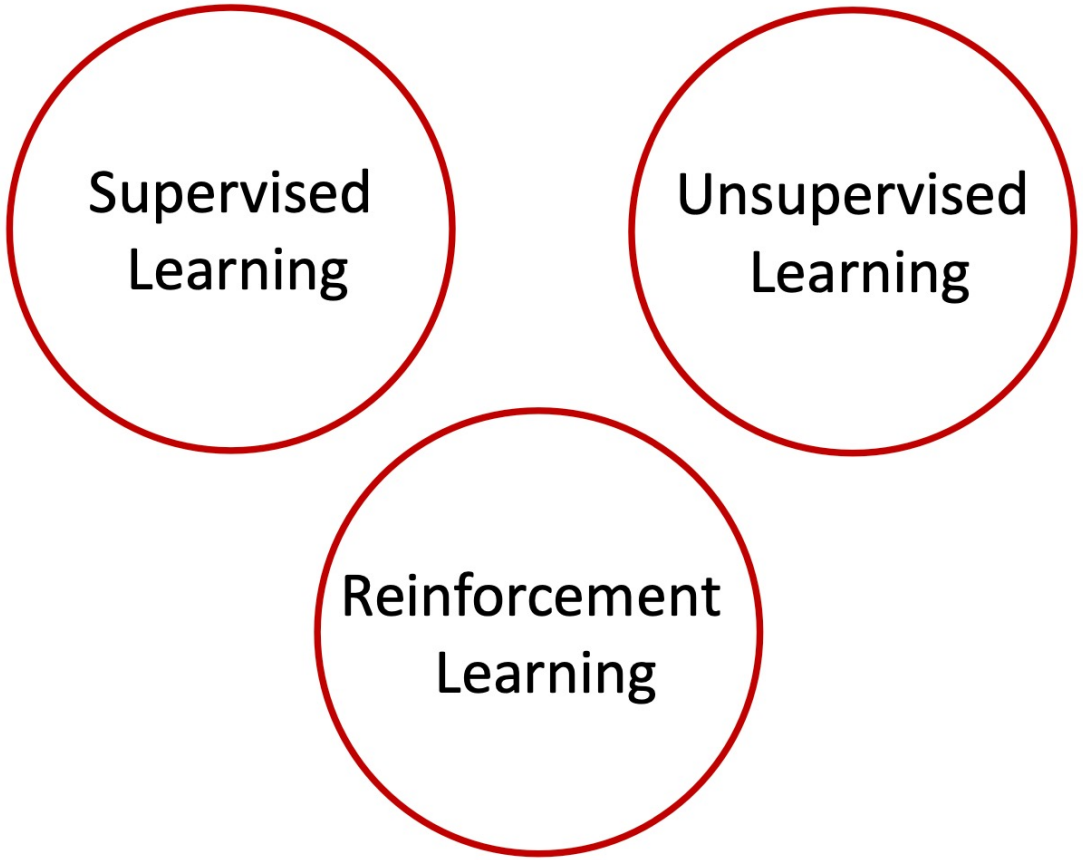
Final Review

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2025/index.html

View based on tasks

- A simplistic view based on tasks



Supervised learning

- Algorithms
  - Linear regression
  - Logistic regression
  - Generalized linear models
  - Generative learning
  - Kernel methods
  - Deep learning
- Performance
  - Generalization, regularization, model-selection

Unsupervised learning

- Algorithms
  - K-means
  - Expectation Maximization
  - PCA
  - ICA

Reinforcement learning

- MDP
  - Algorithms: Value iteration, policy iteration, policy evaluation, policy extraction
- Bandits (exploration-exploitation trade-off)
  - Algorithms: ETC, epsilon-greedy, UCB, TS
- RL
  - Model-based
  - Model-free:
    - Direct estimation, TD-learning, Q-learning
  - Policy-based: Policy gradient

  - Function approximation
  - Deep RL: value-based, policy-based

View based on the workflow

- Identify the task type
  - Regression, classification, clustering, reduction, RL…
- Determine a hypothesis class
  - Linear function, GLM, kernel, neural network->label, log-odd, value function…
- Define the objective function
  - Maximum likelihood, empirical risk minimization
- Optimize the objective function
  - SGD, Newton's; EM (construct lower bound), RL (sampling)
- Evaluate the performance
  - Generalization, regularization, model selection

Supervised Learning

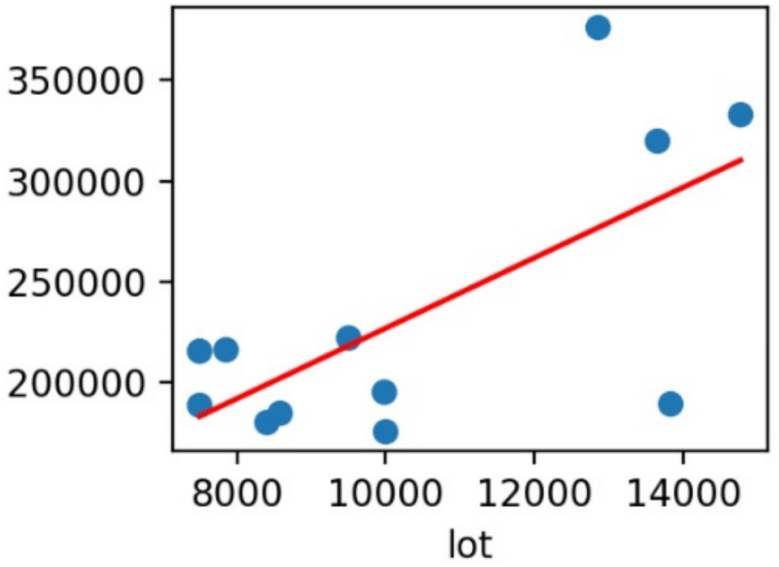
Linear regression

Linear regression

- LMS
  - Gradient descent
  - Normal equation
- Justification for LMS
  - Log likelihood

How to represent h?

- Simplest fit



- $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- Vector notation?

How to learn the parameter?

- Least-square cost function

$$J_\theta = \frac{1}{2}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Least Mean Square Algorithm

- Thus the update rule can be written as

$$\theta_j^{(t+1)} = \theta_j^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}.$$

We write this in *vector notation* for $j = 0, \ldots, d$ as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$$

Batch & stochastic gradient descent

- Consider the update rule $\theta^{(t+1)} = \theta^{(t)} - \alpha \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}.$
- Repeat until converge

- A single update, we examine all data points
- In some modern applications, n may be in the billions or trillions!
  - E.g., we try to "predict" every word on the web
- Idea: Sample a few points (maybe even just one!) to approximate the gradient called Stochastic Gradient (SGD).
  - SGD is the workhorse of modern ML, e.g., pytorch & tensorflow

The matrix form

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(n)})^T & — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(n)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^{(1)}) - y^{(1)} \\ \vdots \\ h_\theta(x^{(n)}) - y^{(n)} \end{bmatrix}. \end{aligned}$$

$$\begin{aligned} \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) &= \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 \\ &= J(\theta) \end{aligned}$$

Normal equation

- Hope to minimize $J(\theta)$, find θ such that $\nabla J(\theta) = 0$

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T (X\theta - \vec{y}) \\
&= \frac{1}{2}\nabla_\theta \left((X\theta)^T X\theta - (X\theta)^T \vec{y} - \vec{y}^T (X\theta) + \vec{y}^T \vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T (X^T X)\theta - \vec{y}^T (X\theta) - \vec{y}^T (X\theta)\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T (X^T X)\theta - 2(X^T \vec{y})^T \theta\right) \\
&= \frac{1}{2}\left(2X^T X\theta - 2X^T \vec{y}\right) \\
&= X^T X\theta - X^T \vec{y}
\end{aligned}
$$

Some useful facts:

$$a^T b = b^T a$$

$$\nabla_x b^T x = b$$

$$\nabla_x x^T A x = 2Ax$$

$$\theta = (X^T X)^{-1} X^T \vec{y}.$$

A Justification for Least Squares?

We make an assumption (common in statistics) that the data are *generated* according to some model (that may contain random choices). That is,

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}.$$

Here, $\varepsilon^{(i)}$ is a random variable that captures "noise" that is, unmodeled effects, measurement errors, etc.

> Please keep in mind: this is just a model! As they say, all models are wrong but some models are *useful.* This model has been *shockingly* useful.

What do we expect of the noise?

What properties should we expect from $\varepsilon^{(i)}$

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}.$$

Again, it's a model and $\varepsilon^{(i)}$ is a random variable:

- $\mathbb{E}[\varepsilon^{(i)}] = 0$ – the noise is unbiased.
- The errors for different points are *independent* and *identically distributed* (called, **iid**)

$$\mathbb{E}[\varepsilon^{(i)}\varepsilon^{(j)}] = \mathbb{E}[\varepsilon^{(i)}]\mathbb{E}[\varepsilon^{(j)}] \text{ for } i \neq j.$$

and

$$\mathbb{E}\left[\left(\varepsilon^{(i)}\right)^2\right] = \sigma^2$$

Here σ^2 is some measure of *how noisy* the data are. Turns out, this effectively defines the *Gaussian or Normal distribution.*

Likelihoods!

- Intuition: among many distributions, pick the one that agrees with the data the most (is most "likely")

$$L(\theta) = p(y|X;\theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)};\theta) \qquad \text{iid assumption}$$

$$= \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\frac{(x^{(i)}\theta - y^{(i)})^2}{2\sigma^2} \right\}$$

Log Likelihoods!

- For convenience, use the Log Likelihood

$$\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{n} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2.
\end{aligned}$$

- Finding a θ that maximizes the log likelihood
  - What happens?
  - Equivalent to minimizing $\frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})^2$

Logistic regression

Intuition of logistic regression

- Consider the odd: p/(1-p) $\in (0, +\infty)$
- Consider the log odd:
  - Logit(p) := log p/(1-p) $\in (-\infty, +\infty)$

- Good properties:
  - p->0, logit -> $-\infty$; p->1, logit -> $+\infty$
  - Symmetry: Logit(p)=-Logit(1-p)
  - Use linear model to approximate the logit: $\theta^\top x$ ~ Logit(p)= log p/(1-p)
  - $p \sim \frac{1}{1+\exp(-\theta^\top x)} := \text{sigmoid}(\theta^\top x) = h_\theta(x)$

Likelihood function

- Let's write the Likelihood function. Recall:

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

- Then,

$$L(\theta) = P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$
$$= \prod_{i=1}^{n} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \quad \text{exponents encode "if-then"}$$

- Taking logs to compute the log likelihood $\ell(\theta)$ we have:

$$\ell(\theta) = \log L(\theta) = \sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

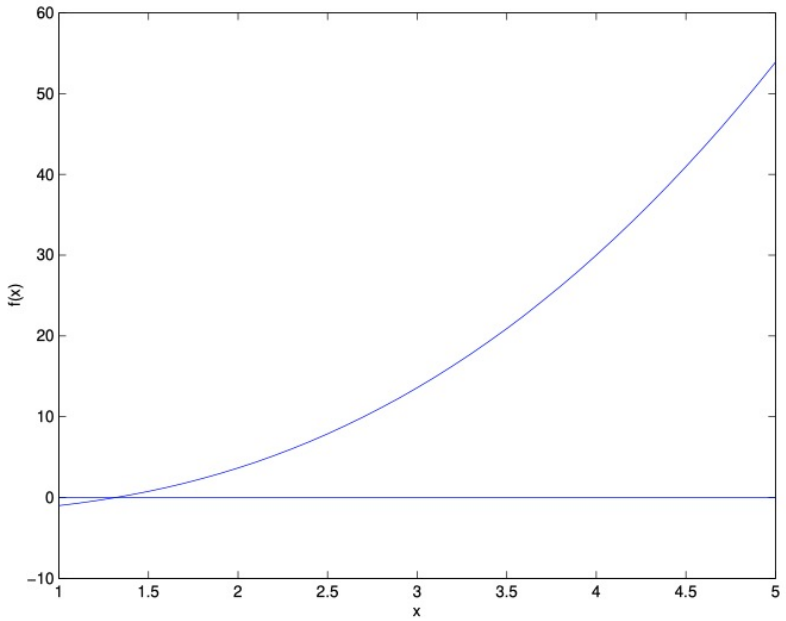
Gradient ascent for log likelihood

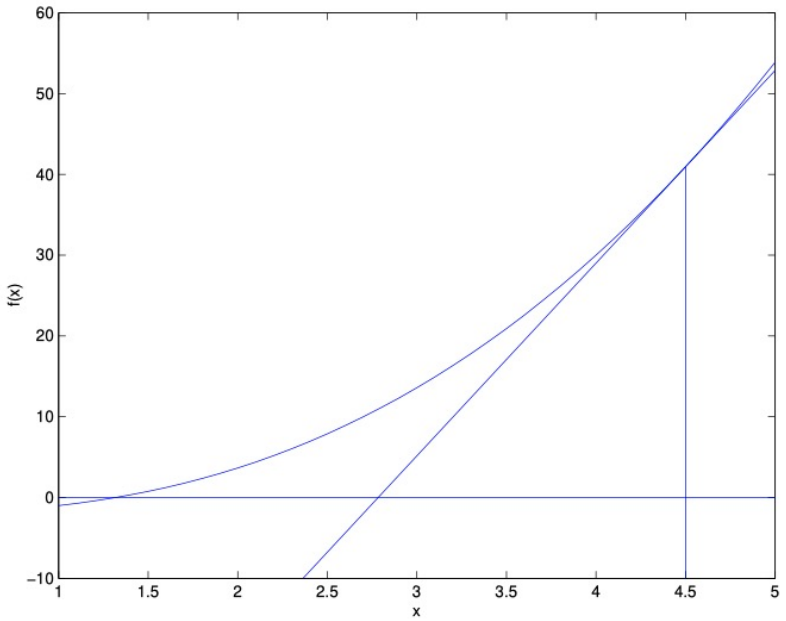
$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j \\
&= \left(y - h_\theta(x) \right) x_j
\end{aligned}
$$

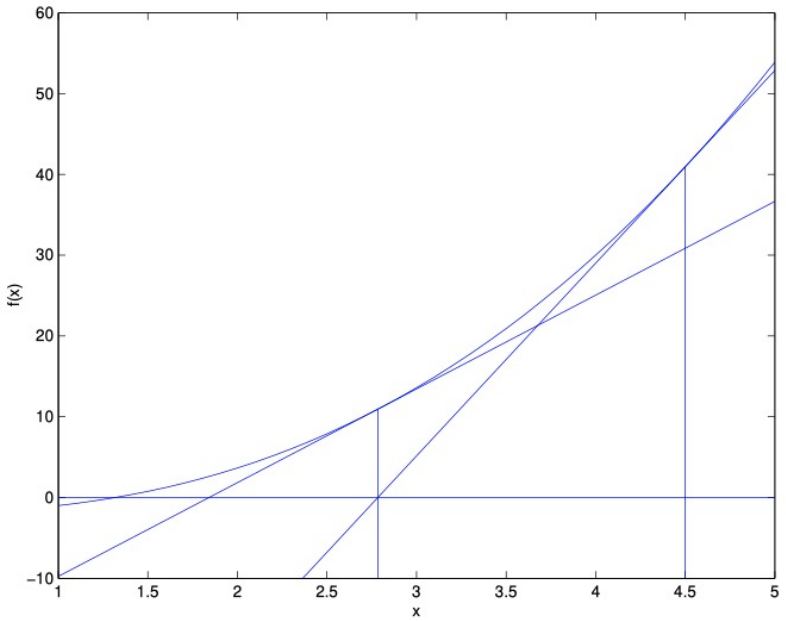
$$
\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}
$$

Newton's method

Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ find θ s.t. $f(\theta) = 0$







Newton's method

- Suppose $\theta_n - \theta_{n+1} = \Delta$
- $\frac{f(\theta_n)-0}{\Delta} = f'(\theta_n)$
- $\theta_n - \theta_{n+1} = \Delta = \frac{f(\theta_n)}{f'(\theta_n)}$

- So the update rule in 1d $\theta := \theta - \frac{f(\theta)}{f'(\theta)}$

- To maximizing the log likelihood? $\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$

Multi-class classification: Softmax function

- Define the softmax function $\text{softmax} : \mathbb{R}^k \to \mathbb{R}^k$ as

$$\text{softmax}(t_1, \ldots, t_k) = \begin{bmatrix} \frac{\exp(t_1)}{\sum_{j=1}^{k} \exp(t_j)} \\ \vdots \\ \frac{\exp(t_k)}{\sum_{j=1}^{k} \exp(t_j)} \end{bmatrix}. \tag{2.9}$$

- Let $(t_1, \ldots, t_k) = (\theta_1^\top x, \cdots, \theta_k^\top x)$

$$\begin{bmatrix} P(y = 1 \mid x; \theta) \\ \vdots \\ P(y = k \mid x; \theta) \end{bmatrix} = \text{softmax}(t_1, \cdots, t_k) = \begin{bmatrix} \frac{\exp(\theta_1^\top x)}{\sum_{j=1}^{k} \exp(\theta_j^\top x)} \\ \vdots \\ \frac{\exp(\theta_k^\top x)}{\sum_{j=1}^{k} \exp(\theta_j^\top x)} \end{bmatrix}$$

GLM

GLM: Motivation

- In the regression problem $y|x;\theta \sim \mathcal{N}(\mu, \sigma^2)$
- In the classification problem $y|x;\theta \sim \text{Bernoulli}(\phi)$

- Whether these distributions can be uniformly represented?

- If P has a a special form, then inference and learning come for free

The exponential family

- $p(y;\eta) = b(y)\exp(\eta^T T(y) - a(\eta))$

- y: data label (scalar)
- η: natural parameter
- $T(y)$: sufficient statistic
- $b(y)$: base measure, depend on y, but not η (scalar)
- $a(\eta)$: log partition function (scalar)

$$1 = \sum_y P(y;\eta) = e^{-a(\eta)} \sum_y b(y)\exp\left\{\eta^T T(y)\right\}$$

$$\Longrightarrow \quad a(\eta) = \log \sum_y b(y)\exp\left\{\eta^T T(y)\right\}$$

An observation

- Notice that for a Gaussian with mean μ we had

$$\eta = \mu, T(y) = y, a(\eta) = \frac{1}{2}\eta^2.$$

- We observe something peculiar:

$$\partial_\eta a(\eta) = \eta = \mu = \mathbb{E}[y] \text{ and } \partial_\eta^2 a(\eta) = 1 = \sigma^2 = \text{var}(y)$$

- That is, derivatives of the log partition function is the expectation and variance. Same for Bernoulli.

Is this true in general?

GLM: Three assumptions/design choices

1. $y \mid x; \theta \sim \text{ExponentialFamily}(\eta)$. I.e., given x and θ, the distribution of y follows some exponential family distribution, with parameter η.

2. Given x, our goal is to predict the expected value of $T(y)$ given x. In most of our examples, we will have $T(y) = y$, so this means we would like the prediction $h(x)$ output by our learned hypothesis h to satisfy $h(x) = \text{E}[y|x]$. (Note that this assumption is satisfied in the choices for $h_\theta(x)$ for both logistic regression and linear regression. For instance, in logistic regression, we had $h_\theta(x) = p(y = 1|x; \theta) = 0 \cdot p(y = 0|x; \theta) + 1 \cdot p(y = 1|x; \theta) = \text{E}[y|x; \theta]$.)

3. The natural parameter η and the inputs x are related linearly: $\eta = \theta^T x$. (Or, if η is vector-valued, then $\eta_i = \theta_i^T x$.)

Workflow of GLMs

- Model formulation

Model Parameter		Natural Parameter		Canonical
θ	$\overset{\theta^T x}{\longmapsto}$	η	$\overset{g}{\longmapsto}$	ϕ : Bernoulli μ : Gaussian λ : Poisson

- Maximum log-likelihood $\max_{\theta} \log p(y \mid x; \theta)$

- Gradient ascent to optimize $\theta^{(t+1)} = \theta^{(t)} + \alpha \left(y^{(i)} - h_{\theta^{(t)}}(x^{(i)}) \right) x^{(i)}$

Generative learning

Discriminative and generative learning algorithms

- Discriminative learning algorithms
  - Try to learn $p(y|x)$
- Generative learning algorithms
  - Try to learn $p(x|y)$ and also $p(y)$

  - Example
  - $p(x|y = 1)$ models the distribution of elephants' features
  - $p(x|y = 0)$ models the distribution of dogs' features

Gaussian discriminant analysis

- Assume that $p(x|y)$ is distributed according to a multivariate Gaussian distribution

Multivariate Gaussian distribution

- d-dimension
- Mean vector $\mu \in \mathbb{R}^d$
- Covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ (symmetric, positive semi-definite)

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right).$$

  - $|\Sigma|$ denotes the determinant of the matrix Σ
- Expectation and covariance

$$\mathrm{E}[X] = \int_x x\, p(x; \mu, \Sigma) dx = \mu \qquad \mathrm{E}[(Z - \mathrm{E}[Z])(Z - \mathrm{E}[Z])^T]$$

The GDA model

- Model $p(x|y)$ using a multivariate normal distribution

$$\begin{aligned} y &\sim \text{Bernoulli}(\phi) \\ x|y=0 &\sim \mathcal{N}(\mu_0, \Sigma) \\ x|y=1 &\sim \mathcal{N}(\mu_1, \Sigma) \end{aligned}$$

- Distribution parameters

$$\begin{aligned} p(y) &= \phi^y(1-\phi)^{1-y} \\ p(x|y=0) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma^{-1}(x-\mu_0)\right) \\ p(x|y=1) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right) \end{aligned}$$

How to estimate the parameters?

- The parameters are φ, Σ, μ_0 and μ_1 (Usually assume common Σ)
- The log-likelihood function for the joint distribution

$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)} | y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi).\end{aligned}$$

Maximum likelihood

- Maximum likelihood yields the result (see the offline derivation)

$$
\begin{aligned}
\phi &= \frac{1}{n}\sum_{i=1}^{n} 1\{y^{(i)} = 1\} \\
\mu_0 &= \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}} \\
\mu_1 &= \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}} \\
\Sigma &= \frac{1}{n}\sum_{i=1}^{n} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T.
\end{aligned}
$$

Kernel Methods

LMS with high-dimensional features: Disadvantages

- Computationally expensive
- let $\phi(x)$ be the vector that contains all the monomials of x with degree ≤ 3
  - Dimension of $\phi(x)$: d^3
  - When $d = 1000, 10^9$
- *Can we avoid this?*

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1^2 \\ x_1 x_2 \\ x_1 x_3 \\ \vdots \\ x_2 x_1 \\ \vdots \\ x_1^3 \\ x_1^2 x_2 \\ \vdots \end{bmatrix}$$

Any great form of θ?

- With the GD, θ can be represented as a linear combination of the vectors $\phi(x)$
- By induction
  - At step 0, initialize $\theta = 0 = \sum_i 0 \cdot \phi(x^{(i)})$
  - Suppose some step, $\theta = \sum_i \beta_i \cdot \phi(x^{(i)})$
  - Then in the next step

$$\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T \phi(x^{(i)})\right) \phi(x^{(i)})$$

$$= \sum_{i=1}^{n} \beta_i \phi(x^{(i)}) + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T \phi(x^{(i)})\right) \phi(x^{(i)})$$

$$= \sum_{i=1}^{n} \underbrace{\left(\beta_i + \alpha \left(y^{(i)} - \theta^T \phi(x^{(i)})\right)\right)}_{\text{new } \beta_i} \phi(x^{(i)})$$

Idea: represent θ by β

- Derive the update rule of β

$$\beta_i := \beta_i + \alpha \left(y^{(i)} - \theta^T \phi(x^{(i)}) \right)$$

$$\theta = \sum_{j=1}^{n} \beta_j \phi(x^{(j)})$$

$$\beta_i := \beta_i + \alpha \left(y^{(i)} - \sum_{j=1}^{n} \beta_j \phi(x^{(j)})^T \phi(x^{(i)}) \right)$$

  - Denote the inner product of the two feature vectors as $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$

Can we accelerate computation?

- At each iteration, we need to compute $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle, \forall j, i \in [n]$

- Acceleration
  - 1. It does not depend on iteration, we can compute it once before starts
  - 2. Computing the inner product does not necessarily require computing $\phi(x^{(i)})$ (see the next page)

Computing $\langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle$

$$\begin{aligned}\langle \phi(x), \phi(z) \rangle &= 1 + \sum_{i=1}^{d} x_i z_i + \sum_{i,j \in \{1,\dots,d\}} x_i x_j z_i z_j + \sum_{i,j,k \in \{1,\dots,d\}} x_i x_j x_k z_i z_j z_k \\ &= 1 + \sum_{i=1}^{d} x_i z_i + \left(\sum_{i=1}^{d} x_i z_i \right)^2 + \left(\sum_{i=1}^{d} x_i z_i \right)^3 \\ &= 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 \end{aligned} \tag{5.9}$$

- Above all, the computation only requires $O(d)$

The final algorithm

- Update β

1. Compute all the values $K(x^{(i)}, x^{(j)}) \triangleq \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ using equation (5.9) for all $i, j \in \{1, \ldots, n\}$. Set $\beta := 0$.

2. **Loop:**

$$\forall i \in \{1, \ldots, n\}, \beta_i := \beta_i + \alpha \left(y^{(i)} - \sum_{j=1}^{n} \beta_j K(x^{(i)}, x^{(j)}) \right) \tag{5.11}$$

Or in vector notation, letting K be the $n \times n$ matrix with $K_{ij} = K(x^{(i)}, x^{(j)})$, we have

$$\beta := \beta + \alpha(\vec{y} - K\beta)$$

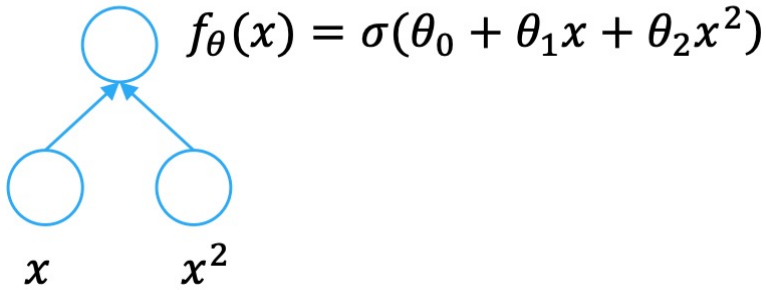
- Compute the prediction

$$\theta^T \phi(x) = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})^T \phi(x) = \sum_{i=1}^{n} \beta_i K(x^{(i)}, x)$$

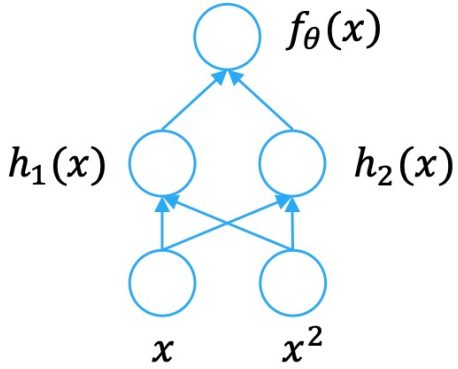
Deep Learning

Computation

- Single-layer function
  - $f_\theta(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$



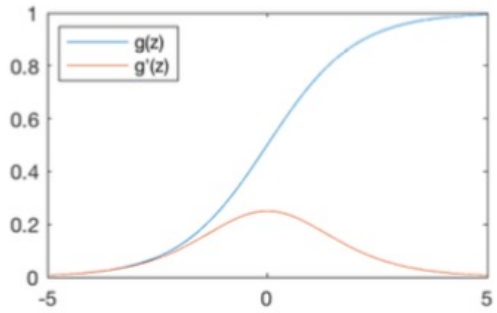
- Multi-layer function
  - $h_1(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$
  - $h_2(x) = \sigma(\theta_3 + \theta_4 x_1 + \theta_5 x_2)$
  - $f_\theta(x) = \sigma(\theta_6 + \theta_7 h_1 + \theta_8 h_2)$



Non-linear activation functions

- Adding non-linearity allows the network to learn and represent complex patterns in the data
- Common non-linear activation functions

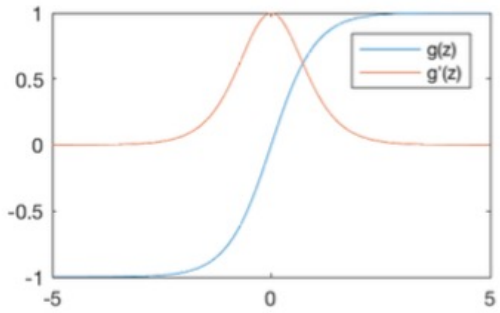
Sigmoid Function



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

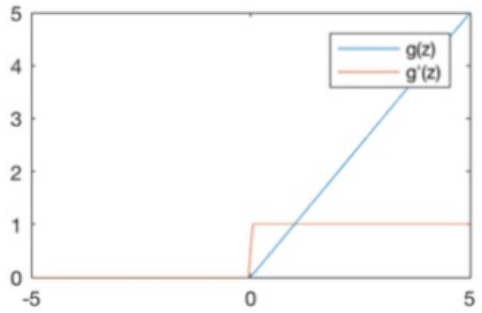
Hyperbolic Tangent



$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\sigma'(z) = 1 - \sigma(z)^2$$

Rectified Linear Unit (ReLU)



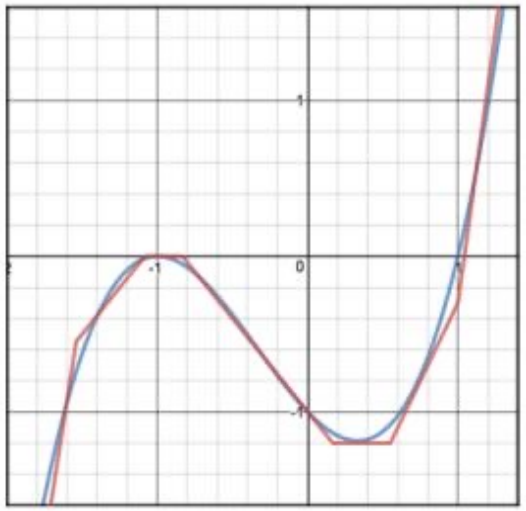
$$\sigma(z) = \max(0, z)$$

$$\sigma'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

[source: MIT 6.S191 introtodeeplearning.com]

Universal approximation theorem

- Theorem (Universal Function Approximators). A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.



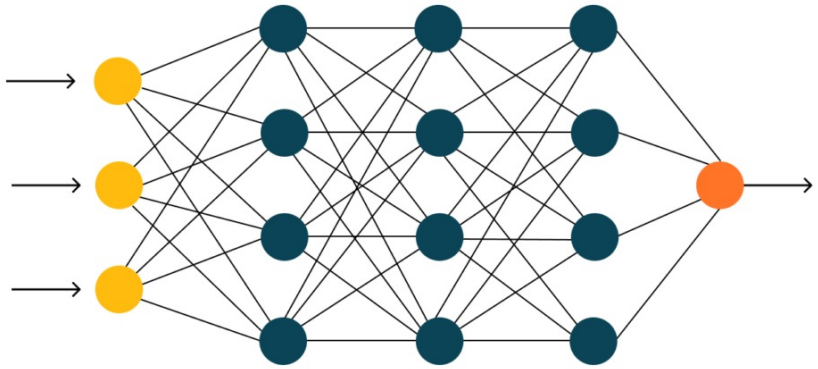
$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

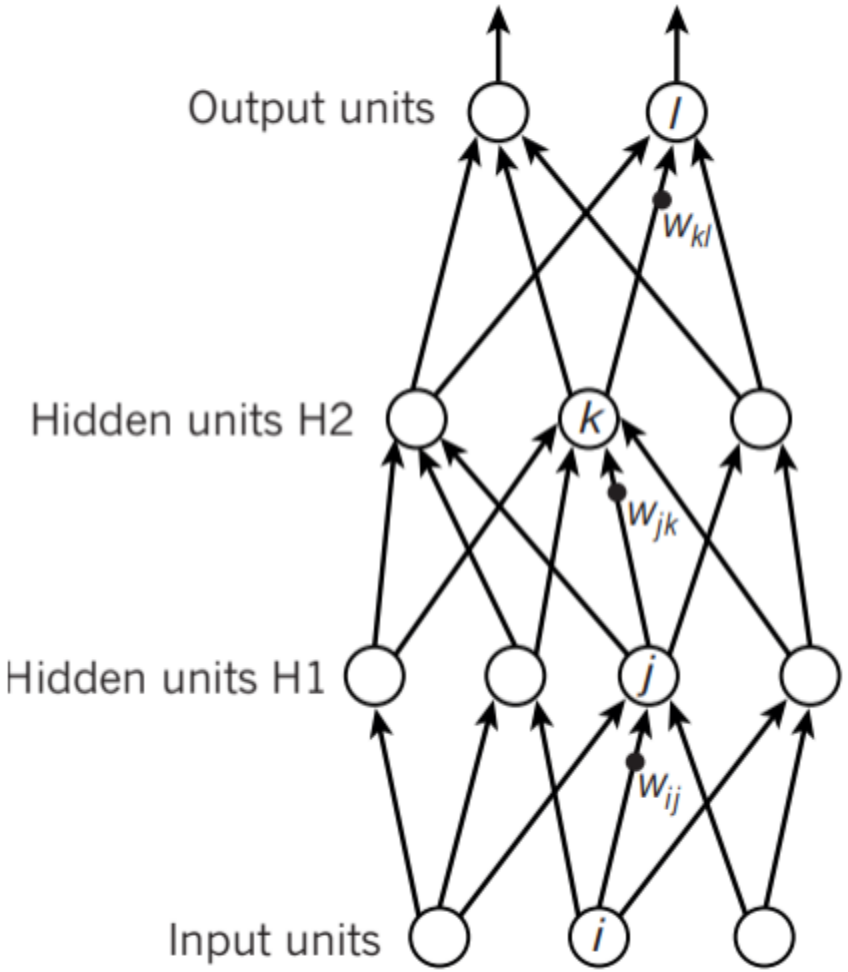
Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." *Neural networks* 2.5 (1989): 359-366

Connection to the kernel methods

- Kernel methods
  - Design the non-linear feature map function
  - The performance significantly depends on the choice of feature map
  - Feature engineering: process of choosing the feature maps
- Neural network
  - Automatically learn the right feature map
  - Requires often less feature engineering



Feed forward vs. Backpropagation



$$y_l = f(z_l)$$

$$z_l = \sum_{k \,\varepsilon\, \text{H2}} w_{kl}\, y_k$$

$$y_k = f(z_k)$$

$$z_k = \sum_{j \,\varepsilon\, \text{H1}} w_{jk}\, y_j$$

$$y_j = f(z_j)$$

$$z_j = \sum_{i \,\varepsilon\, \text{Input}} w_{ij}\, x_i$$

Compare outputs with correct answer to get error derivatives

$$\frac{\partial E}{\partial y_l} = y_l - t_l$$

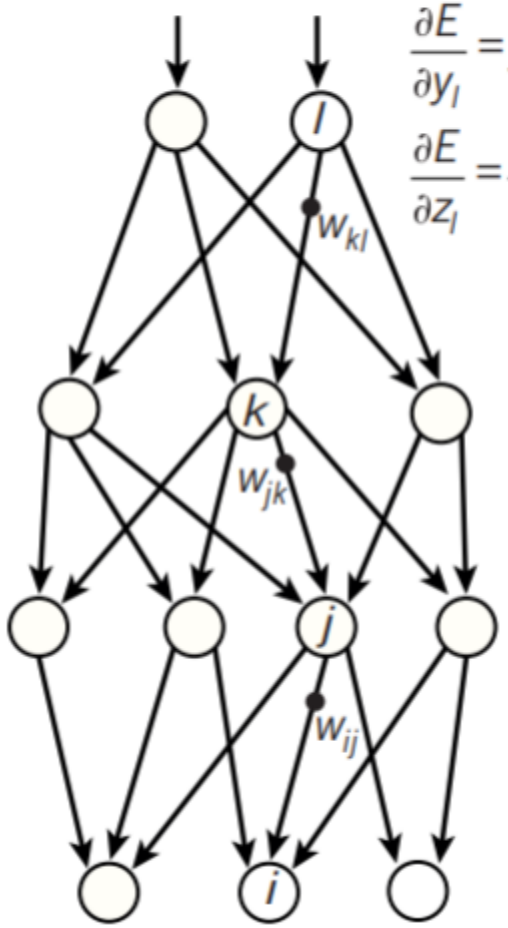
$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$

$$\frac{\partial E}{\partial y_k} = \sum_{l \,\varepsilon\, \text{out}} w_{kl}\frac{\partial E}{\partial z_l}$$

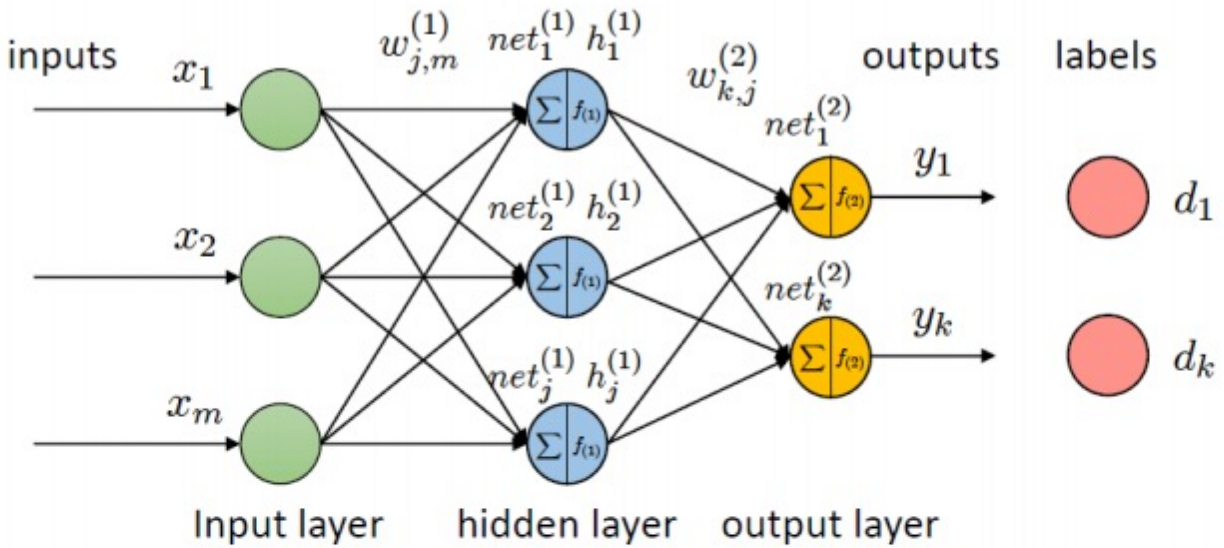
$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

$$\frac{\partial E}{\partial y_j} = \sum_{k \,\varepsilon\, \text{H2}} w_{jk}\frac{\partial E}{\partial z_k}$$

$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$



Make a prediction



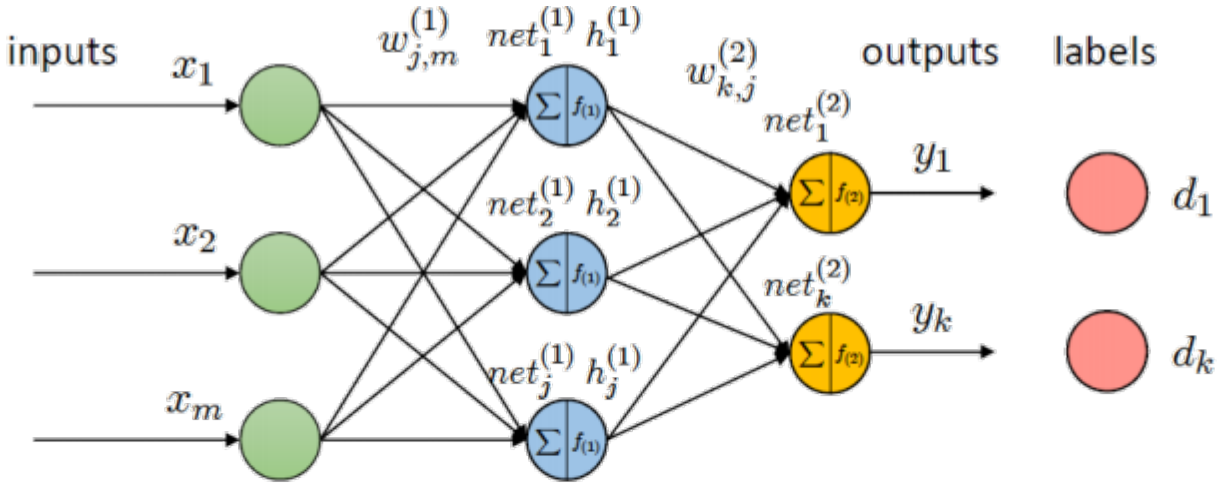
Two-layer feedforward neural network

Feed-forward prediction:

$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \longrightarrow y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(1)} h_j^{(1)}\right)$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\qquad$ $net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Backpropagation



Two-layer feedforward neural network

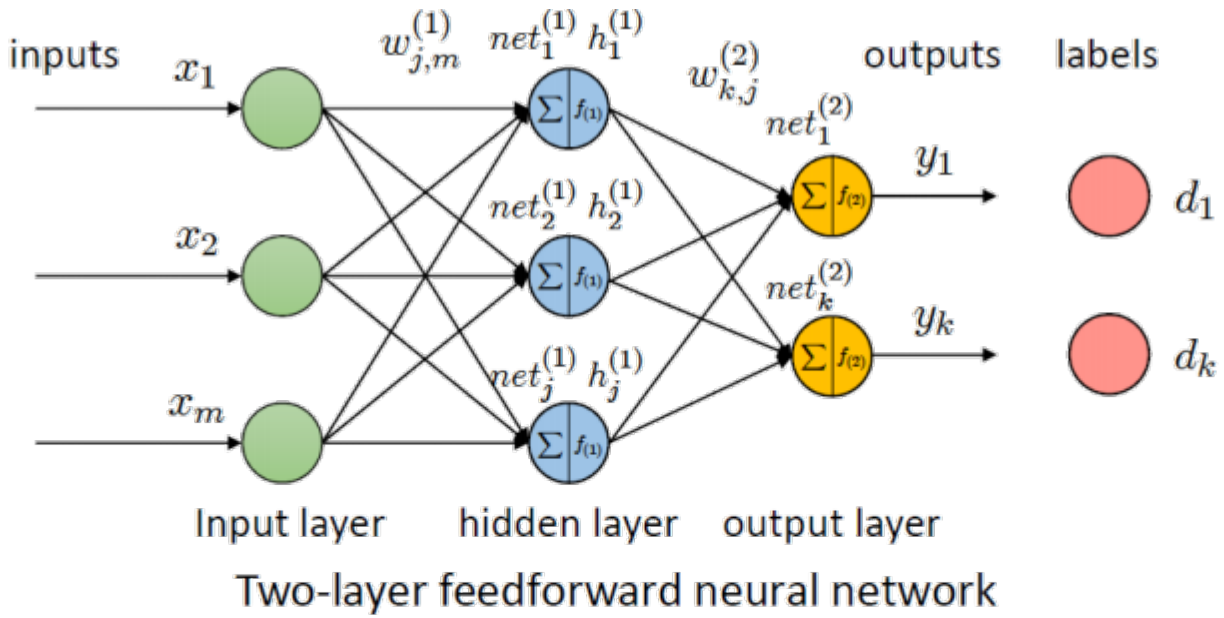
- Assume all the activation functions are sigmoid
- Error function $E = \frac{1}{2}\sum_k (y_k - d_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - d_k$
- $\frac{\partial y_k}{\partial w_{k,j}^{(2)}} = f'_{(2)}(net_k^{(2)})h_j^{(1)} = y_k(1-y_k)h_j^{(1)}$
- $\Rightarrow \frac{\partial E}{\partial w_{k,j}^{(2)}} = (y_k - d_k)y_k(1-y_k)h_j^{(1)}$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} - \eta(y_k - d_k)y_k(1-y_k)h_j^{(1)}$

Feed-forward prediction:

$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)} x_m\right) \longrightarrow y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(2)} h_j^{(1)}\right) \longrightarrow y_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $\quad net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$

Backpropagation (cont.)



Two-layer feedforward neural network

- Error function $E = \frac{1}{2}\sum_k (y_k - d_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - d_k$
- $\frac{\partial y_k}{\partial h_j^{(1)}} = y_k(1-y_k)w_{k,j}^{(2)}$
- $\frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = f'_{(1)}\left(net_j^{(1)}\right)x_m = h_j^{(1)}\left(1-h_j^{(1)}\right)x_m$
- $\Rightarrow \frac{\partial E}{\partial w_{j,m}^{(1)}} = h_j^{(1)}\left(1-h_j^{(1)}\right)\sum_k w_{k,j}^{(2)}(y_k - d_k)y_k(1-y_k)x_m$
- $\Rightarrow w_{j,m}^{(1)} \leftarrow w_{j,m}^{(1)} - \eta h_j^{(1)}\left(1-h_j^{(1)}\right)\sum_k w_{k,j}^{(2)}(y_k - d_k)y_k(1-y_k)x_m$

Feed-forward prediction:

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}\left(\sum_m w_{j,m}^{(1)}x_m\right) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}\left(\sum_j w_{k,j}^{(1)}h_j^{(1)}\right)$$

$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

where $\quad net_j^{(1)} = \sum_m w_{j,m}^{(1)}x_m \qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)}h_j^{(1)}$

Generalization

Intuition

- Recall in previous classes
  - We typically learn a model h_θ by minimizing the training loss/error
  - $J_\theta = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$
  - This is not the ultimate goal
- The ultimate goal
  - Sample a test data from the test distribution $\mathcal{D}$
  - Measure the model's error on the test data (test loss/error)

$$L(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[(y - h_\theta(x))^2]$$

  - Can be approximated by the average error on many sampled test examples

Challenges

- The test examples are unseen
  - Even though the training set is sampled from the same distribution $\mathcal{D}$, it can not guaranteed that the test error is close to the training error
  - Minimizing training error may not lead to a small test error
- Important concepts
  - Overfitting: the model predicts accurately on the training dataset but doesn't generalize well to other test examples
  - Underfitting: the training error is relatively large (typically the test error is also relatively large)

- How the test error is influenced by the learning procedure, especially the choice of model parameterizations?

How about fitting a linear model?

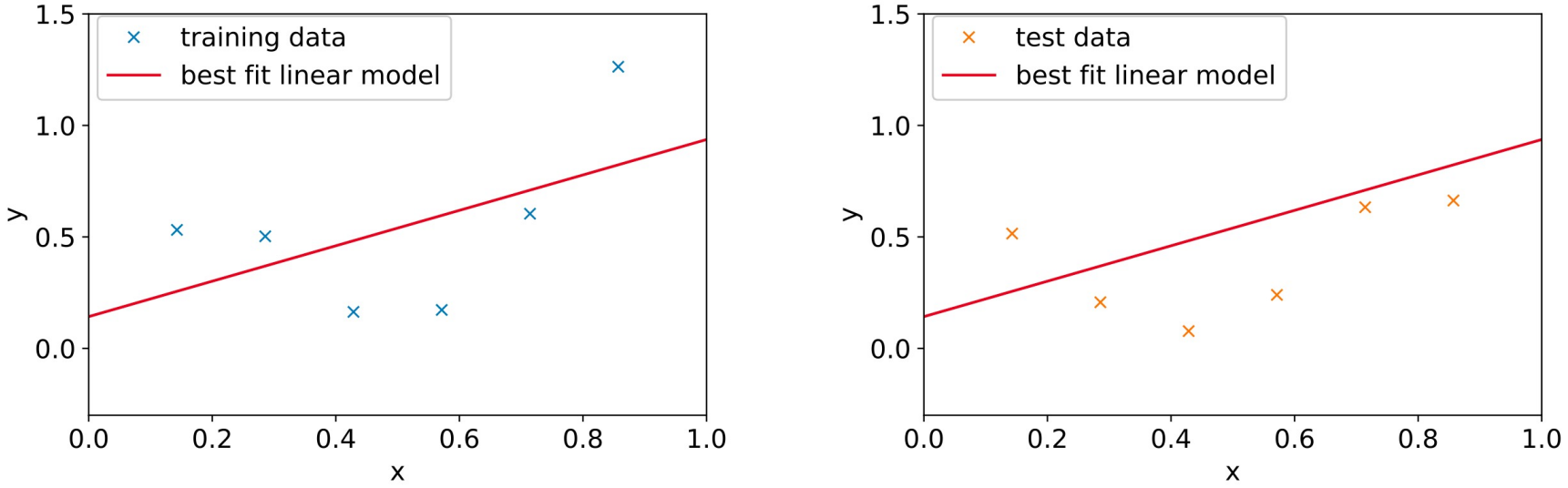


Figure 8.2: The best fit linear model has large training and test errors.

- The true relationship between y and x is not linear
- Any linear model is far away from the true function
- The training error is large, underfitting

How about fitting a linear model? (cont'd)

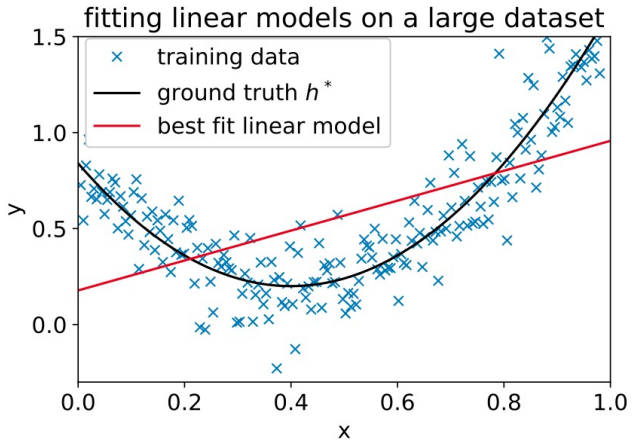


Figure 8.3: The best fit linear model on a much larger dataset still has a large training error.

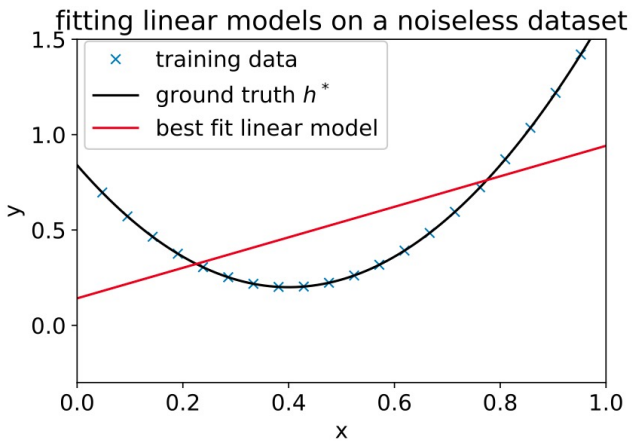


Figure 8.4: The best fit linear model on a noiseless dataset also has a large training/test error.

- Fundamental bottleneck: linear model family's inability to capture the structure in the data
- Define model bias: the test error even if we were to fit it to a very (say, infinitely) large training dataset

How about a 5th-degree polynomial?

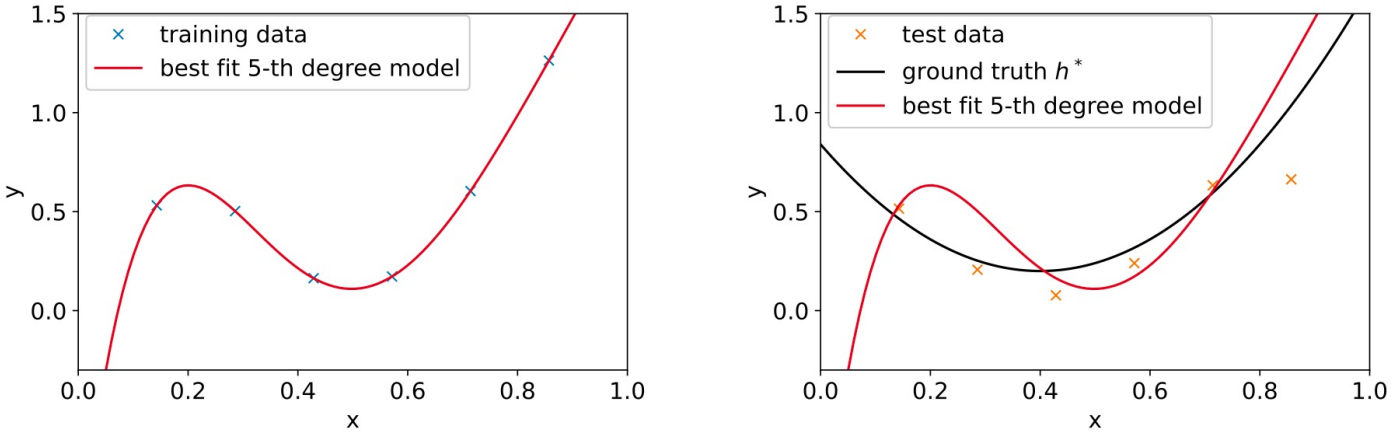
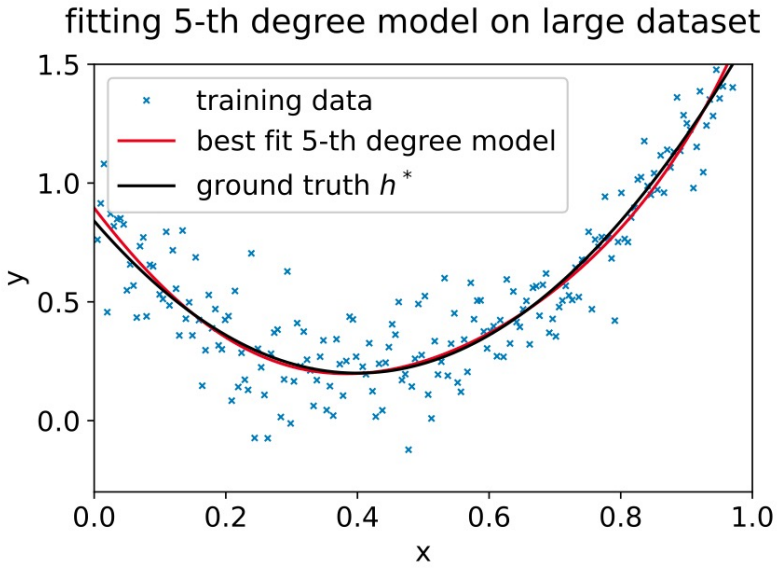


Figure 8.5: Best fit 5-th degree polynomial has zero training error, but still has a large test error and does not recover the the ground truth. This is a classic situation of overfitting.

- Predict well on the training set, does not work well on test examples

How about a 5th-degree polynomial? (cont'd)



- When the training set becomes huge, the model recovers the ground-truth

How about a 5th-degree polynomial? (cont'd)

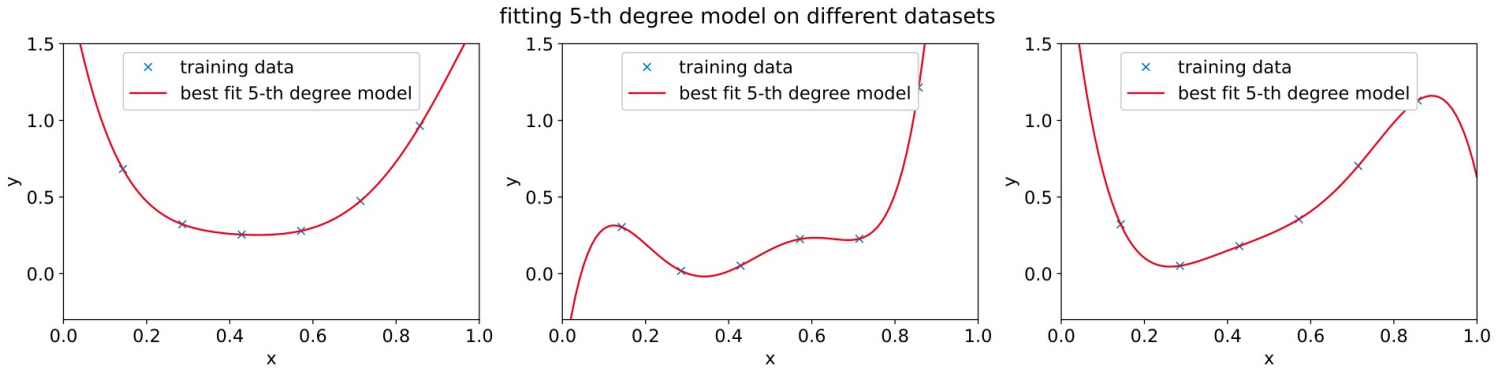


Figure 8.7: The best fit 5-th degree models on three different datasets generated from the same distribution behave quite differently, suggesting the existence of a large variance.

- Failure: fitting patterns in the data that happened to be present in the small, finite training set (NOT the real relationship between x and y)
- Define variance: the amount of variations across models learnt on multiple different training datasets (drawn from the same underlying distribution)

Bias-variance trade-off

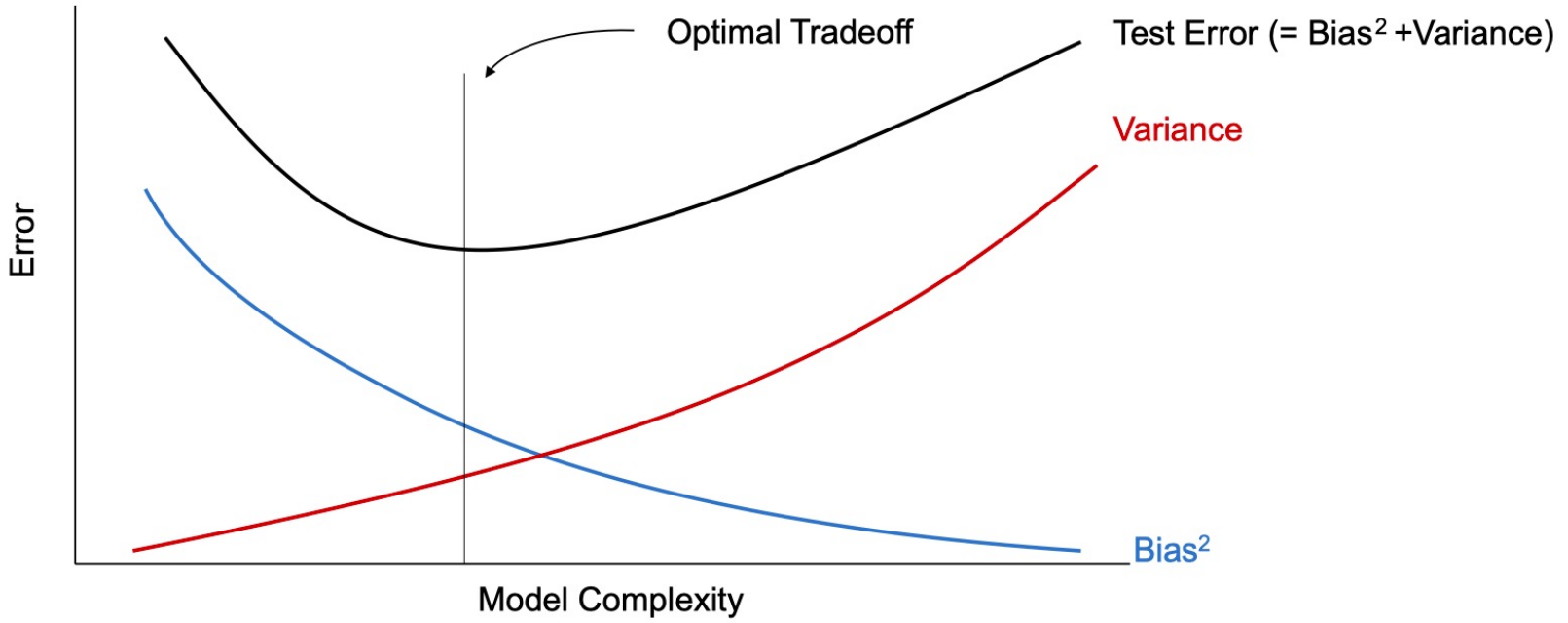


Figure 8.8: An illustration of the typical bias-variance tradeoff.

Problem setting: regression

- Draw a training dataset $S = \{x^{(i)}, y^{(i)}\}_{i=1}^{n}$ such that $y^{(i)} = h^{\star}(x^{(i)}) + \xi^{(i)}$ where $\xi^{(i)} \in N(0, \sigma^2)$.
- Train a model on the dataset S, denoted by $\hat{h}_S$.
- Take a test example (x, y) such that $y = h^{\star}(x) + \xi$ where $\xi \sim N(0, \sigma^2)$, and measure the expected test error (averaged over the random draw of the training set S and the randomness of ξ)

$$\mathrm{MSE}(x) = \mathbb{E}_{S,\xi}[(y - \hat{h}_S(x))^2] \tag{8.2}$$

Decomposition

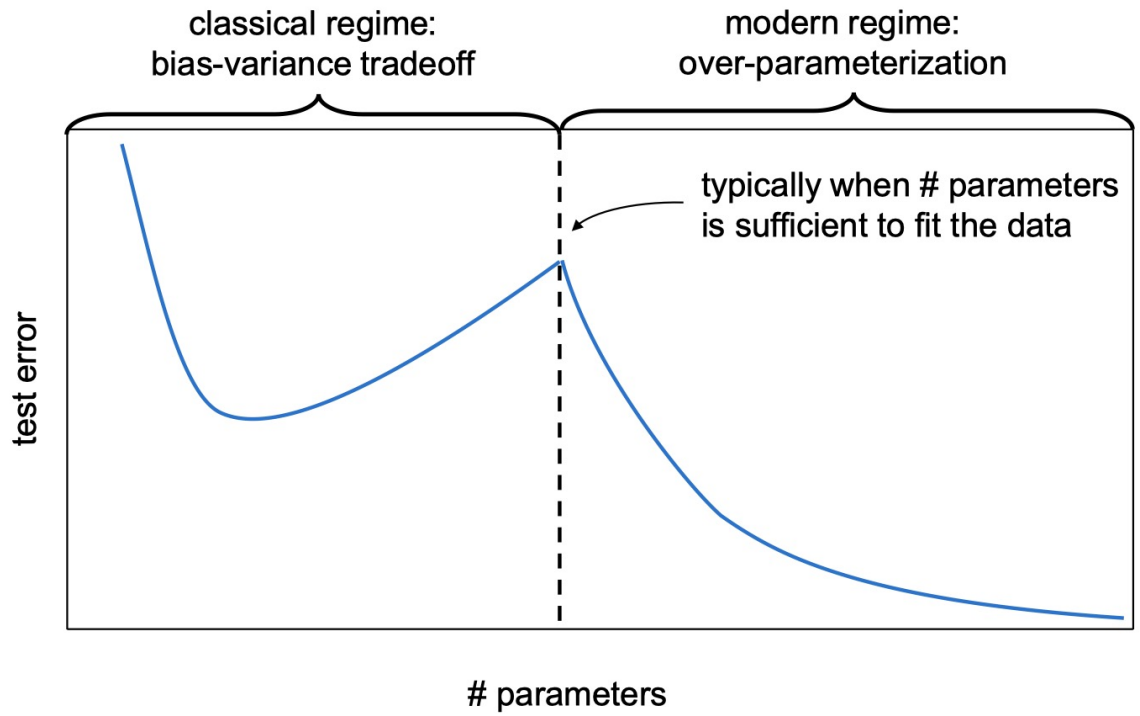
- $$\begin{aligned}\mathrm{MSE}(x) = \mathbb{E}[(y - h_S(x))^2] &= \mathbb{E}[(\xi + (h^\star(x) - h_S(x)))^2] \\ &= \mathbb{E}[\xi^2] + \mathbb{E}[(h^\star(x) - h_S(x))^2] \\ &= \sigma^2 + \mathbb{E}[(h^\star(x) - h_S(x))^2]\end{aligned}$$

- Define $h_{avg}(x) = \mathbb{E}_S[(h_S(x))]$
  - The model obtained by drawing an infinite number of datasets, training on them, and averaging their predictions on x

- $$\begin{aligned}\mathrm{MSE}(x) &= \sigma^2 + \mathbb{E}[(h^\star(x) - h_S(x))^2] \\ &= \sigma^2 + (h^\star(x) - h_{\mathrm{avg}}(x))^2 + \mathbb{E}[(h_{\mathrm{avg}} - h_S(x))^2] \\ &= \underbrace{\sigma^2}_{\text{unavoidable}} + \underbrace{(h^\star(x) - h_{\mathrm{avg}}(x))^2}_{\triangleq \text{ bias}^2} + \underbrace{\mathrm{var}(h_S(x))}_{\triangleq \text{ variance}}\end{aligned}$$

Model-wise double descent

- Recent works demonstrated that the test error can present a "double descent" phenomenon in a range of machine learning models including linear models and deep neural networks



Sample complexity bounds

Problem setting

- To simplify, consider the classification problem with $y \in \{0,1\}$
- Training set $S = \{(x^i, y^i); i = 1,2, \dots, n\}$, drawn iid from $\mathcal{D}$

- For hypothesis h, define training error (empirical risk/error)

$$\hat{\varepsilon}(h) = \frac{1}{n}\sum_{i=1}^{n} 1\{h(x^{(i)}) \neq y^{(i)}\}$$

- Define the generalization error $\varepsilon(h) = P_{(x,y)\sim\mathcal{D}}(h(x) \neq y)$

One of PAC assumption: training and testing set are from the same D

Theorem of generalization error

- **Theorem.** Let $|\mathcal{H}| = k$, and let any n, δ be fixed. Then with probability at least $1 - \delta$, we have that

$$\varepsilon(\hat{h}) \leq \left(\min_{h \in \mathcal{H}} \varepsilon(h) \right) + 2\sqrt{\frac{1}{2n} \log \frac{2k}{\delta}}.$$

- Explanation of bias/variance
  - If we switch to a larger function class $\mathcal{H}' \supseteq \mathcal{H}$
  - The first term decreases: lower bias
  - The second term increases as k increases: higher variance

Regularization

Regularization

- Meaning of regularization
  - Adding an additional term to control the model complexity and prevent overfitting

$$J_\lambda(\theta) = J(\theta) + \lambda R(\theta)$$

  - $J(\theta)$: the original loss, e.g., MSE
  - $R(\theta)$: the regularizer, typically non-negative
  - $\lambda \geq 0$: regularization parameter

Model selection

Solution 1: Select the one with the minimum training loss?

- Given the training set S

1. Train each model M_i on S, to get some hypothesis h_i.
2. Pick the hypotheses with the smallest training error.

- What's the problem?
  - Lower training error prefers complex models
  - These models usually overfits

Solution 2: Hold-out cross validation

- Split the training set S
  - $S = S_{train}(\text{usually } 70\%) + S_{cv}\ (\text{usually } 30\%)$
  - Train each model M_i on S_{train} only, to get some hypothesis h_i
  - Evaluate h_i on S_{cv}, denote the error as $\hat{\varepsilon}_{S_{\text{cv}}}(h_i)$ (validation error)
  - Pick the hypothesis with the smallest validation error

- The CV set plays the role of testing set
- Evaluate the model in terms of approximate generalization error
- Avoid overfitting

Improvement: k-fold cross validation

1. Randomly split S into k disjoint subsets of m/k training examples each. Lets call these subsets $S_1, \ldots, S_k$.

2. For each model M_i, we evaluate it as follows:

   For $j = 1, \ldots, k$

   Train the model M_i on $S_1 \cup \cdots \cup S_{j-1} \cup S_{j+1} \cup \cdots S_k$ (i.e., train on all the data except S_j) to get some hypothesis h_{ij}.

   Test the hypothesis h_{ij} on S_j, to get $\hat{\varepsilon}_{S_j}(h_{ij})$.

   The estimated generalization error of model M_i is then calculated as the average of the $\hat{\varepsilon}_{S_j}(h_{ij})$'s (averaged over j).

3. Pick the model M_i with the lowest estimated generalization error, and retrain that model on the entire training set S. The resulting hypothesis is then output as our final answer.

- Typical choice: k=10

Frequentist V.S. Bayesian

- Consider θ as the model parameter
- Frequentist view
  - θ is constant-valued but unknown
  - We need to estimate this parameter, such as MLE

$$\theta_{\mathrm{MLE}} = \arg\max_{\theta} \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta).$$

- Bayesian review
  - θ is a random variable with unknown value
  - We can specify a prior distribution $p(\theta)$ on θ that expresses our "prior beliefs" about the parameters

Bayesian view

- Given a training set $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$
- Compute the posterior of θ

$$
\begin{aligned}
p(\theta|S) &= \frac{p(S|\theta)p(\theta)}{p(S)} \\
&= \frac{\left(\prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta)\right) p(\theta)}{\int_{\theta} \left(\prod_{i=1}^{n} p(y^{(i)}|x^{(i)}, \theta)p(\theta)\right) d\theta}
\end{aligned}
$$

- To predict the label of a new data x

$$
p(y|x, S) = \int_{\theta} p(y|x, \theta)p(\theta|S)d\theta \qquad \mathrm{E}[y|x, S] = \int_{y} yp(y|x, S)dy
$$

Maximum a posteriori (MAP)

- Approximate the posterior distribution for θ
- Use single point estimate

$$p(\theta|S) = \frac{p(S|\theta)p(\theta)}{p(S)} = \frac{\left(\prod_{i=1}^{n} p(y^{(i)}|x^{(i)},\theta)\right) p(\theta)}{\int_\theta \left(\prod_{i=1}^{n} p(y^{(i)}|x^{(i)},\theta)p(\theta)\right) d\theta}$$

$$\theta_{\text{MAP}} = \arg\max_\theta \prod_{i=1}^{n} p(y^{(i)}|x^{(i)},\theta)p(\theta)$$

Additional term compared with MLE

- The prior $p(\theta)$ is usually assumed to be $\theta \sim \mathcal{N}(0, \tau^2 I)$
- Parameters with smaller norm are more preferred than MLE
- Less susceptible to overfitting

Unsupervised Learning

K-means

The k-means clustering algorithm

- 1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^d$ randomly.

  2. Repeat until convergence: {

  For every i, set

  $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

  For each j, set

  $$\mu_j := \frac{\sum_{i=1}^{n} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{n} 1\{c^{(i)} = j\}}.$$

  }

Convergence analysis (cont'd)

- Define the distortion function

$$J(c, \mu) = \sum_{i=1}^{n} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

- K-means is exactly coordinate descent on J
- J must monotonically decrease, and the value of J must converge

EM

Intuition

- Recall that in unsupervised learning, we are given the training set without labels

$$\{x^{(1)}, \ldots, x^{(n)}\}$$

- We can assume these data are from different underlying classes $j = 1,2, \ldots, k$
- Each class is modeled by a Gaussian $\mathcal{N}(\mu_j, \Sigma_j)$
- The class label follows a multinomial distribution
  - Each data can only belong to one of these classes
  - Distribution parameter ϕ with $\phi_j \geq 0$ and $\sum_j \phi_j = 1$

Mixture of gaussian models

- Each data x^i corresponds to a **(latent)** class label z^i
- $z^i \sim$ Multinomial(ϕ), with $\phi_j \geq 0$ and $\sum_j \phi_j = 1$
  - $\mathbb{P}(z^i = j) = \phi_j$
- $x^i \mid z^i = j \sim \mathcal{N}(\mu_j, \Sigma_j)$

Maximum likelihood

- Log-likelihood

$$\begin{aligned}\ell(\phi, \mu, \Sigma) &= \sum_{i=1}^{n} \log p(x^{(i)}; \phi, \mu, \Sigma) \\ &= \sum_{i=1}^{n} \log \sum_{z^{(i)}=1}^{k} p(x^{(i)}|z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi)\end{aligned}$$

- Zero the derivatives of this formula, but challenging to find the closed-form solution

Relaxation: If we know the class label

- The log-likelihood becomes

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{n} \log p(x^{(i)}|z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi)$$

How to estimate the parameters?

- The parameters are φ, Σ, μ_0 and μ_1 (Usually assume common Σ)
- The log-likelihood function for the joint distribution

$$\begin{aligned} \ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)}|y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi). \end{aligned}$$

Iterative algorithm to update z^i

- Repeat until converge
  - Guess the value of z^i: compute the posterior probability

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

  - Based on z^i, use maximum likelihood to estimate parameters

Iterative algorithm to update z^i

- Repeat until converge
  - Guess the value of z^i: compute the posterior probability
  - Based on z^i, use maximum likelihood to estimate parameters

$$\phi_j := \frac{1}{n}\sum_{i=1}^{n} w_j^{(i)},$$

$$\mu_j := \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)}}$$

Comparison with existing forms?

Expectation-Maximization

- Repeat until converge
  - Guess the value of z^i: compute the posterior probability

Step E

  - Based on z^i, use maximum likelihood to estimate parameters

Step M

General EM: Setting

- Recall we have the training set $\{x^{(1)}, \ldots, x^{(n)}\}$
- We have a latent variable model $p(x, z; \theta)$

- Hope to maximize the likelihood

$$\ell(\theta) = \sum_{i=1}^{n} \log p(x^{(i)}; \theta)$$

$$= \sum_{i=1}^{n} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)$$

$$p(x; \theta) = \sum_{z} p(x, z; \theta)$$

Intuition

- Directly optimizing the likelihood is infeasible

- How about optimizing the **lower bound** of the likelihood?
  - Construct a lower bound – Step E
  - Optimizing the lower bound – Step M

Lower bound of the likelihood

- Hope to derive the **lower bound** for

$$\log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

-

$$\begin{aligned}
\log p(x;\theta) &= \log \sum_z p(x,z;\theta) \\
&= \log \sum_z Q(z) \frac{p(x,z;\theta)}{Q(z)} \\
&\geq \sum_z Q(z) \log \frac{p(x,z;\theta)}{Q(z)}
\end{aligned}$$

Q is any distribution on z with $Q(z) \geq 0$ and $\sum_z Q(z) = 1$

Jensen's inequality

Choice of Q (cont'd)

- Hope the inequality hold with equality

How?

- Recall that in the Jensen's inequality, the equality holds when X is a constant
  - To make $\frac{p(x,z;\theta)}{Q(z)}$ be a constant, let $Q(z) \propto p(x,z;\theta)$
  - Since $\sum_z Q(z) = 1$, it follows that

$$\begin{aligned} Q(z) &= \frac{p(x,z;\theta)}{\sum_z p(x,z;\theta)} \\ &= \frac{p(x,z;\theta)}{p(x;\theta)} \\ &= p(z|x;\theta) \end{aligned}$$

Verify the equality with $Q(z) = p(z|x;\theta)$

- $$\sum_z Q(z) \log \frac{p(x,z;\theta)}{Q(z)} = \sum_z p(z|x;\theta) \log \frac{p(x,z;\theta)}{p(z|x;\theta)}$$

Evidence lower bound (ELBO)

$$= \sum_z p(z|x;\theta) \log \frac{p(z|x;\theta)p(x;\theta)}{p(z|x;\theta)}$$

$$= \sum_z p(z|x;\theta) \log p(x;\theta)$$

$$= \log p(x;\theta) \sum_z p(z|x;\theta)$$

$$= \log p(x;\theta) \qquad (\text{because } \textstyle\sum_z p(z|x;\theta) = 1)$$

EM algorithm procedure

- Foundation

$$\forall Q, \theta, x, \quad \log p(x; \theta) \geq \mathrm{ELBO}(x; Q, \theta)$$

- Procedure of EM
  - Setting Q(z) = p(z|x; θ) so that ELBO(x; Q, θ) = log p(x; θ)
  - Maximizing ELBO(x; Q, θ) w.r.t θ while fixing the choice of Q

Formal procedure of EM

- Repeat until convergence {

  (E-step) For each i, set

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)};\theta).$$

  (M-step) Set

$$\theta := \arg\max_\theta \sum_{i=1}^{n} \mathrm{ELBO}(x^{(i)}; Q_i, \theta)$$

$$= \arg\max_\theta \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)};\theta)}{Q_i(z^{(i)})}$$

}

Convergence analysis

- Objective: prove $\ell(\theta^{(t)}) \leq \ell(\theta^{(t+1)})$

- Proof

$$\ell(\theta^{(t+1)}) \geq \sum_{i=1}^{n} \mathrm{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t+1)})$$

Jensen's inequality

$$\geq \sum_{i=1}^{n} \mathrm{ELBO}(x^{(i)}; Q_i^{(t)}, \theta^{(t)})$$

Updating rule

$$= \ell(\theta^{(t)})$$

Selection of Q

EM=alternating maximization on ELBO(Q, θ)

- Define ELBO(Q, θ)

$$\text{ELBO}(Q, \theta) = \sum_{i=1}^{n} \text{ELBO}(x^{(i)}; Q_i, \theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

  - E step: maximizes ELBO(Q, θ) with respect to Q
  - M step: maximizes ELBO(Q, θ) with respect to θ

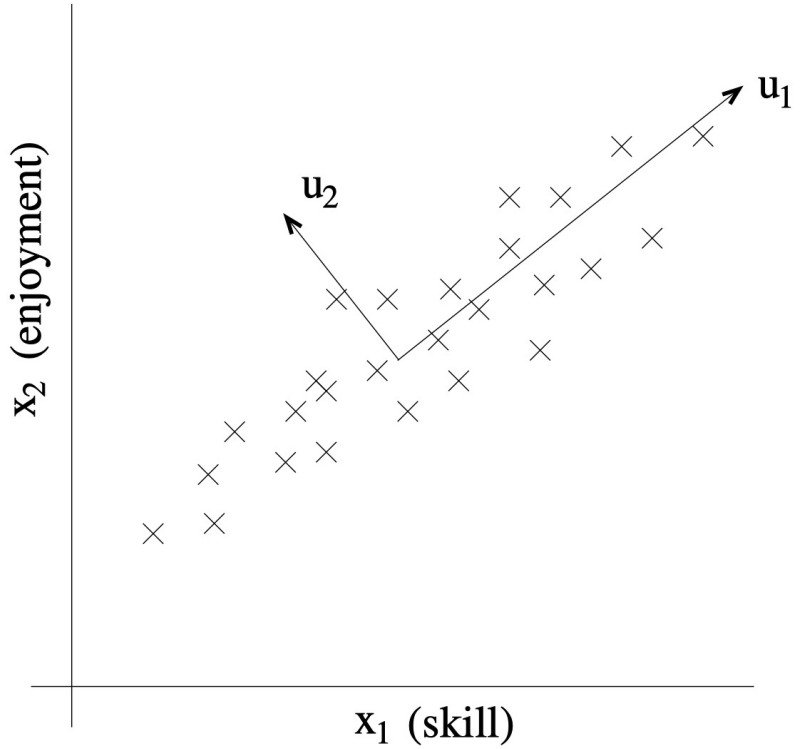
Hint: show that

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x,z;\theta)}{Q(z)}$$

$$= \log p(x) - D_{KL}(Q \| p_{z|x})$$

PCA

Which basis to select?

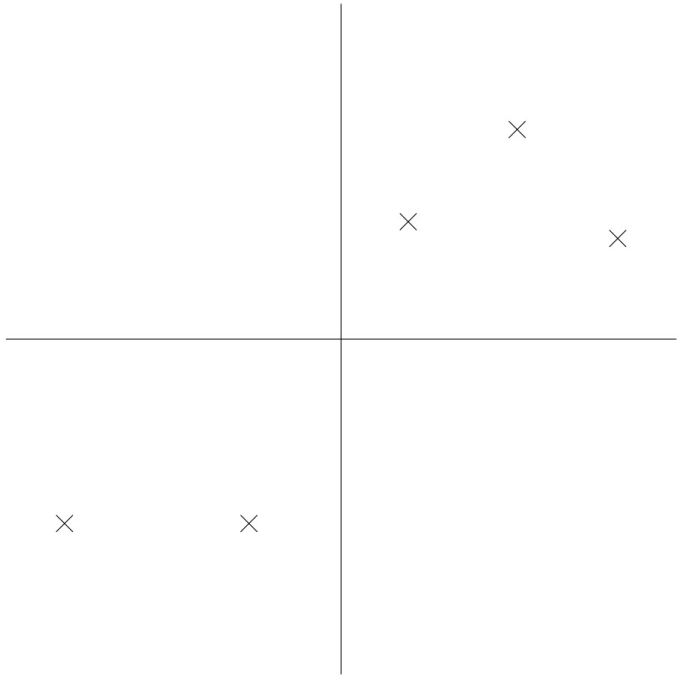


- The direction on which the data approximately lies

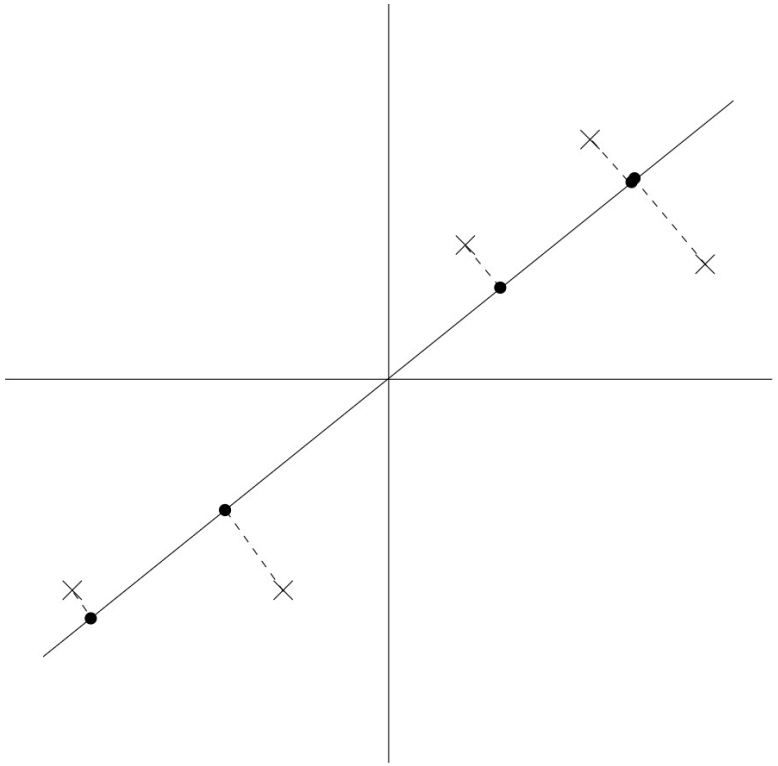
Intuition

- The data has natural "spread" in some directions more than others
- The **major axis** is the direction where data varies the most
- If we project data onto this axis, we **retain the most information** (variance)

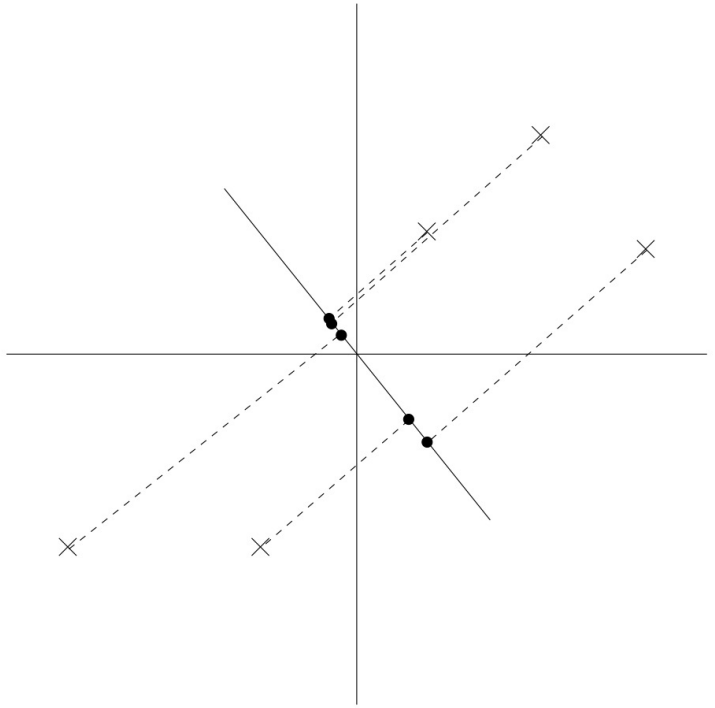
Example



Data



Selection 1



Selection 2

Mathematical Formulation

- The length of the projection of x onto u is $x^{\top}u$
- Maximizing the variance of the projections is equivalent to maximize

$$\frac{1}{n}\sum_{i=1}^{n}(x^{(i)T}u)^2 = \frac{1}{n}\sum_{i=1}^{n}u^T x^{(i)}x^{(i)T}u$$

$$= u^T\left(\frac{1}{n}\sum_{i=1}^{n}x^{(i)}x^{(i)T}\right)u.$$

Solution

- We want to maximize $\mathbf{u}^T \Sigma \mathbf{u}$

  Subject to: $\mathbf{u}^T \mathbf{u} = 1$

  Lagrangian:

$$\mathcal{L}(\mathbf{u}, \lambda) = \mathbf{u}^T \Sigma \mathbf{u} - \lambda(\mathbf{u}^T \mathbf{u} - 1)$$

  Set gradient to 0:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{u}} = 0 \Rightarrow \Sigma \mathbf{u} = \lambda \mathbf{u}$$

- The objective becomes finding the principal eigenvector of Σ

Extension to larger dimension

- If we wish to project our data into a k-dimensional subspace (k < d)
- Choose to be the top k eigenvectors of Σ

- Due to that Σ is symmetric, u_i's will be orthogonal to each other
- u_i's now form a new orthogonal basis for the data

Obtain new, low-dimension features

- Represent the data in the new basis

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in \mathbb{R}^k$$

- PCA is also referred to as a dimensionality reduction algorithm
- The vectors $u_1, \ldots, u_k$ are called the first k principal components of the data

ICA

Motivation

- Consider the cocktail party problem
  - *d* speakers are talking simultaneously in a room
  - Place *d* microphones at different locations
  - Each microphone records a different combination of the speakers' voices
- Can we recover the original speech signals of each speaker?

Problem formulation

- Source $s \in \mathbb{R}^d$
- Observation $x \in \mathbb{R}^d$

- Model the observation and source

$$x = As$$

  - A is the mixing matrix

Problem formulation (cont'd)

- Now we have multiple observations

$$\{x^{(i)}; i = 1, \ldots, n\}$$

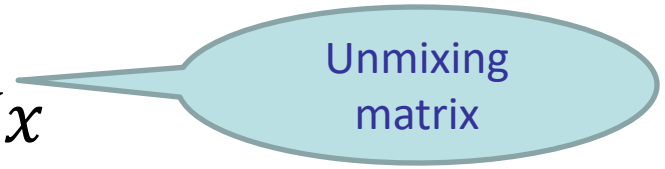
  - The i-the data satisfies $x^{(i)} = As^{(i)}$

- Illustration
  - x_j^i is the acoustic reading recorded by microphone j at time i
  - s_j^i is the sound that speaker j was uttering at time i

Objective

- Given observation x^i, can we recover the sources?

- How?
  - $s = A^{-1}x := Wx$



  - $W = \begin{bmatrix} — & w_1^T & — \\ & \vdots & \\ — & w_d^T & — \end{bmatrix}$ *then* $s_j^i = W_j^\top x^i$

Maximum likelihood

- Construct a joint distribution of the sources

$$p(s) = \prod_{j=1}^{d} p_s(s_j)$$

Imply independence

- Recall that the observation follows $x = As$, $s = A^{-1}x := Wx$

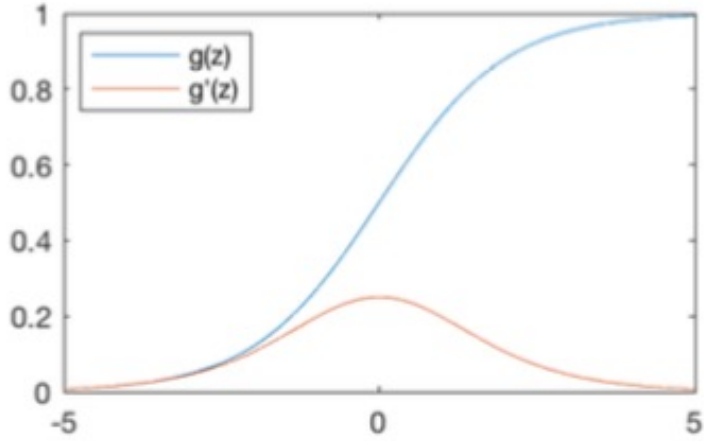
- What's the probability of x?
  - $p_x(x) = p_s(Wx)|W|$? $\Longrightarrow$ $p(x) = \prod_{j=1}^{d} p_s(w_j^T x) \cdot |W|$

How to specify a density for s?
Cannot be gaussian

Selecting Sigmoid

- Log-likelihood becomes

$$\ell(W) = \sum_{i=1}^{n} \left(\sum_{j=1}^{d} \log g'(w_j^T x^{(i)}) + \log |W| \right)$$



$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

- Using stochastic gradient ascent to optimize

Reinforcement Learning

MDP

Optimal Quantities

- The value (utility) of a state s:

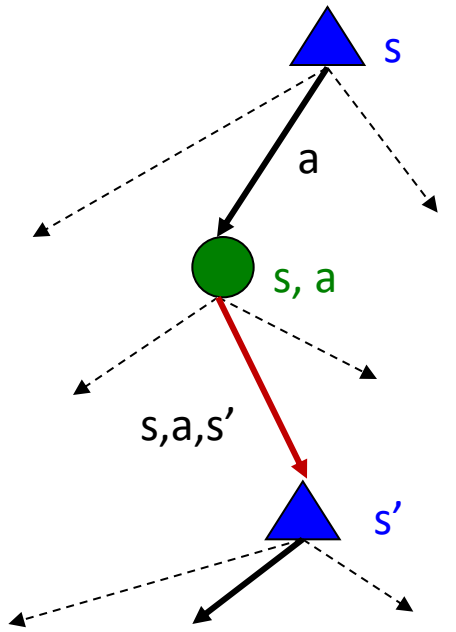
  $V^*(s)$ = expected utility starting in s and acting optimally

- The value (utility) of a q-state (s,a):

  $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

- The optimal policy:

  $\pi^*(s)$ = optimal action from state s



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

[Demo – gridworld values (L8D4)]

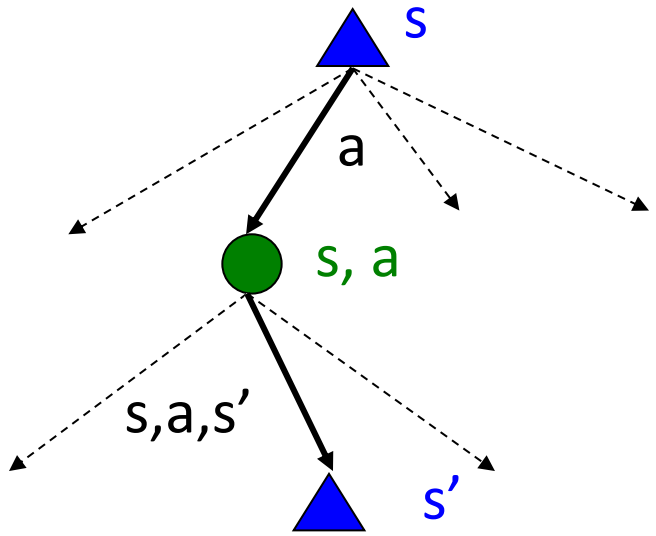
The Bellman Equations

- Definition of "optimal utility" via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right]$$

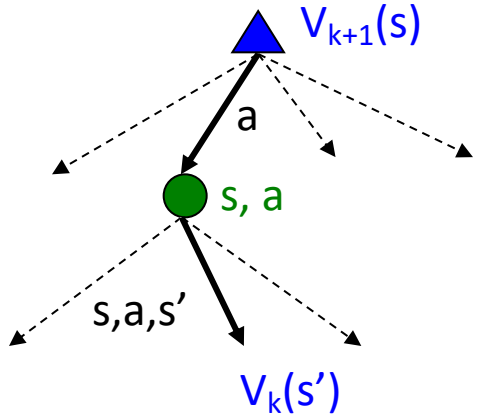


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

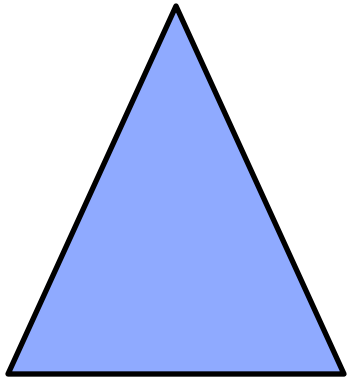
Value Iteration

- Start with $V_0(s) = 0$: no time steps left means an expected reward sum of zero
- Given vector of $V_k(s)$ values, do one ply of expectimax from each state:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$

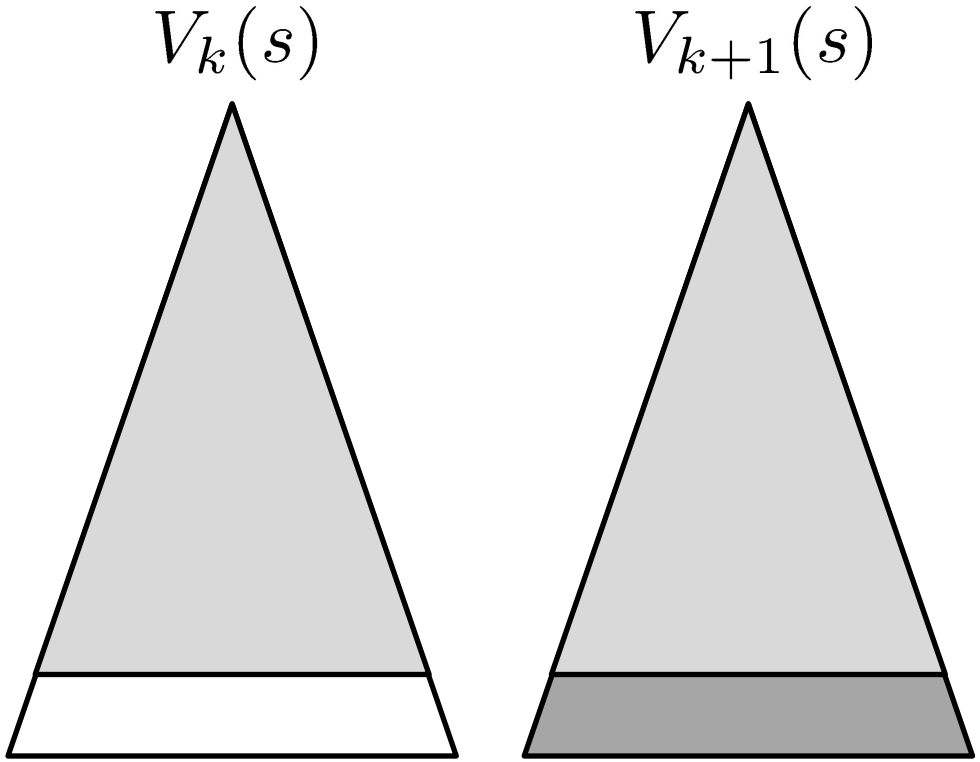


- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



Convergence

- How do we know the V_k vectors are going to converge?
- Case 1: If the tree has maximum depth M, then V_M holds the actual untruncated values
- Case 2: If the discount is less than 1
  - Sketch: For any state V_k and V_{k+1} can be viewed as depth k+1 expectimax results in nearly identical search trees
  - The difference is that on the bottom layer, V_{k+1} has actual rewards while V_k has zeros
  - That last layer is at best all R_{MAX}
  - It is at worst R_{MIN}
  - But everything is discounted by γ^k that far out
  - So V_k and V_{k+1} are at most γ^k max|R| different
  - So as k increases, the values converge



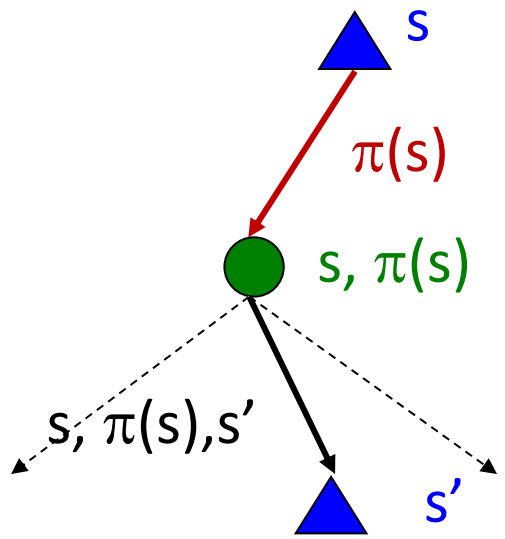
Policy Evaluation

- How do we calculate the V's for a fixed policy π?

- Idea 1: Turn recursive Bellman equations into updates (like value iteration)

$$V_0^\pi(s) = 0$$

$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$

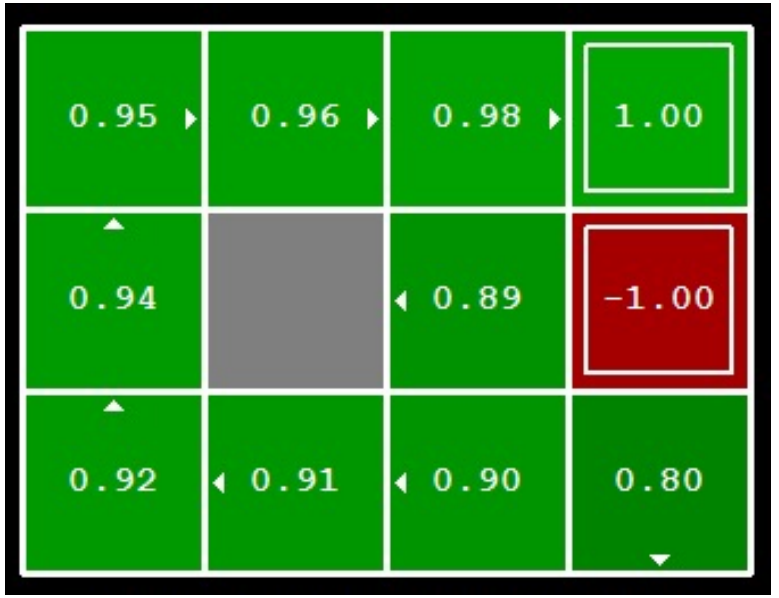


- Efficiency: $O(S^2)$ per iteration

- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Computing Actions from Values

- Let's imagine we have the optimal values V*(s)
- How should we act?
  - It's not obvious!
- We need to do a mini-expectimax (one step)

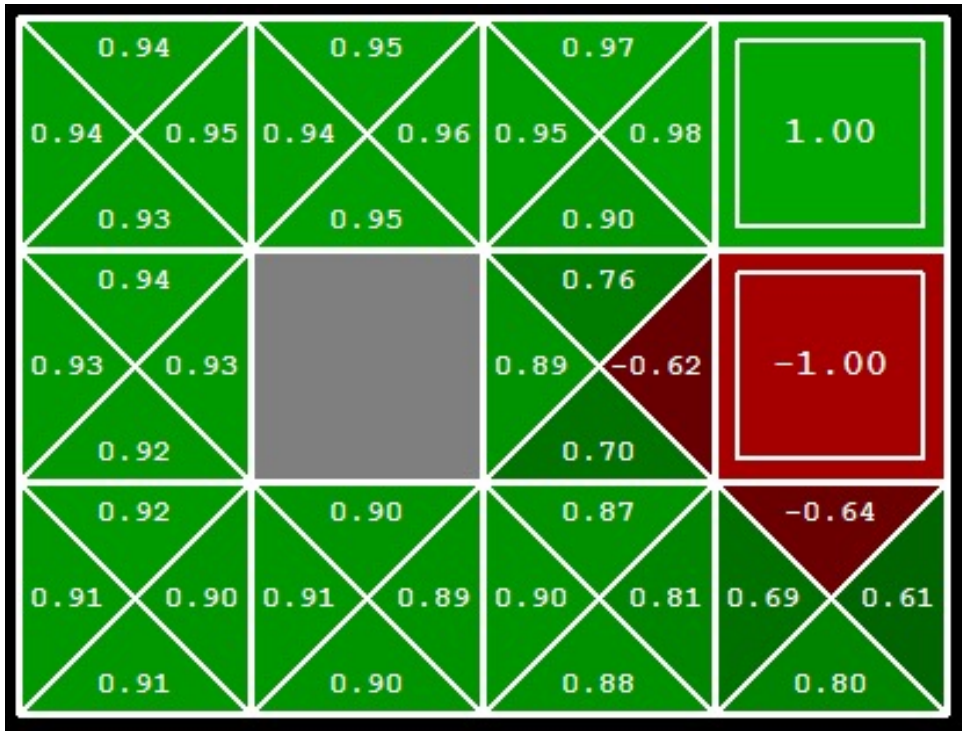


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- This is called policy extraction, since it gets the policy implied by the values

Computing Actions from Q-Values

- Let's imagine we have the optimal q-values:



- How should we act?
  - Completely trivial to decide!

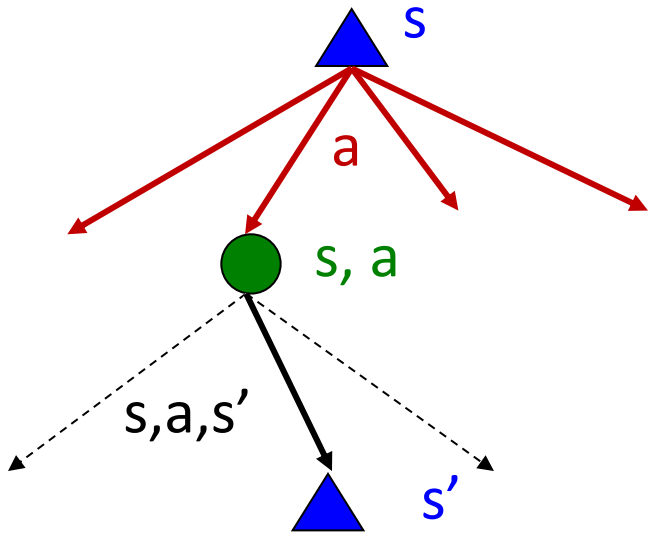
$$\pi^*(s) = \arg\max_a Q^*(s,a)$$

- Important lesson: actions are easier to select from q-values than values!

Recap: Problems with Value Iteration

- Value iteration repeats the Bellman updates:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_k(s') \right]$$



- Problem 1: It's slow – $O(S^2A)$ per iteration
- Problem 2: The "arg max" at each state rarely changes
- Problem 3: The policy often converges long before the values

[Demo: value iteration (L9D2)]

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- This is policy iteration
  - It's still optimal!
  - Can converge (much) faster under some conditions

Policy Iteration (PI)

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s') \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s') \right]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^{\pi_i}(s') \right]$$

Convergence of PI

- 1. Improvement: Does each policy improvement step produce a better policy?
- 2. Convergence: Does PI converge to an optimal policy?

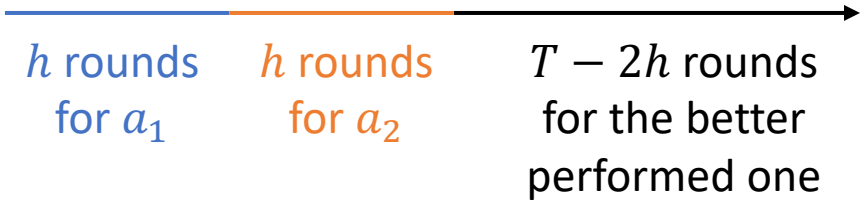
Bandits

Explore-then-commit (ETC) [Garivier et al., 2016]

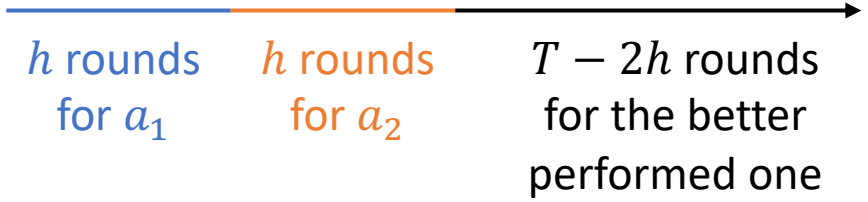
- There are $K = 2$ arms (choices/plans/...)
- Suppose
  - $\mu_1 > \mu_2$
  - $\Delta = \mu_1 - \mu_2$

A/B testing

- Explore-then-commit (ETC) algorithm
  - Select each arm h times
  - Find the empirically best arm A
  - Choose $A_t = A$ for all remaining rounds



Explore-then-commit (cont.)



- Regret analysis:

$$Reg(T) = T \cdot \mu_1 - \mathbb{E}\left[\sum_{t=1}^{T} \mu_{A_t}\right]$$

Sample mean

$$= h\Delta + (T - 2h) \cdot \Delta \cdot \mathbb{P}(\hat{\mu}_1 < \hat{\mu}_2)$$

$$= h\Delta + (T - 2h) \cdot \Delta \cdot \mathbb{P}\big((\hat{\mu}_2 - \mu_2) - (\hat{\mu}_1 - \mu_1) > \Delta\big)$$

$$\leq h\Delta + T \cdot \Delta \cdot \exp\left(-\frac{h\Delta^2}{4}\right)$$

Hoeffding's inequality

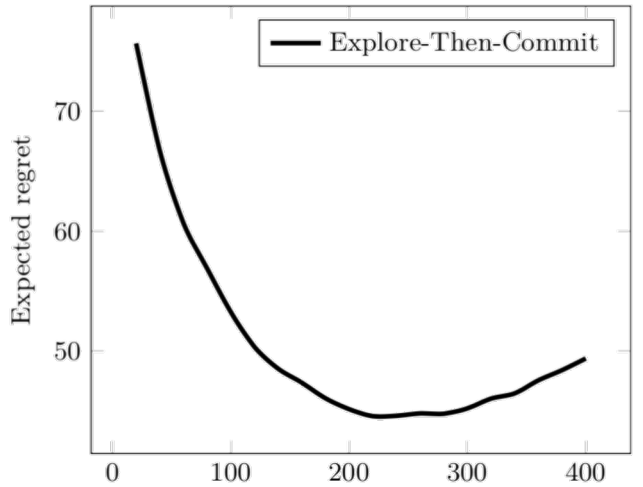
Exploration Exploitation

$$\leq O\left(\frac{\log T}{\Delta}\right)$$

Choose $h = \left\lceil \frac{4}{\Delta^2} \log\left(\frac{T\Delta^2}{4}\right) \right\rceil$

require the knowledge of Δ

- $Reg(T) = \Omega(T\Delta)$ if $h = 100$
- $Reg(T) = \Omega(T\Delta)$ if $h = T/10$



Only with the best choice of h the regret would be smallest

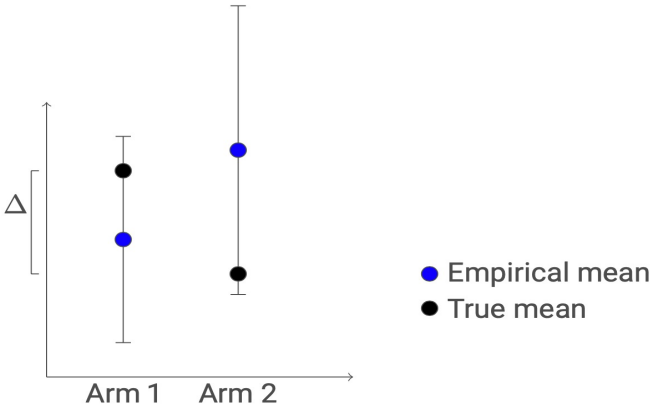
Upper confidence bound (UCB) [Auer et al., 2002]

- With high probability $\geq 1-\delta$ By Hoeffding's inequality

$$\mu_j \in \left[\hat{\mu}_j - \sqrt{\frac{\log 1/\delta}{T_j}}, \hat{\mu}_j + \sqrt{\frac{\log 1/\delta}{T_j}}\right]$$

Sample mean

Number of selections of a_j



- Optimism: Believe arms have higher rewards, encourage exploration
  - The UCB value represents the reward estimates
- For each round t, select the arm

$$A(t) \in \text{argmax}_{j\in[K]} \left\{\hat{\mu}_j + \sqrt{\frac{\log 1/\delta}{T_j(t)}}\right\}$$

Upper confidence bound (UCB)

Exploitation

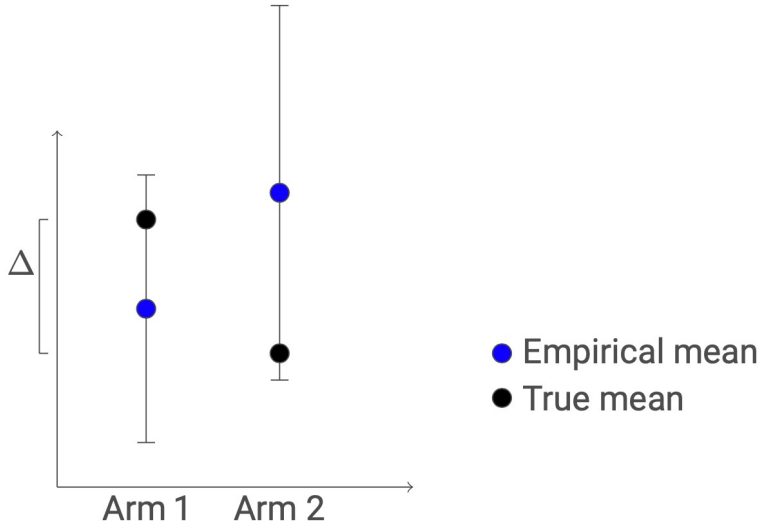
Exploration

Upper confidence bound (UCB) (cont.)

- Assume arm a_1 is the best arm
- If sub-optimal arm a_j is selected
  - w/ high probability

    $$\mu_1 \leq \mathrm{UCB}_1 \leq \mathrm{UCB}_j \leq \mu_j + 2\sqrt{\frac{\log 1/\delta}{T_j(t)}}$$

  - $\Longrightarrow 2\sqrt{\frac{\log 1/\delta}{T_j(t)}} \geq \Delta_j := \mu_1 - \mu_j$
  - $\Longrightarrow T_j(t) \leq O\left(\frac{\log 1/\delta}{\Delta_j^2}\right)$



Can choose δ adaptive to time t

- By choosing $\delta = 1/T$, cumulative regret:

$$O\left(\sum_{j\neq 1} \frac{\log T}{\Delta_j^2} \cdot \Delta_j\right) = O(K \log T/\Delta)$$

$\Delta := \min_{j\neq 1}\Delta_j$
Without knowing Δ

RL

Approaches to reinforcement learning

1. Model-based: Learn the model, solve it, execute the solution
2. Learn values from experiences, use to make decisions
    a. Direct evaluation
    b. Temporal difference learning
    c. Q-learning
3. Optimize the policy directly

Model-Based Learning

- Model-Based Idea:
  - Learn an approximate model based on experiences
  - Solve for values as if the learned model were correct

- Step 1: Learn empirical MDP model
  - Count outcomes s' for each s, a
  - Directly estimate each entry in $T(s,a,s')$ from counts
  - Discover each $R(s,a,s')$ when we experience the transition

- Step 2: Solve the learned MDP
  - Use, e.g., value or policy iteration, as before

Basic idea of model-free methods

- To approximate expectations with respect to a distribution, you can either
  - Estimate the distribution from samples, compute an expectation
  - Or, bypass the distribution and estimate the expectation from samples directly

Direct evaluation

- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from s onwards

- Idea:
  - Use ***returns***, the *actual* sums of discounted rewards from s
  - Average over multiple trials and visits to s

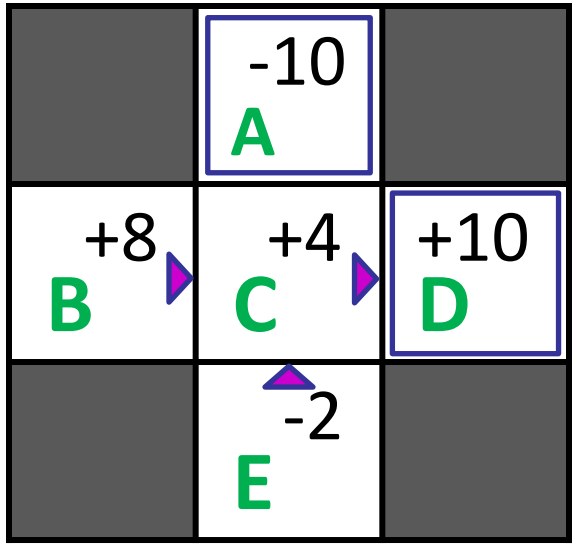
- This is called ***direct evaluation*** (or direct utility estimation)

Problems with Direct Estimation

- What's good about direct estimation?
  - It's easy to understand
  - It doesn't require any knowledge of *T* and *R*
  - It converges to the right answer in the limit
- What's bad about it?
  - Each state must be learned separately (fixable)
  - It ***ignores information about state connections***
  - So, it takes a long time to learn

E.g., B=at home, study hard
E=at library, study hard
C=know material, go to exam

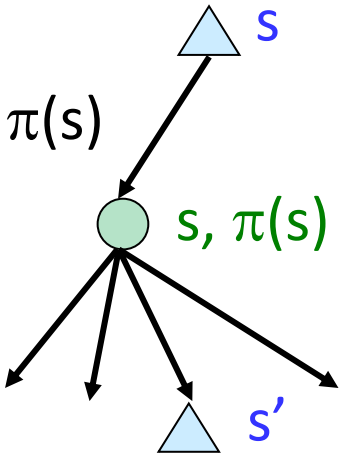
Output Values



If B and E both go to C under this policy, how can their values be different?

Temporal Difference Learning

- Big idea: learn from every experience!
  - Update V(s) each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

Sample of V(s): $sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$

Update to V(s): $V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + (\alpha)sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

Example: TD Value Estimation

- Experience transition *i:* (s_i, a_i, s_i', r_i).
- Compute sampled value "target": $r_i + \gamma V^{\pi}(s_i')$.
- Compute "TD error": $\delta_i = \left(r_i + \gamma V^{\pi}(s_i')\right) - V^{\pi}(s_i)$.
- Update: $V^{\pi}(s_i)$ += $\alpha_i \cdot \delta_i$.

s	*V(s)*
A	0
B	-2
C	9
D	10
E	8

i	*s*	*a*	*s'*	*r*	$r + \gamma V^{\pi}(s')$	$V^{\pi}(s)$	δ
1	B	east	C	-1	-1 + 0	0	-1
2	C	east	D	-1	-1 + 0	0	-1
3	D	exit	---	10	10 + 0	0	+10
4	B	east	C	-1	-1 + -1	-1	-1
5	C	east	D	-1	-1 + 10	-1	+10
6	D	exit	---	10	10 + 0	10	0
7	E	north	C	-1	-1 + 9	0	+8

B, east, C, -1
C, east, D, -1
D, exit, x, +10

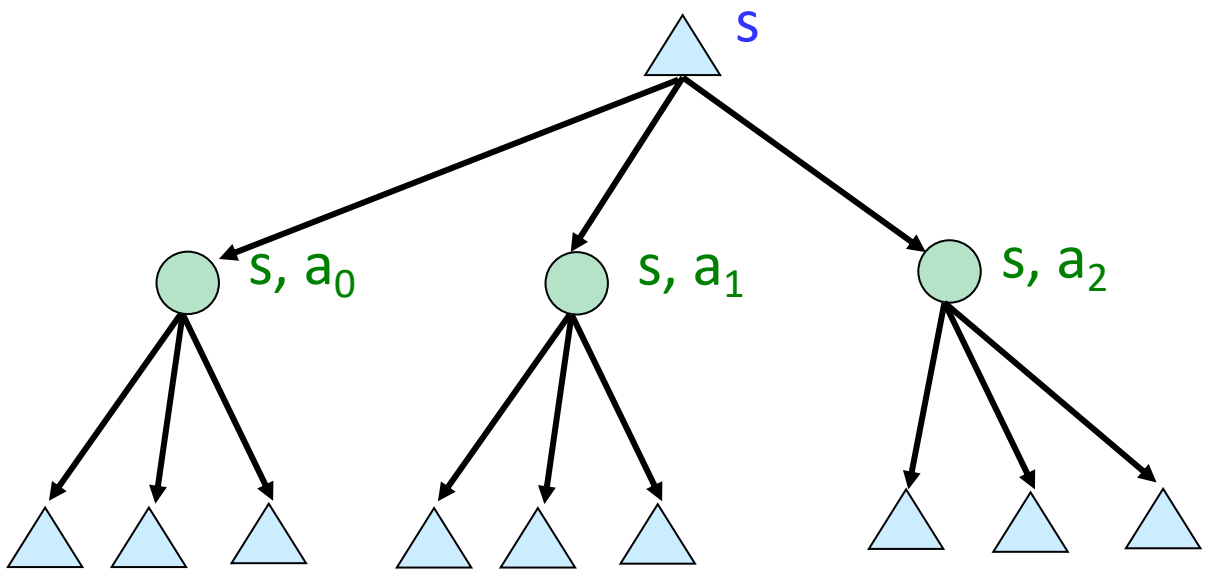
B, east, C, -1
C, east, D, -1
D, exit, x, +10

E, north, C, -1
C, east, D, -1
D, exit, x, +10

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Problems with TD Value Learning

- Model-free policy evaluation! 🎉
- Bellman updates with running sample mean! 🎉

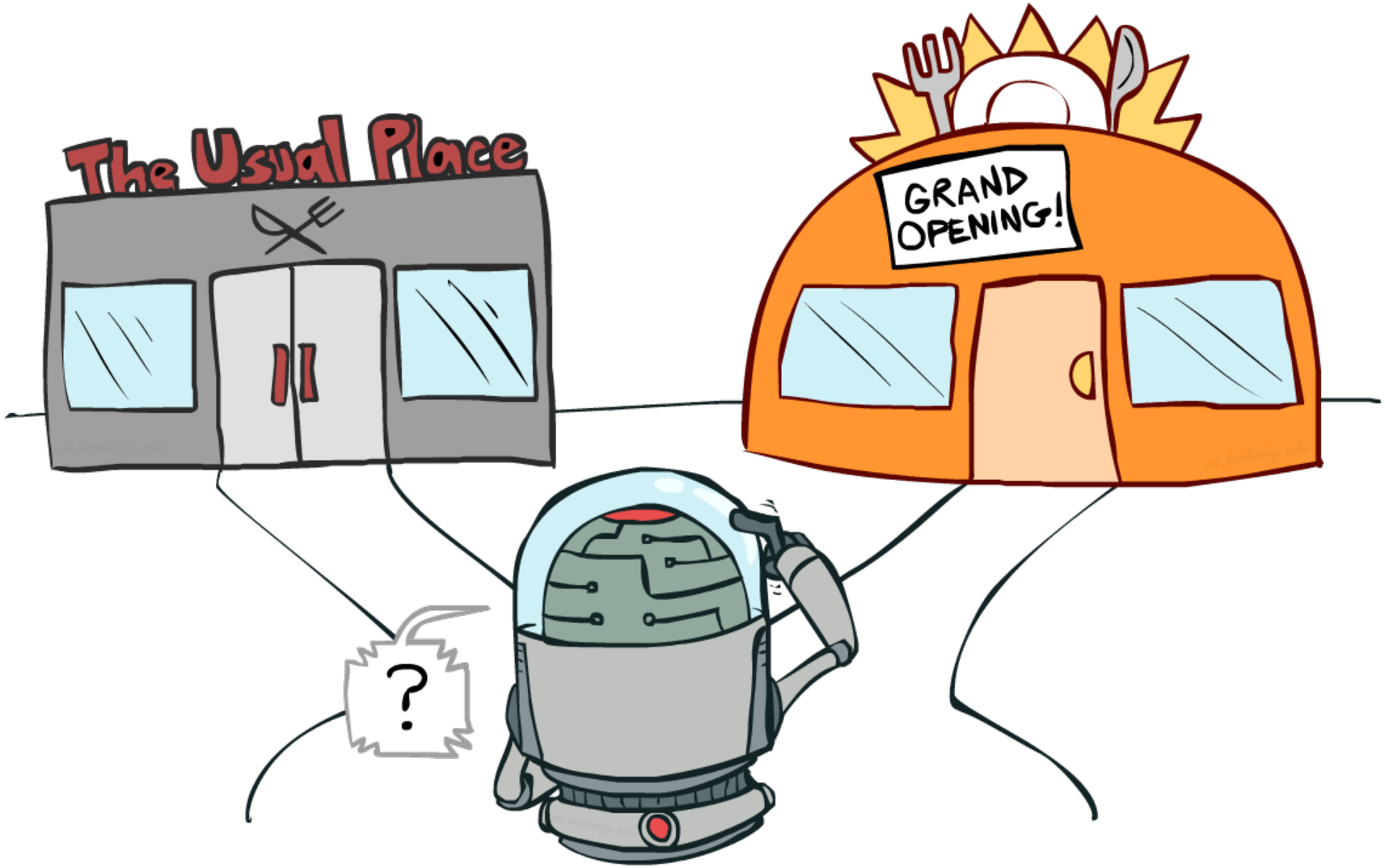


- Need the transition model to improve the policy! 😱

Q-learning as approximate Q-iteration

- Recall the definition of Q values:
  - $Q^*(s,a)$ = expected return from doing a in s and then behaving optimally thereafter; and $\pi^*(s) = \max_a Q^*(s,a)$
- Bellman equation for Q values:
  - $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$
- Approximate Bellman update for Q values:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
- We obtain a policy from learned Q(s,a), with no model!
  - (No free lunch: Q(s,a) table is |A| times bigger than V(s) table)

Exploration vs. Exploitation



Exploration method 1: ε-greedy

- ε-greedy exploration
  - Every time step, flip a biased coin
  - With (small) probability ε, act randomly
  - With (large) probability 1-ε, act on current policy

- Properties of ε-greedy exploration
  - Every s,a pair is tried infinitely often
  - Does a lot of stupid things
    - Jumping off a cliff *lots of times* to make sure it hurts
  - Keeps doing stupid things for ever
    - Decay ε towards 0

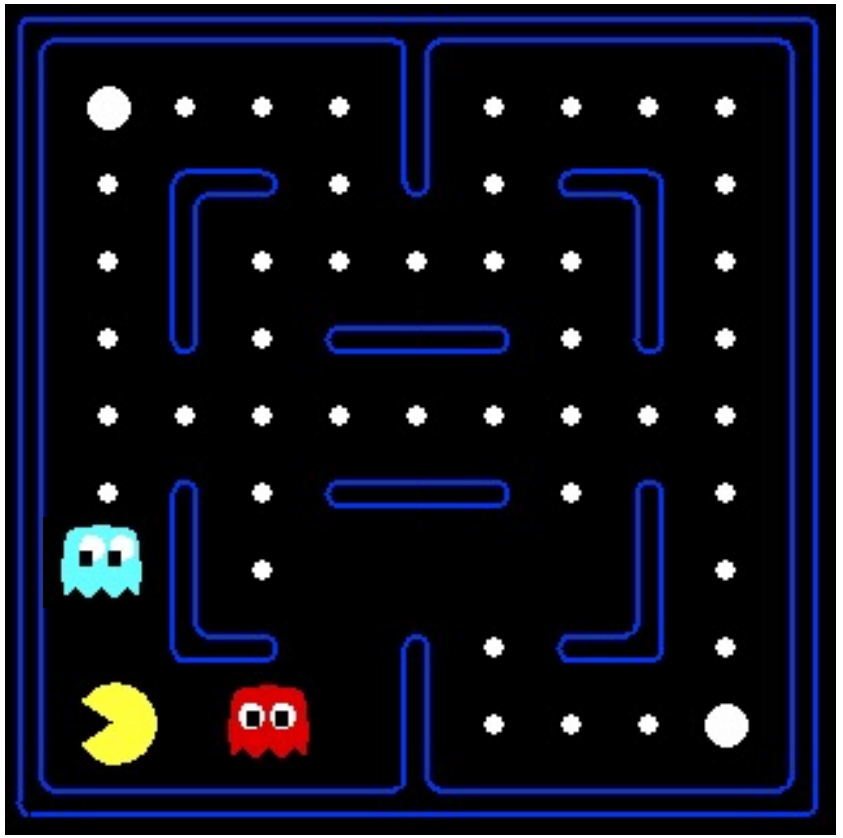
Method 2: Optimistic Exploration Functions

- ***Exploration functions*** implement this tradeoff
  - Takes a value estimate u and a visit count n, and returns an optimistic utility, e.g., $f(u,n) = u + k/\sqrt{n}$

- Regular Q-update:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a Q(s',a)]$
- Modified Q-update:
  - $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_a f(Q(s',a'),n(s',a'))]$
- Note: this propagates the "bonus" back to states that lead to unknown states as well!

Feature-Based Representations

- Solution: describe a state using a vector of ***features***
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost f_{GST}
    - Distance to closest dot
    - Number of ghosts
    - 1 / (distance to closest dot) f_{DOT}
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g., action moves closer to food)



Linear Value Functions

- We can express V or Q (approximately) as weighted linear functions of feature values:
  - $V_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$
  - $Q_\theta(s,a) = \theta_1 f_1(s,a) + \theta_2 f_2(s,a) + \ldots + \theta_n f_n(s,a)$

- Advantage: our experience is summed up in a few powerful numbers
  - Can compress a value function for chess (10^{43} states) down to about 30 weights!

- Disadvantage: states may share features but have very different expected utility!

SGD for Linear Value Functions

- Goal: Find parameter vector θ that minimizes the mean squared error between the true and approximate value function

$$J(\theta) = \mathbb{E}_\pi[\frac{1}{2}\big(V^\pi(s) - V_\theta(s)\big)^2]$$

- Stochastic gradient descent:

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$$

$$= \theta + \alpha\big(V^\pi(s) - V_\theta(s)\big)\frac{\partial V_\theta(s)}{\partial \theta}$$

Temporal-Difference (TD) Learning Objective

$$\theta \leftarrow \theta + \alpha\big(V^{\pi}(s) - V_{\theta}(s)\big)x(s)$$

- In TD learning, $r_{t+1} + \gamma V_{\theta}(s_{t+1})$ is a data sample for the target
- Apply supervised learning on "training data":

$$\langle s_1, r_2 + \gamma V_{\theta}(s_2)\rangle, \langle s_2, r_3 + \gamma V_{\theta}(s_3)\rangle, \ldots, \langle s_T, r_T\rangle$$

- For each data sample, update

$$\theta \leftarrow \theta + \alpha\big(r_{t+1} + \gamma V_{\theta}(s_{t+1}) - V_{\theta}(s)\big)x(s_t)$$

Q-Value Function Approximation

- Approximate the action-value function:

$$Q_\theta(s,a) \simeq Q^\pi(s,a)$$

- Objective: Minimize the **mean squared error**:

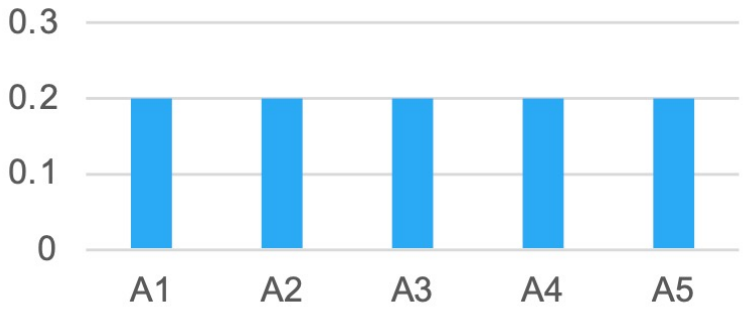
$$J(\theta) = \mathbb{E}_\pi\left[\frac{1}{2}(Q^\pi(s,a) - Q_\theta(s,a))^2\right]$$

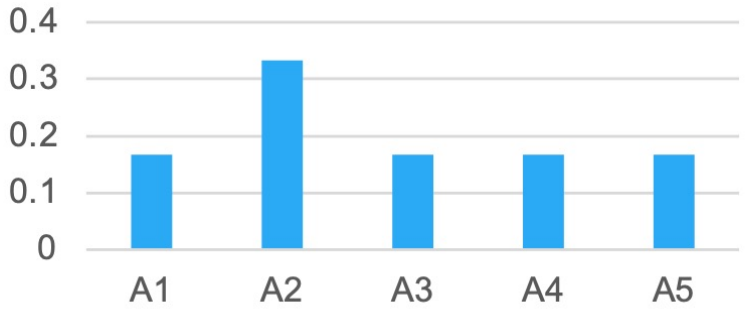
- Stochastic Gradient Descent on a single sample

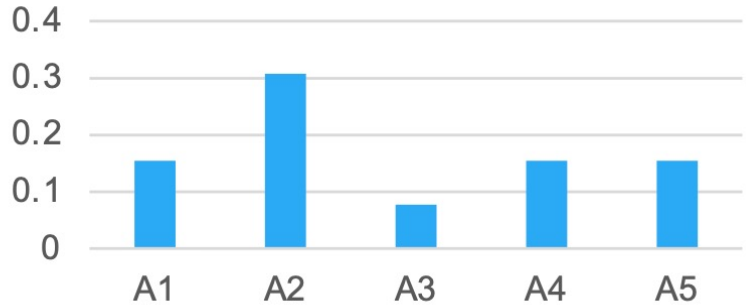
$$\theta \leftarrow \theta + \alpha\left(r_{t+1} + \gamma Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s,a)\right)\frac{\partial Q_\theta(s,a)}{\partial \theta}$$

Policy Gradient

- Simplest version:
  - Start with initial policy $\pi(s)$ that assigns probability to each action
  - Sample actions according to policy π
  - Update policy:
    - If an episode led to high utility, make sampled actions more likely
    - If an episode led to low utility, make sampled actions less likely







Policy Gradient in a Single-Step MDP

- Consider a simple single-step Markov Decision Process (MDP)
  - The initial state is drawn from a distribution: $s \sim d(s)$
  - The process terminates after one action, yielding a reward r_{sa}
- Expected Value of the Policy

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(a|s) r_{sa}$$

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s \in S} d(s) \sum_{a \in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$

Likelihood Ratio Trick

- Use the identity:

$$\frac{\partial \pi_\theta(a|s)}{\partial \theta} = \pi_\theta(a|s)\frac{1}{\pi_\theta(a|s)}\frac{\partial \pi_\theta(a|s)}{\partial \theta}$$
$$= \pi_\theta(a|s)\frac{\partial \log \pi_\theta(a|s)}{\partial \theta}$$

- The gradient of the expected return can be written as:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[r] = \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(a|s) r_{sa}$$
$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{s\in S} d(s) \sum_{a\in A} \frac{\partial \pi_\theta(a|s)}{\partial \theta} r_{sa}$$
$$= \sum_{s\in S} d(s) \sum_{a\in A} \pi_\theta(a|s) \frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa}$$
$$= \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} r_{sa}\right]$$

Can be approximated by sampling s from d(s) and a from π_θ

Extension to Multi-step MDP

- Replace the instantaneous reward r(s,a) with the Q-value

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s,a)\right]$$

Richard Sutton's Reinforcement Learning: An Introduction (Chapter 13)

REINFORCE Algorithm

- Use the cumulative reward G_t as an estimator for $Q^{\pi_\theta}(s, a)$

- initialize θ arbitrarily

  for each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ do

    for $t = 1$ to $T - 1$ do

      $$\theta \leftarrow \theta + \alpha \frac{\partial}{\partial \theta} \log \pi_\theta(a_t | s_t) G_t$$

    end for

  end for

  return θ

Actor-Critic

- Intuition
  - REINFORCE estimates the policy gradient using Monte Carlo returns G_t to approximate $Q(s_t, a_t)$
  - Why not learn a trainable value function $Q_\phi(s, a)$ to estimate $Q^\pi(s, a)$ directly?
- Actor and critic

Actor $\pi_\theta(a|s)$

Improve the policy based on value estimates provided by the critic

Critic $Q_\Phi(s, a)$

Evaluate the value of actions taken by the actor's policy

Training of the Actor-Critic Algorithm

- Critic: $Q_\phi(s, a)$
  - Learns to accurately estimate the action-value under the current actor policy

$$Q_\Phi(s, a) \simeq r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi_\theta(a'|s')}[Q_\Phi(s', a')]$$

- Actor: $\pi_\theta(a|s)$
  - Learns to take actions that maximize the critic's estimated value

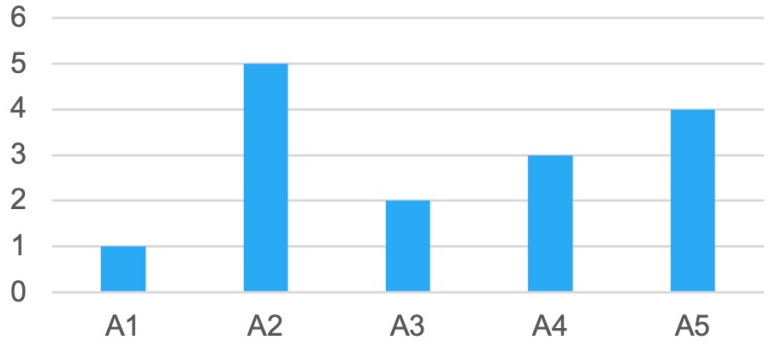
$$J(\theta) = \mathbb{E}_{s \sim p, \pi_\theta}[\pi_\theta(a|s) Q_\Phi(s, a)]$$

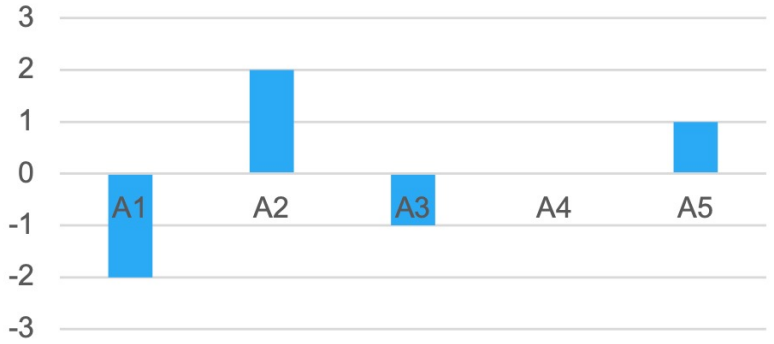
$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q_\Phi(s, a)\right]$$

A2C: Advantageous Actor-Critic

- Idea: Normalize the critic's score by subtracting a baseline function (often a value function V(s))
  - Provides more informative feedback:
    - Decrease the probability of worse-than-average actions
    - Increase the probability of better-than-average actions
  - Helps to further reduce variance in policy gradient estimates

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

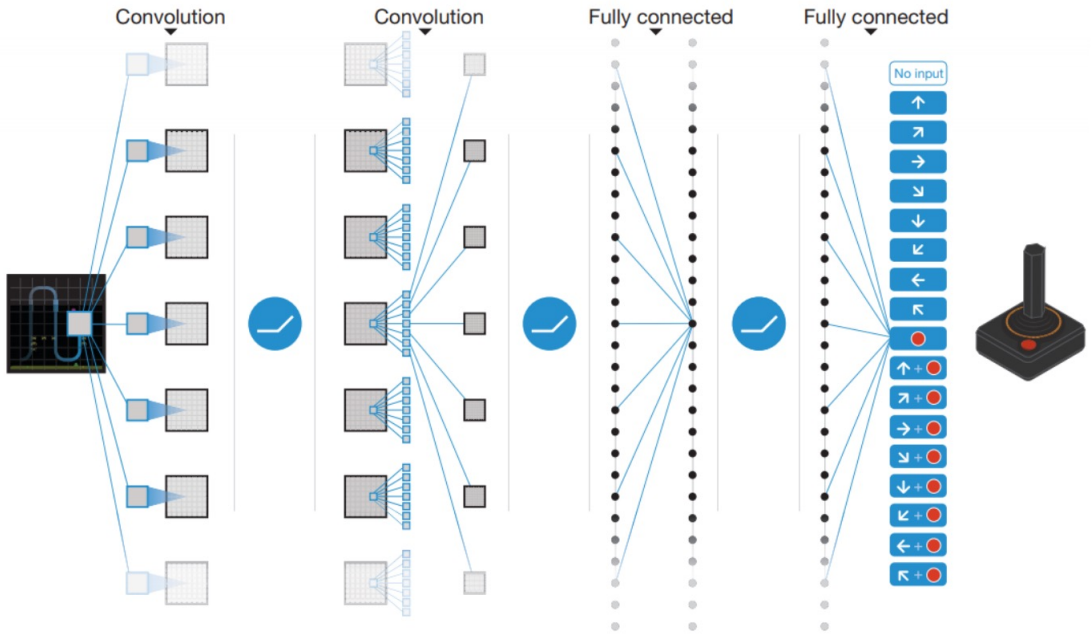




Deep RL

Value methods: DQN

- Deep Q-Network (DQN)
  - Uses a deep neural network to approximate Q(s,a)
    - → Replaces the Q-table with a parameterized function for scalability
  - The network takes state s as input, outputs Q-values for all actions a simultaneously



Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. Playing Atari with Deep Reinforcement Learning. NIPS 2013 workshop.

DQN (cont.)

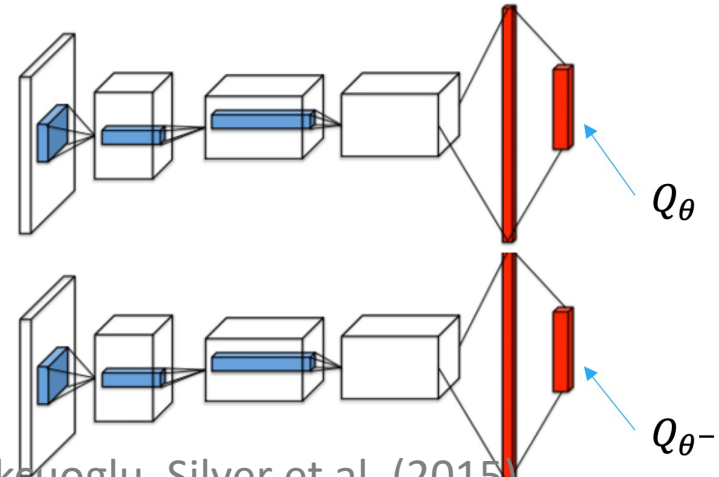
- Intuition: Use a deep neural network to approximate Q(s,a)
  - Instability arises in the learning process
    - Samples $\{(s_t, a_t, s_{t+1}, r_t)\}$ are collected sequentially and do not satisfy the i.i.d. assumption
    - Frequent updates of Q(s,a) cause instability
- Solutions: Experience replay
  - Store transitions $e_t = (s_t, a_t, s_{t+1}, r_t)$ in a replay buffer D
    Sample uniformly from D to reduce sample correlation
  - Dual network architecture: Use an evaluation network and a target network for improved stability

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Target network

- Target network $Q_{\theta^-}(s, a)$
  - Maintains a copy of the Q-network with older parameters θ^-
  - Parameters θ^- are updated periodically (every C steps) to match the evaluation network
- Loss Function (at iteration i)

$$L_i(\theta_i) = \mathbb{E}_{s_t, a_t, s_{t+1}, r_t, p_t \sim D}\left[\frac{1}{2}\omega_t\left(r_t + \gamma \max_{a'} Q_{\theta_i^-}(s_{t+1}, a') - Q_{\theta_i}(s_t, a_t)\right)^2\right]$$



"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

DQN training procedure

- Collect transitions using an ε-greedy exploration policy
  - Store $\{(s_t, a_t, s_{t+1}, r_t)\}$ into the replay buffer
- Sample a minibatch of k transitions from the buffer
- Update networks:
  - Compute the target using the sampled transitions
  - Update the evaluation network Q_θ
  - Every C steps, synchronize the target network Q_{θ^-} with the evaluation network

"Human-Level Control Through Deep Reinforcement Learning", Mnih, Kavukcuoglu, Silver et al. (2015)

Overestimation in Q-Learning

- Q-function overestimation
  - The target value is computed as: $y_t = r_t + \gamma \max_{a'} Q_\theta(s_{t+1}, a')$
  - The max operator leads to increasingly larger Q-values, potentially exceeding the true value
- Cause of overestimation

$$\max_{a' \in A} Q_{\theta'}(s_{t+1}, a') = Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_{\theta'}(s_{t+1}, a'))$$

  - The chosen action might be overestimated due to Q-function error

Double DQN

- Uses two separate networks for action selection and value estimation, respectively.

$$\text{DQN} \quad y_t = r_t + \gamma Q_\theta(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

$$\text{Double DQN} \quad y_t = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

"Double Reinforcement Learning with Double Q-Learning", van Hasselt et al. (2016)

Dueling DQN

- Advantage function

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s,a)]$$

- Different forms of advantage aggregation

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \max_{a' \in |A|} A(s,a';\theta,\alpha)\right)$$

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)\right)$$

Policy network gradient

- For stochastic policies, the probability of selecting an action is typically modeled using a softmax function:

$$\pi_\theta(a|s) = \frac{e^{f_\theta(s,a)}}{\sum_{a'} e^{f_\theta(s,a')}}$$

  - $f_\theta(s,a)$ is a score function (e.g., logits) for the state-action pair
  - Parameterized by θ, often realized via a neural network

- Gradient of the log-form

$$\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} = \frac{\partial f_\theta(s,a)}{\partial \theta} - \frac{1}{\sum_{a'} e^{f_\theta(s,a')}} \sum_{a''} e^{f_\theta(s,a'')} \frac{\partial f_\theta(s,a'')}{\partial \theta}$$

$$= \frac{\partial f_\theta(s,a)}{\partial \theta} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)} \left[\frac{\partial f_\theta(s,a')}{\partial \theta} \right]$$

Policy network gradient (cont.)

- Gradient of the log-form

$$\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} = \frac{\partial f_\theta(s,a)}{\partial \theta} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)} \left[\frac{\partial f_\theta(s,a')}{\partial \theta} \right]$$

- Gradient of the policy network

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\pi_\theta} \left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s,a) \right]$$

$$= \mathbb{E}_{\pi_\theta} \left[\left(\underbrace{\frac{\partial f_\theta(s,a)}{\partial \theta}}_{\text{Back propagation}} - \mathbb{E}_{a' \sim \pi_\theta(a'|s)} \left[\underbrace{\frac{\partial f_\theta(s,a')}{\partial \theta}}_{\text{Back propagation}} \right] \right) Q^{\pi_\theta}(s,a) \right]$$

Comparison: DQN v.s. Policy gradient

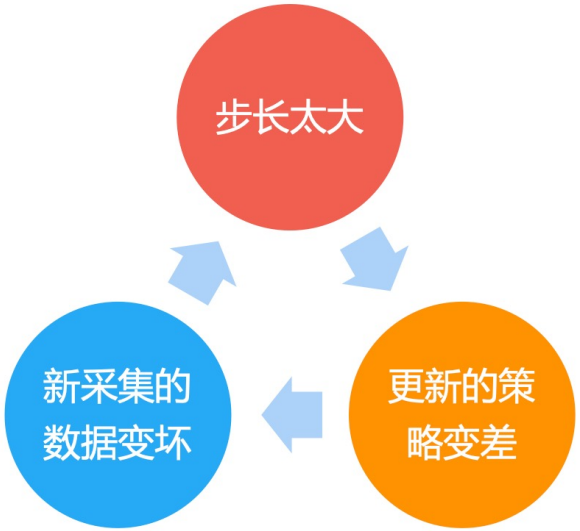
- Q-Learning:
  - Learns a Q-value function $Q_\theta(s, a)$ parameterized by θ
  - Objective: Minimize the TD error
- Policy gradient
  - Learns a policy $\pi_\theta(a \mid s)$ directly, parameterized by θ
  - Objective: Maximize the expected return directly

$$\max_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s, a)]$$

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta} = \theta + \alpha \mathbb{E}_{\pi_\theta}\left[\frac{\partial \log \pi_\theta(a|s)}{\partial \theta} Q^{\pi_\theta}(s, a)\right]$$

Limitations of policy gradient methods

- Learning rate (step size) selection is challenging in policy gradient algorithms
  - Since the data distribution changes as the policy updates, a previously good learning rate may become ineffective.
  - A poor choice of step size can significantly degrade performance:
    - Too large → policy diverges or collapses
    - Too small → slow convergence or stagnation



Optimization gap of the objective function

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_t \gamma^t r(s_t, a_t)]$$
$$J(\theta) = \mathbb{E}_{s_0 \sim p_\theta(s_0)}[V^{\pi_\theta}(s_0)]$$

- New policy θ' and old policy θ

$$J(\theta') - J(\theta) = J(\theta') - \mathbb{E}_{s_0 \sim p(s_0)}[V^{\pi_\theta}(s_0)]$$

$$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)}[\sum_{t=0} \gamma^t A^{\pi_\theta}(s_t, a_t)]$$

Sampling inconvenience

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

Importance sampling

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

$$J(\theta') - J(\theta)$$

$$= \mathbb{E}_{\tau \sim p_{\theta'}(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t) \right]$$

$$= \sum_t \mathbb{E}_{s_t \sim p_{\theta'}(s_t)} [\mathbb{E}_{a_t \sim \pi_{\theta'}(a_t|s_t)} [\gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$$= \sum_t \mathbb{E}_{s_t \sim p_{\theta'}(s_t)} [\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} [\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$p_{\theta'}$, approximation

Importance sampling

TRPO Policy Constraint

- Use KL divergence to constrain policy update magnitude:

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$$\text{such that } \mathbb{E}_{s_t \sim p_\theta(s_t)}[D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t))] \leq \epsilon$$

  - In practice: use penalized objective with KL divergence penalty instead of hard constraint

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$$-\lambda(D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t)) - \epsilon)$$

    - Update θ' and $\lambda \leftarrow \lambda + \alpha(D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t)) - \epsilon)$

TRPO Drawbacks

Use KL divergence to constrain policy update magnitude:

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]]$$

$$\text{such that } \mathbb{E}_{s_t \sim p_\theta(s_t)}[D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t))] \leq \epsilon$$

- In practice: use penalized objective with KL divergence penalty instead of hard constraint

$$\theta' \leftarrow \arg\max_{\theta'} \sum_t \mathbb{E}_{s_t \sim p_\theta(s_t)}[\mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)} \gamma^t A^{\pi_\theta}(s_t, a_t)]] - \lambda(D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t)) - \epsilon)$$

  - Update θ' and $\lambda \leftarrow \lambda + \alpha(D_{KL}(\pi_{\theta'}(a_t|s_t) \parallel \pi_\theta(a_t|s_t)) - \epsilon)$

- High variance from importance weights
- Difficult to solve constrained optimization

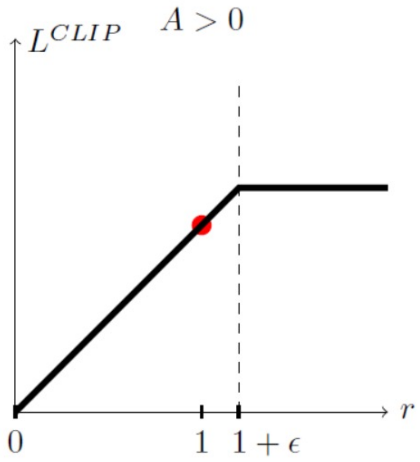
Proximal Policy Optimization (PPO)

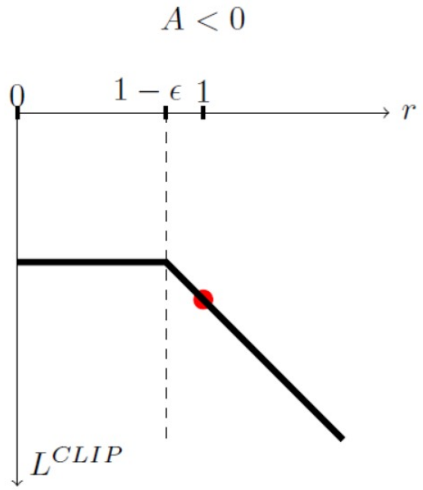
- Clipped Surrogate Objective

conservative policy iteration

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$





Construct the lower bound: $L^{CLIP}(\theta) \leq L^{CPI}(\theta)$

Equivalent at r=1: $L^{CLIP}(\theta) = L^{CPI}(\theta)$

PPO: improvement over TRPO

- 1.Clipped surrogate objective

conservative policy iteration

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right]$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]$$

- 2.Generalized advantage estimation

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T)$$

  - Use parallel actors to collect rollouts, compute advantage estimates, and update parameters with minibatches.

PPO: improvement over TRPO

- 3. Adaptive penalty parameter

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t)|\pi_\theta(\cdot|s_t)] \right]$$

  - Adjust the penalty coefficient β dynamically:
    - Compute the KL value $d = \hat{\mathbb{E}}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t)|\pi_\theta(\cdot|s_t)] \right]$
    - If d < target / 1.5 → β← β / 2
    - If d > target × 1.5 → β← β × 2

Note: Here, 1.5 and 2 are empirical parameters, and the algorithm performance is not very sensitive to them