南方科技大学

MAT8034: Machine Learning

Final Review

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

南方科技大学

MAT8034: Machine Learning

Supervised Learning: Linear Regression

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

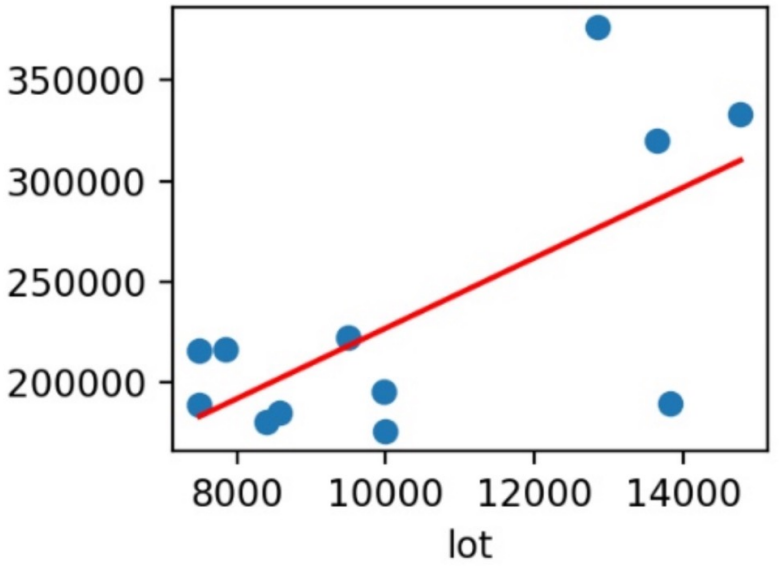
Part of slide credit: Stanford CS229

Definition

- Input $\mathcal{X}$ (house data), output $\mathcal{Y}$ (price)
- A hypothesis or a prediction function $h: \mathcal{X} \to \mathcal{Y}$
- A training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$
  - $x^{(1)}$: $x_1^{(1)}$ represents the living area, $x_2^{(1)}$ represents the #bed room
- Given a training set, our goal is to produce a good function h
  - Will use h in the new data not in the training set

How to represent h?

- Simplest fit



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

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 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

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, stochastic gradient descent (SGD)
  - SGD is the workhorse of modern ML, e.g., pytorch & tensorflow

Stochastic minibatch

- We randomly select a **batch** of $B \subseteq \{1, \ldots, n\}$ where $|B| < n$.
- We approximate the gradient using just those B points as follows (vs. gradient descent)

$$\frac{1}{|B|} \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)} \text{ v.s. } \frac{1}{n} \sum_{j=1}^{n} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- So our update rule for SGD is:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- NB: scaling of $|B|$ versus n is "hidden" inside choice of α_B.

Stochastic minibatch v.s. Gradient descent

- Recall our rule B points as follows:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha_B \sum_{j \in B} \left(h_\theta(x^{(j)}) - y^{(j)} \right) x^{(j)}.$$

- If $|B| = \{1, \ldots, n\}$ (the whole set), then they coincide.
- Smaller B implies a lower quality approximation of the gradient (higher variance).
- Nevertheless, it may actually converge faster! (Case where the dataset has many copies of the same point–extreme, but lots of redundancy)

Normal Equations

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:

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

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

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

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

Probabilistic interpretation

A Justification for Least Squares?

- **Given** a training set $\{(x^{(i)}, y^{(i)}) \text{ for } i = 1, \ldots, n\}$ in which $x^{(i)} \in \mathbb{R}^{d+1}$ and $y^{(i)} \in \mathbb{R}$.
- **Do** find $\theta \in \mathbb{R}^{d+1}$ s.t. $\theta = \text{argmin}_\theta \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$ in which $h_\theta(x) = \theta^T x$.

Where did this model come from?

One way to view is via a *probabilistic interpretation (helpful throughout the course)*.

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.*

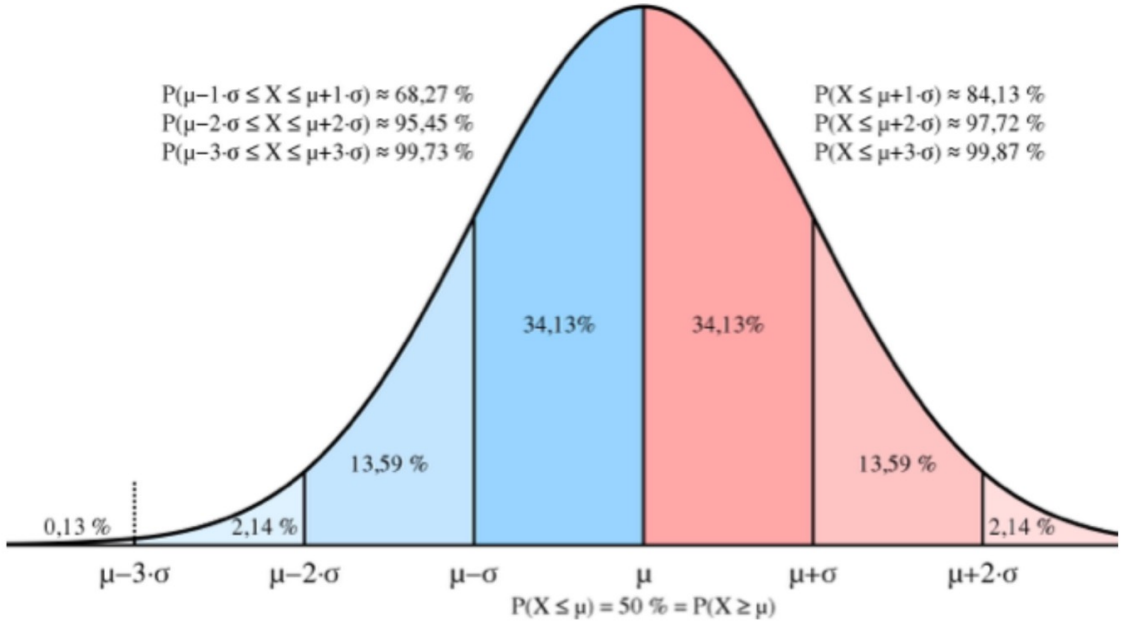
Gaussian distribution

We write $z \sim \mathcal{N}(\mu, \sigma^2)$ and read these symbols as

> *z is distributed as a normal with mean μ and standard deviation σ^2.*

or equivalently

$$P(z) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(z-\mu)^2}{2\sigma^2}\right\}.$$



Distribution of y

Recall in our model,

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)} \text{ in which } \varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2).$$

or more compactly notation:

$$y^{(i)} \mid x^{(i)}; \theta \sim \mathcal{N}(\theta^T x, \sigma^2).$$

equivalently,

$$P\left(y^{(i)} \mid x^{(i)}; \theta\right) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(y^{(i)} - x^T\theta)^2}{2\sigma^2}\right\}$$

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

Summary

- The regression problem for house pricing
- LMS
  - Gradient descent
  - Normal equation
- Justification for LMS
  - Log likelihood

南方科技大学

MAT8034: Machine Learning

Classification and Logistic Regression

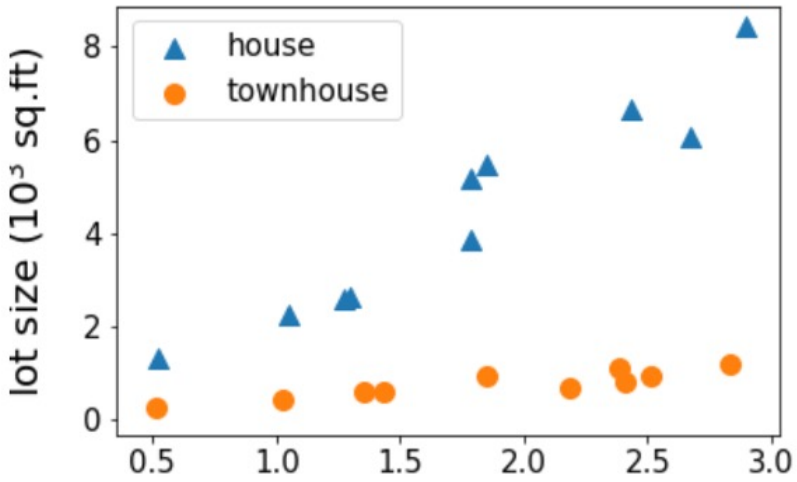
Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

Intuition of logistic regression

- Hope to use the linear method to solve the classification problem
- Given a a training set: $\{(x^{(i)}, y^{(i)}), i = 1,2, \dots, n\}$, let $y^{(i)} \in \{0,1\}$



- Build the connection between p and $\theta^{\top}x = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- $p \in (0,1)$ but $\theta^{\top}x \in (-\infty, +\infty)$

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)
  - $p \sim \frac{1}{1+\exp(-\theta^\top x)} := \text{sigmoid}(\theta^\top x) = h_\theta(x)$

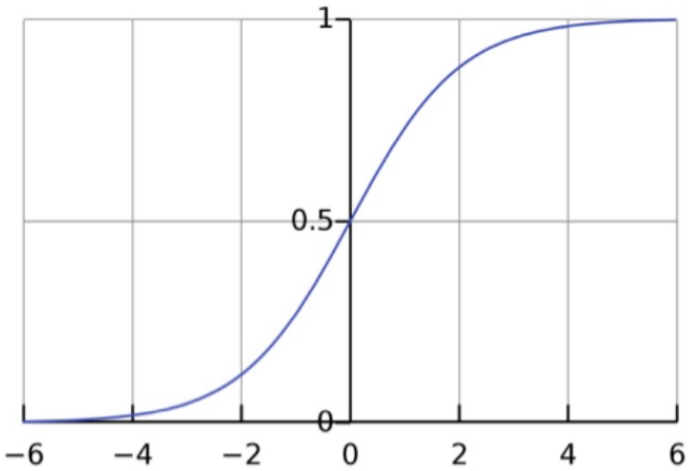
Logistic Regression

- Given a training set $\{(x^{(i)}, y^{(i)})$ for $i = 1, \ldots, n\}$ let $y^{(i)} \in \{0, 1\}$. Want $h_\theta(x) \in [0, 1]$. Let's pick a smooth function:

$$h_\theta(x) = g(\theta^T x)$$

- Here, g is a link function. There are *many*. . . but we'll pick one!

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



How do we interpret $h_\theta(x)$?

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

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

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

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

Another view: logistic loss

- In linear regression
  - The loss function is $J(h_\theta(x^{(i)}), y) = \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
- For the classification
  - Define the loss function

Cross-entropy loss

$$\ell_{\text{logistic}}(t, y) \triangleq y \log(1 + \exp(-t)) + (1 - y) \log(1 + \exp(t)) \,. \qquad (2.3)$$

  - When $y = 1$, minimizing the loss gets $t \to +\infty, p \to 1$
  - When $y = 0$, minimizing the loss gets $t \to -\infty, p \to 0$

Another view: logistic loss

- For the classification
  - Define the loss function

$$\ell_{\text{logistic}}(t, y) \triangleq y \log(1 + \exp(-t)) + (1 - y) \log(1 + \exp(t)) \,. \tag{2.3}$$

  - The relationship between the loss and log likelihood $-\ell(\theta) = \ell_{\text{logistic}}(\theta^\top x, y)$

$$\frac{\partial \ell_{\text{logistic}}(t, y)}{\partial t} = y \frac{-\exp(-t)}{1 + \exp(-t)} + (1 - y) \frac{1}{1 + \exp(-t)} \tag{2.5}$$

$$= 1/(1 + \exp(-t)) - y. \tag{2.6}$$

Then, using the chain rule, we have that

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = -\frac{\partial \ell_{\text{logistic}}(t, y)}{\partial t} \cdot \frac{\partial t}{\partial \theta_j} \tag{2.7}$$

$$= (y - 1/(1 + \exp(-t))) \cdot x_j = (y - h_\theta(x)) x_j \,, \tag{2.8}$$

Newton's method

Another algorithm to maximize $\ell(\theta)$

Newton's method: formulation

- Returning to logistic regression with g(z) being the sigmoid function
- A different algorithm for maximizing the log likelihood $\ell(\theta)$

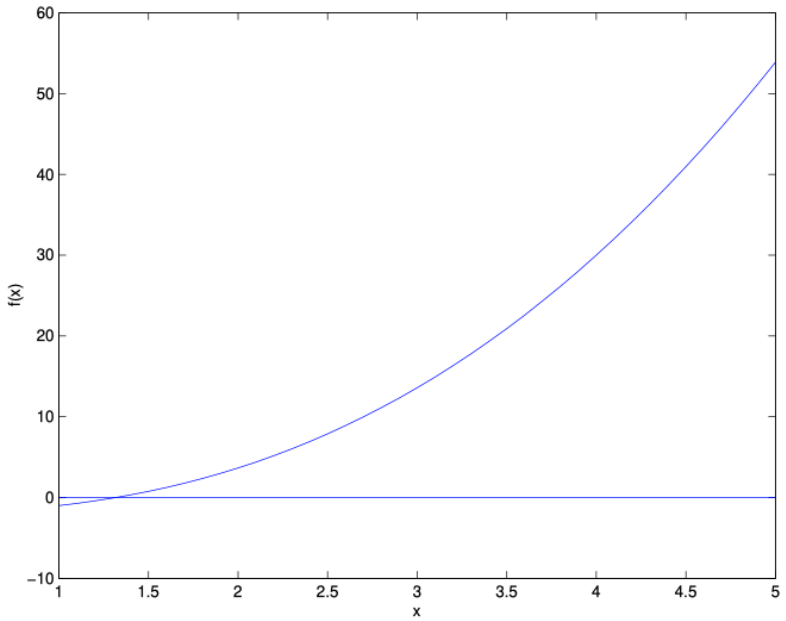
- To maximize $\ell(\theta)$, hope to find θ such that $\nabla\ell(\theta) = 0$

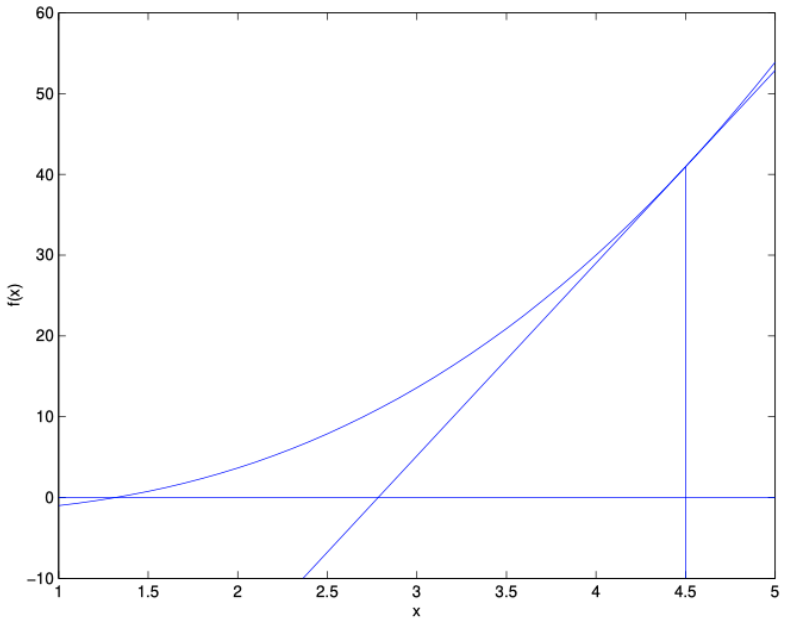
- New formulation

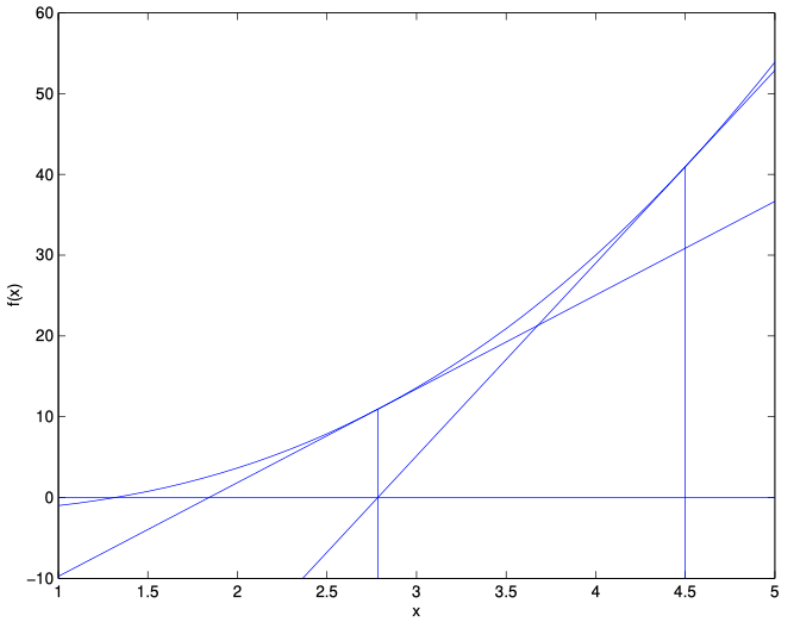
$$\text{Given } f : \mathbb{R}^d \to \mathbb{R} \text{ find } \theta \text{ s.t. } f(\theta) = 0$$

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? $\theta := \theta - \frac{\ell'(\theta)}{\ell''(\theta)}$

Properties of Newton's method

- Convergence rate?
  - Use the Hessian information to determine step size, more adaptive
  - May converge very fast
- Computational cost?
  - Computing Hessian requires $O(d^2)$

南方科技大学

MAT8034: Machine Learning

Generalized Linear Models

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

The exponential family

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)

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

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

Example 1: Bernoulli distribution

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

- Bernoulli(ϕ)
- $p(y = 1;\phi) = \phi;\ p(y = 0;\phi) = 1 - \phi$
- $$\begin{aligned} p(y;\phi) &= \phi^y(1-\phi)^{1-y} \\ &= \exp(y\log\phi + (1-y)\log(1-\phi)) \\ &= \exp\left(\left(\log\left(\frac{\phi}{1-\phi}\right)\right)y + \log(1-\phi)\right) \end{aligned}$$

$$\eta = \log(\phi/(1-\phi))$$

$$\begin{aligned} T(y) &= y \\ a(\eta) &= -\log(1-\phi) \\ &= \log(1+e^\eta) \\ b(y) &= 1 \end{aligned}$$

Example 2: Gaussian distribution with $\sigma^2 = 1$

- Gaussian(μ, 1)

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

$$\begin{aligned} p(y;\mu) &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}(y-\mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{1}{2}y^2\right)\cdot\exp\left(\mu y - \frac{1}{2}\mu^2\right) \end{aligned}$$

Thus, we see that the Gaussian is in the exponential family, with

$$\begin{aligned} \eta &= \mu \\ T(y) &= y \\ a(\eta) &= \mu^2/2 \\ &= \eta^2/2 \\ b(y) &= (1/\sqrt{2\pi})\exp(-y^2/2). \end{aligned}$$

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?

Log Partition Function

- Yes! Recall that

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

- Then, taking derivatives

$$\nabla_\eta a(\eta) = \frac{\sum_y T(y) b(y) \exp\left\{\eta^T T(y)\right\}}{\sum_y b(y) \exp\left\{\eta^T T(y)\right\}} = \mathbb{E}[T(y); \eta]$$

- Note: $\nabla^2_\eta a(\eta) = \text{var}[T(y); \eta]$, you can check!

- Takeaway: In this way, once we're in the exponential family, we get inference "for free" meaning in the same way for every member

Some Facts About Exponential Models

- There are many canonical exponential family models:
  - Binary $\mapsto$ Bernoulli
  - Multiple Classses $\mapsto$ Multinomial
  - Real $\mapsto$ Gaussian
  - Counts $\mapsto$ Poisson
  - $\mathbb{R}_+ \mapsto$ Gamma, Exponential
  - Distributions $\mapsto$ Dirichlet
- In this course, we'll use $T(y) = y$.

The GLMs

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 $\eta_i = \theta_i^T x$.)

Design choice

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

南方科技大学

MAT8034: Machine Learning

Generative Learning Algorithms

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

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

Generative learning algorithms

- Use Bayes rule
  - $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$
- Prediction
  - $\text{argmax}_y p(y|x) = \text{argmax}_y p(x|y)p(y)$

Gaussian discriminant analysis

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

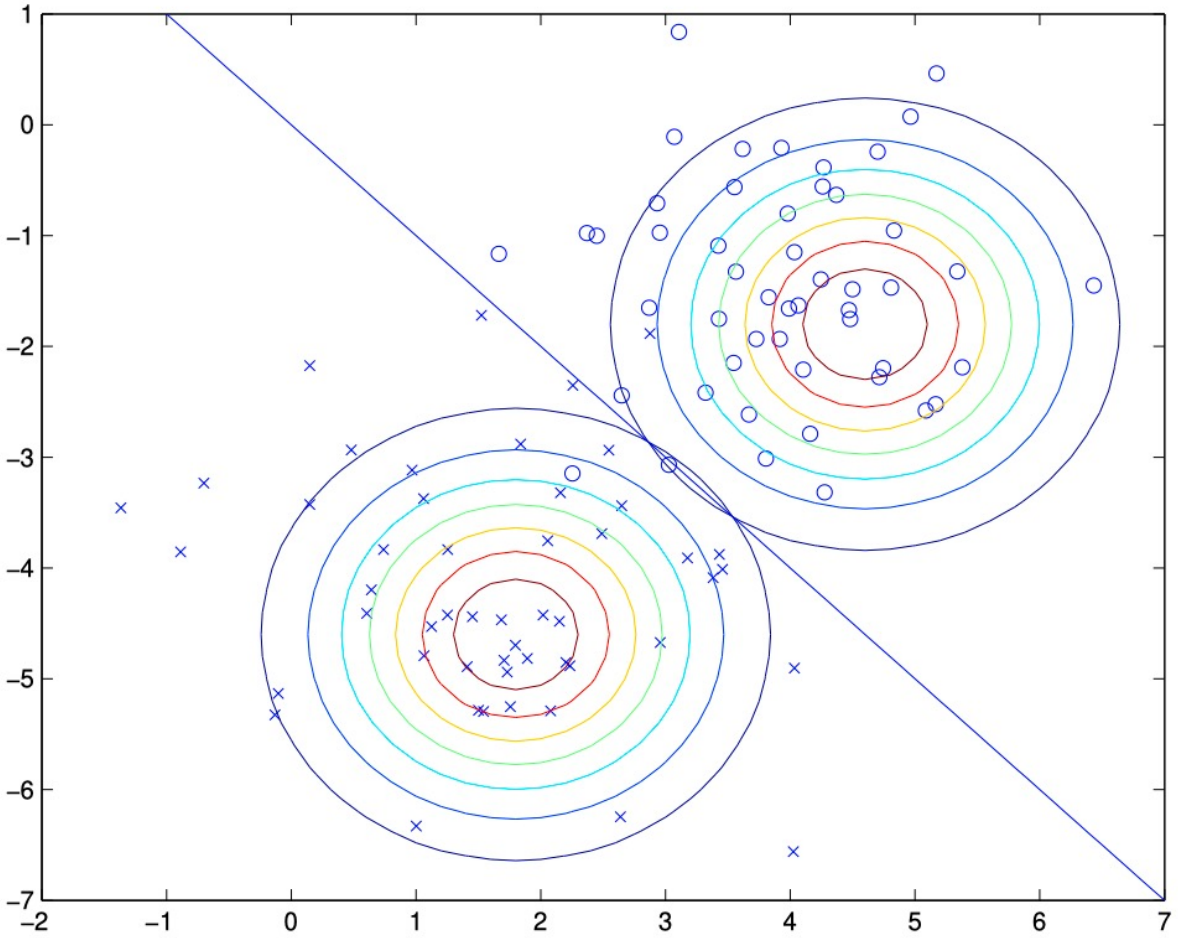
$$\begin{aligned}\ell(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^{n} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^{n} p(x^{(i)} | y^{(i)}; \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi).\end{aligned}$$

Maximum likelihood

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

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

Illustration of GDA



Decision boundary: $p(y = 1|x) = 0.5$

GDA V.S. logistic regression

- GDA has the same form as logistic regression

$$p(y = 1|x; \phi, \Sigma, \mu_0, \mu_1) = \frac{1}{1 + \exp(-\theta^T x)}$$

- How?
  - (see the offline derivation)

Which one is better?

- GDA assumes multivariate normal distribution
- p(y|x) being a logistic function does not imply p(x|y) follows multivariate normal distribution
  - Other distributions can also yield this form
- GDA makes stronger modeling assumptions
  - When the modeling assumptions are (approximate) correct
  - Is more data efficient
- Logistic regression makes weaker assumptions
  - More robust to deviations from modeling assumptions
  - When the data is indeed non-Gaussian, logistic regression is better
- In practice logistic regression is used more often

南方科技大学

MAT8034: Machine Learning

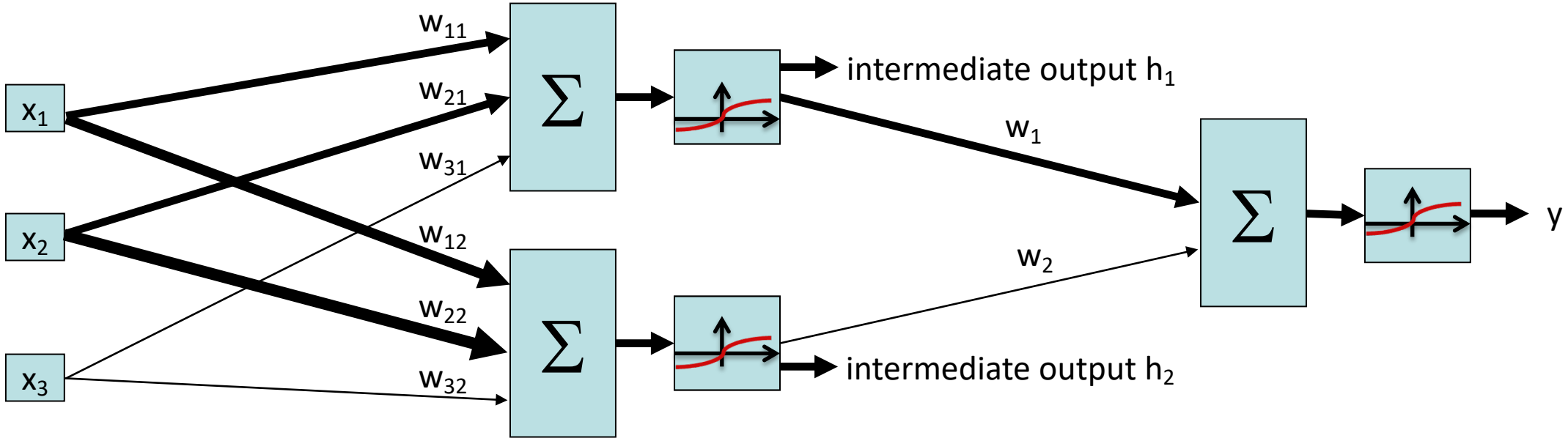
Deep Learning

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: ai.berkeley.edu; SJTU VE445; Stanford cs229;

2-Layer, 2-Neuron Neural Network



$$y = \sigma(w_1 h_1 + w_2 h_2)$$
$$= \sigma(w_1 \sigma(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2 \sigma(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))$$

Vectorization

$$
\begin{aligned}
y &= \sigma(w_1 h_1 + w_2 h_2) \\
&= \sigma(w_1 \sigma(w_{11} x_1 + w_{21} x_2 + w_{31} x_3) + w_2 \sigma(w_{12} x_1 + w_{22} x_2 + w_{32} x_3))
\end{aligned}
$$

The same equation, formatted with matrices:

$$
\begin{aligned}
&\sigma\left(\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}\right) \\
&= \sigma\left(\begin{bmatrix} w_{11} x_1 + w_{21} x_2 + w_{31} x_3 & w_{12} x_1 + w_{22} x_2 + w_{32} x_3 \end{bmatrix}\right) \\
&= \begin{bmatrix} h_1 & h_2 \end{bmatrix}
\end{aligned}
$$

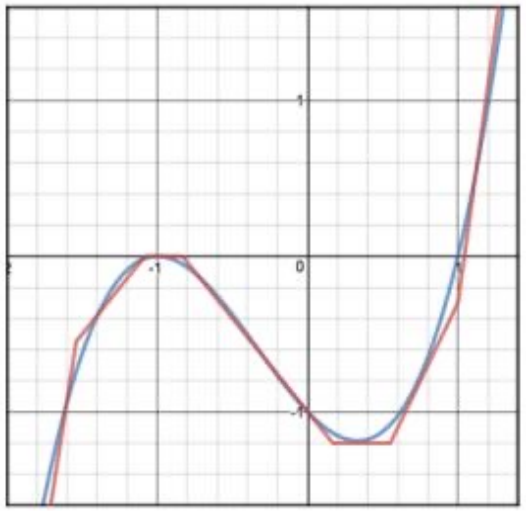
$$
\sigma\left(\begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\right) = \sigma(w_1 h_1 + w_2 h_2) = y
$$

The same equation, formatted more compactly by introducing variables representing each matrix:

$$
\sigma(x \times W_{\text{layer } 1}) = h \qquad \sigma(h \times W_{\text{layer } 2}) = y
$$

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.



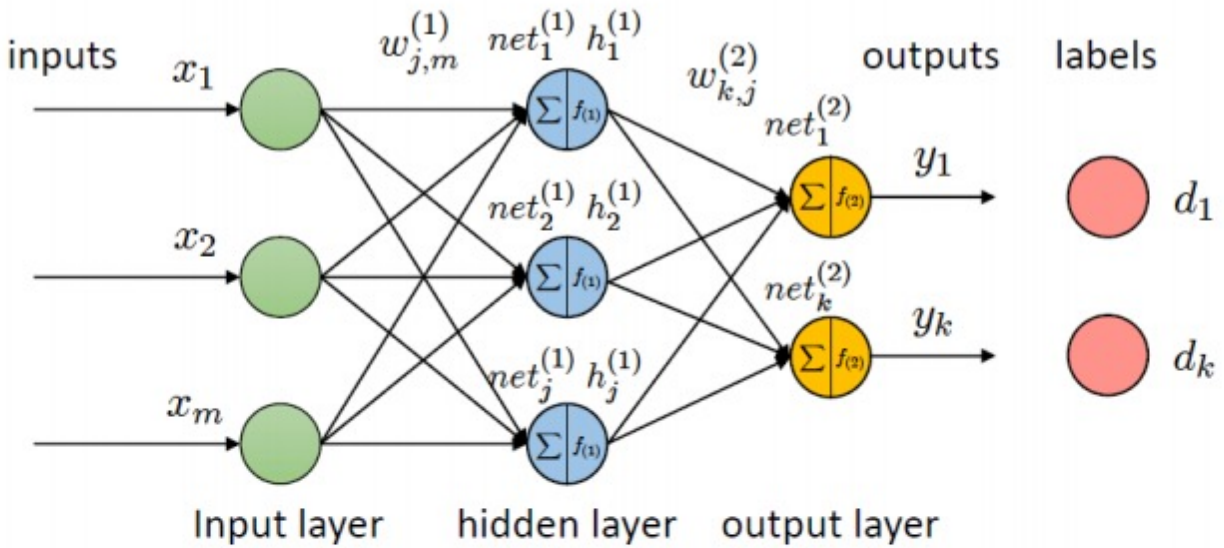
$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

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

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

Training: Backpropagation

Make a prediction



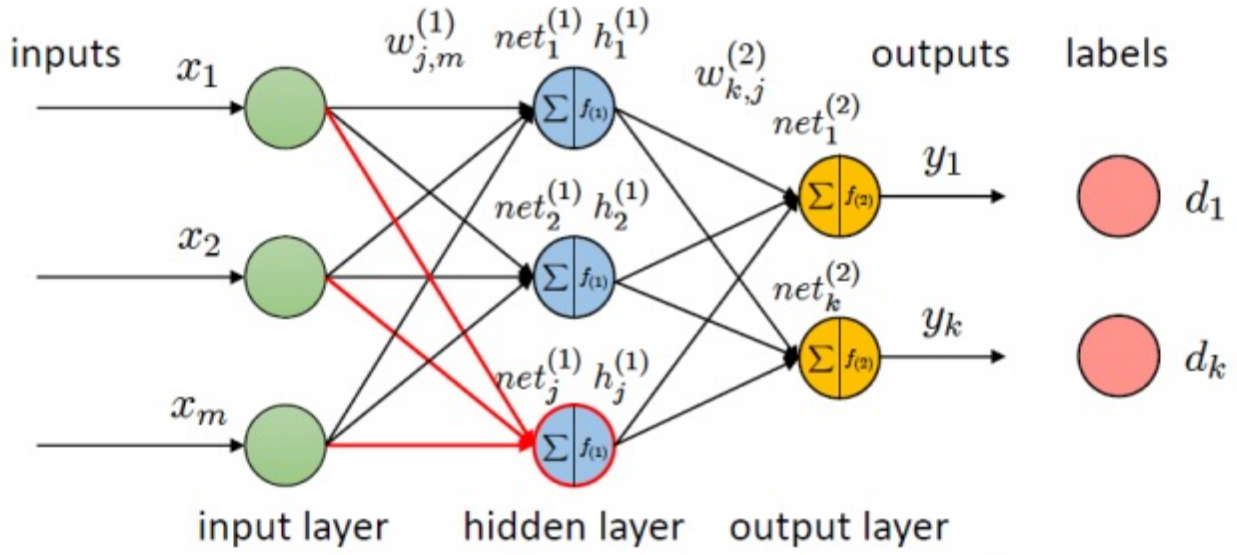
Two-layer feedforward neural network

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

Make a prediction (cont.)



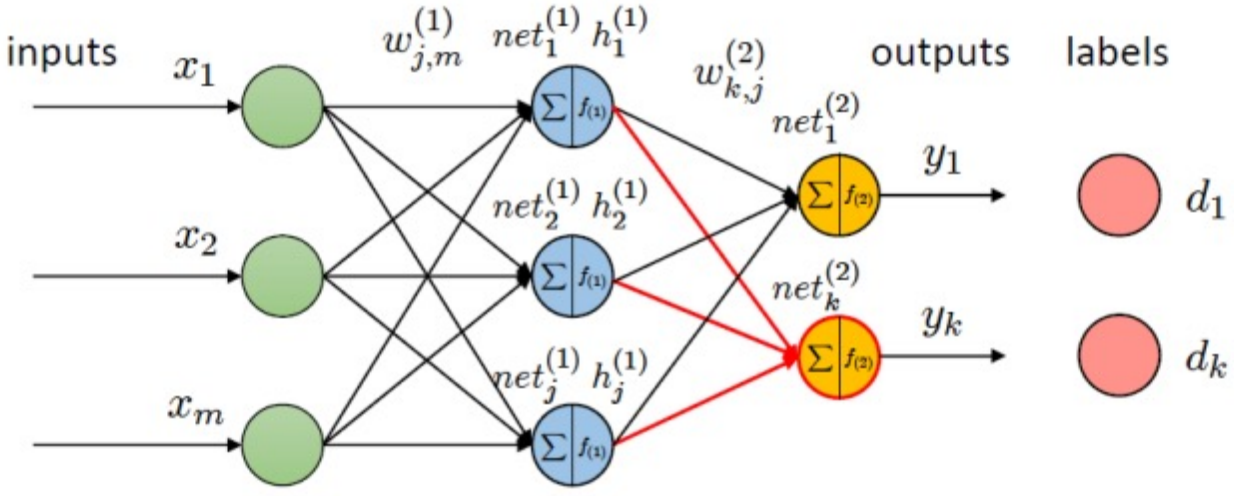
Two-layer feedforward neural network

Feed-forward prediction:

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

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

Make a prediction (cont.)



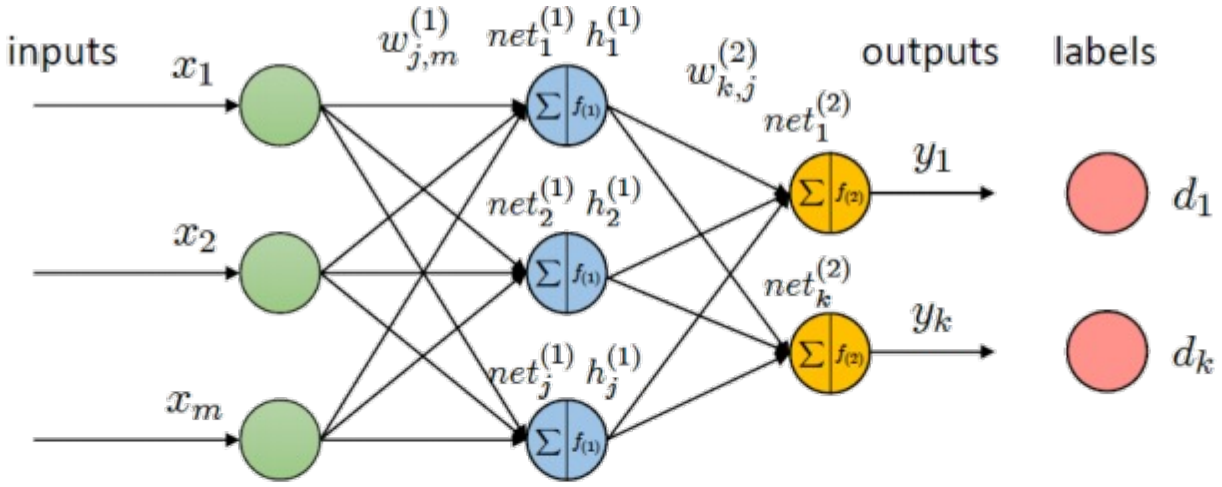
Two-layer feedforward neural network

Feed-forward prediction:

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

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 $E = \frac{1}{2}\sum_k (y_k - d_k)^2$
- $\frac{\partial E}{\partial y_k} = y_k - d_k$
- $\frac{\partial y_k}{\partial w_{k,j}^{(2)}} = f'_{(2)}(net_k^{(2)})h_j^{(1)} = y_k(1-y_k)h_j^{(1)}$
- $\Rightarrow \frac{\partial E}{\partial w_{k,j}^{(2)}} = (y_k - d_k)y_k(1-y_k)h_j^{(1)}$
- $\Rightarrow w_{k,j}^{(2)} \leftarrow w_{k,j}^{(2)} - \eta(y_k - d_k)y_k(1-y_k)h_j^{(1)}$

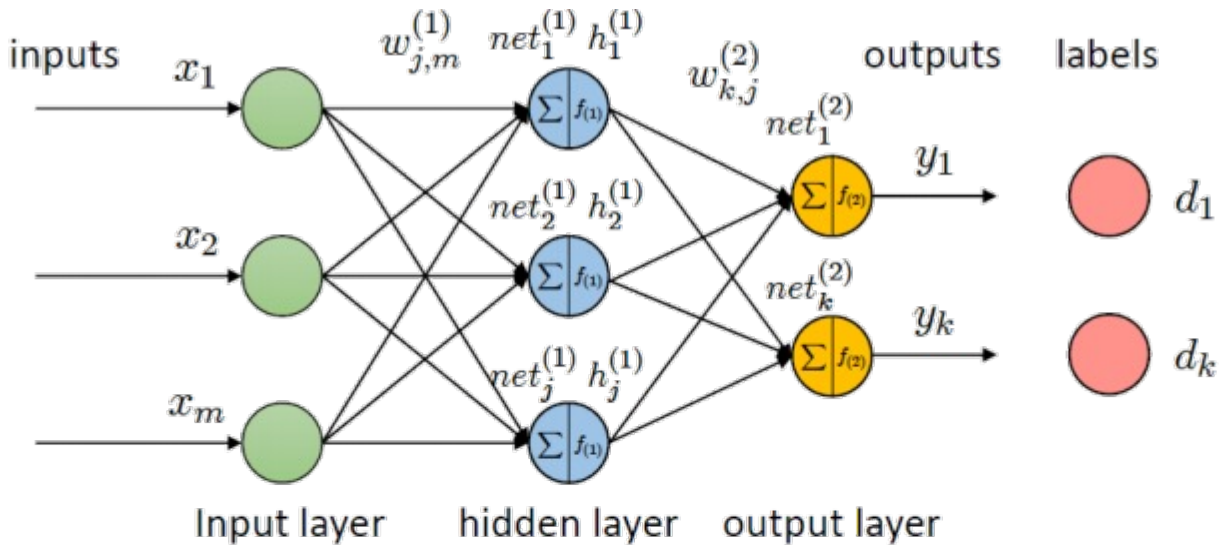
$\delta_k^{(2)}$

Feed-forward prediction:

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

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

$\delta_k^{(2)}$

$\delta_j^{(1)}$

Feed-forward prediction:

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

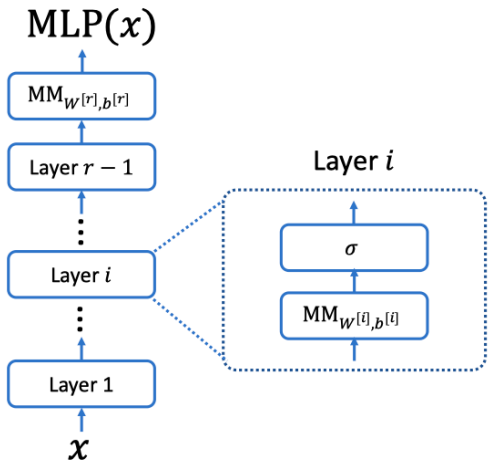
$$x = (x_1, \ldots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

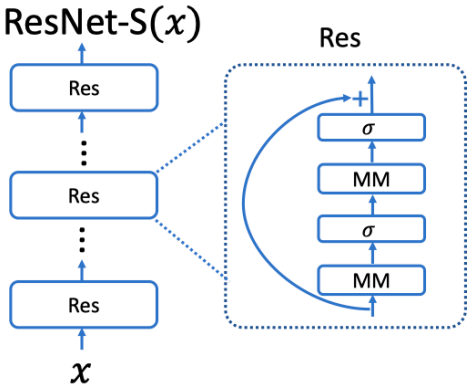
where $\quad net_j^{(1)} = \sum_m w_{j,m}^{(1)}x_m \qquad net_k^{(2)} = \sum_j w_{k,j}^{(2)}h_j^{(1)}$

Modules in modern neural networks

Residual connections

- An important network structure in CV: ResNet





- Residual connections

$$\text{Res}(z) = z + \sigma(\text{MM}(\sigma(\text{MM}(z))))$$

Classic ResNet uses convolution layers instead of vanilla matrix multiplication, and adds batch normalization between convolutions and activations.

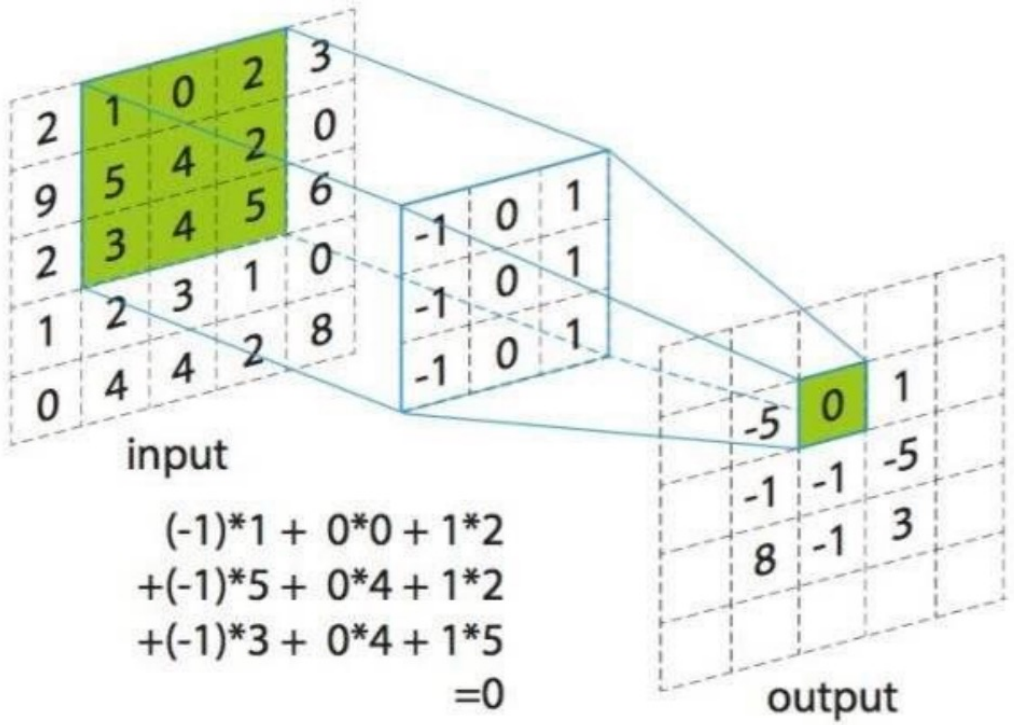
Residual connections (cont'd)

- Advantages of residual connections
  - Enable identity mapping, Improve the ability of model expression
  - Mitigate gradient disappearance, Ease training of deep networks

- Applications
  - Computer Vision (ResNet)
  - Natural Language Processing (Transformer encoder/decoder block)
  - Reinforcement Learning (policy/value networks)

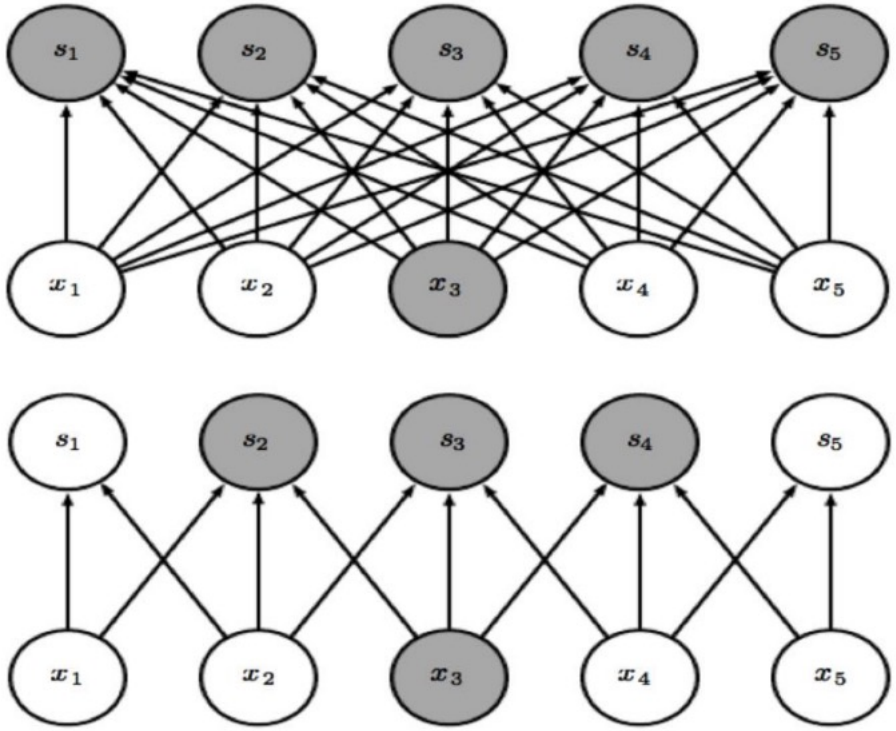
Convolutional layers

- Intuition
  - Given an input matrix (e.g. an image)
  - Use a small matrix (called filter or kernel) to screening the input at every position of the input matrix
  - Put the convolution results at corresponding positions



Convolutional layers (cont'd)

- Advantage
  - Sparse connections
  - Weight sharing



MLP
Edges: 5*5
Parameters: 5*5

Convolution
Edges: 3*3+2*2
Parameters: 3

Layer normalization

- Maps a vector to a more normalized vector
- A sub-module of the layer normalization $\longrightarrow$ $\text{LN-S}(z) = \begin{bmatrix} \frac{z_1 - \hat{\mu}}{\hat{\sigma}} \\ \frac{z_2 - \hat{\mu}}{\hat{\sigma}} \\ \vdots \\ \frac{z_m - \hat{\mu}}{\hat{\sigma}} \end{bmatrix}$
  - $\hat{\mu} = \frac{\sum_{i=1}^{m} z_i}{m}$ is the empirical mean of the vector
  - $\hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{m} (z_i - \hat{\mu}^2)}{m}}$ is the empirical standard deviation
  - Intuition: normalized to having empirical mean zero and empirical standard deviation 1

Layer normalization (cont'd)

- More general mean and variance

$$\mathrm{LN}(z) = \beta + \gamma \cdot \text{LN-S}(z) = \begin{bmatrix} \beta + \gamma \left(\frac{z_1 - \hat{\mu}}{\hat{\sigma}}\right) \\ \beta + \gamma \left(\frac{z_2 - \hat{\mu}}{\hat{\sigma}}\right) \\ \vdots \\ \beta + \gamma \left(\frac{z_m - \hat{\mu}}{\hat{\sigma}}\right) \end{bmatrix}$$

  - β, γ are learnable parameters
- Properties: Scaling-invariant

$$\mathrm{LN}(\mathrm{MM}_{\alpha W, \alpha b}(z)) = \mathrm{LN}(\mathrm{MM}_{W,b}(z)), \forall \alpha > 0.$$

- Applications
  - Transformer / BERT / GPT / RL policy networks

Attention

- Intuition: the importance of different inputs depends on the content

```
The chicken didn't cross the road because it
```

What should be the properties of "it"?

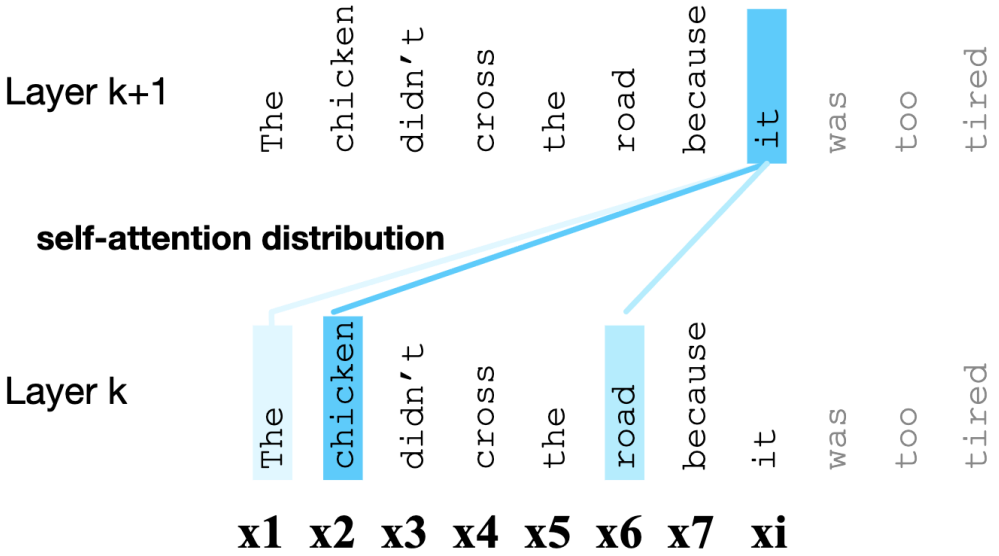
```
The chicken didn't cross the road because it was too tired
The chicken didn't cross the road because it was too wide
```

At this point in the sentence, it's probably referring to either the chicken or the street

- Attention: a mechanism that dynamically decides which inputs are important

Part of slides reference: https://web.stanford.edu/~jurafsky/slp3/slides/transformer24aug.pdf

Simplified version of attention



Given a sequence of token embeddings:

$$\mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i$$

Produce: $\mathbf{a}_i$ = a weighted sum of $\mathbf{x}_1$ through $\mathbf{x}_7$ (and $\mathbf{x}_i$)

Weighted by their similarity to $\mathbf{x}_i$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \;\; \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

An actual attention head

- High-level idea: instead of using vectors (like x_i and x_4) directly, we'll represent 3 separate roles each vector x_i plays:
  - query: what the current token is looking for
  - key: how this token can be matched
  - value: the information stored in the token
- use matrices to project each vector into a representation of its role as query, key, value
  - **query**: W^Q
  - **key**: W^K
  - **value**: W^V

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{V}}$$

Final equation for one attention head

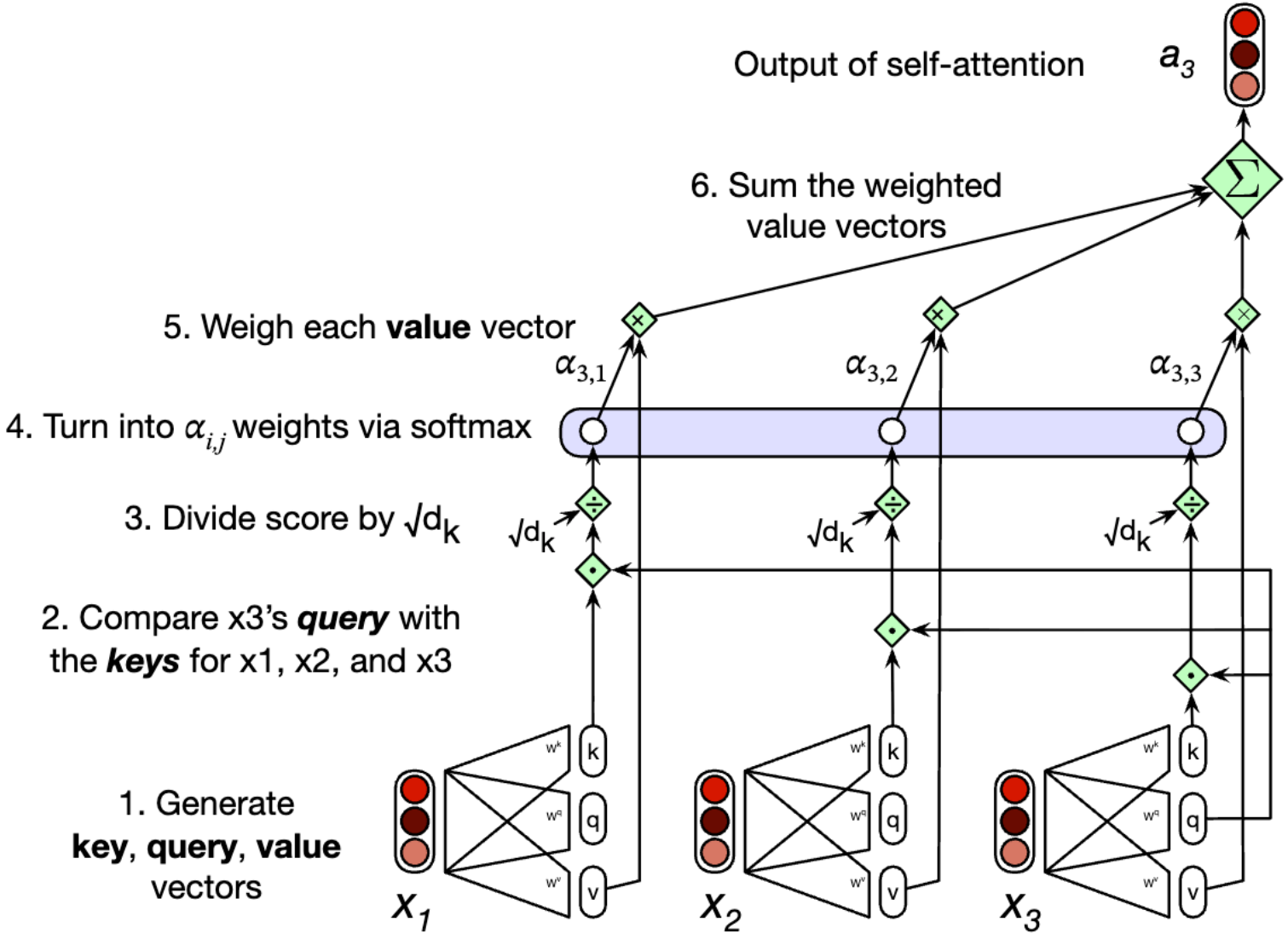
$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^{\mathbf{Q}}; \quad \mathbf{k}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{K}}; \quad \mathbf{v}_j = \mathbf{x}_j \mathbf{W}^{\mathbf{V}}$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Calculating the value of a3

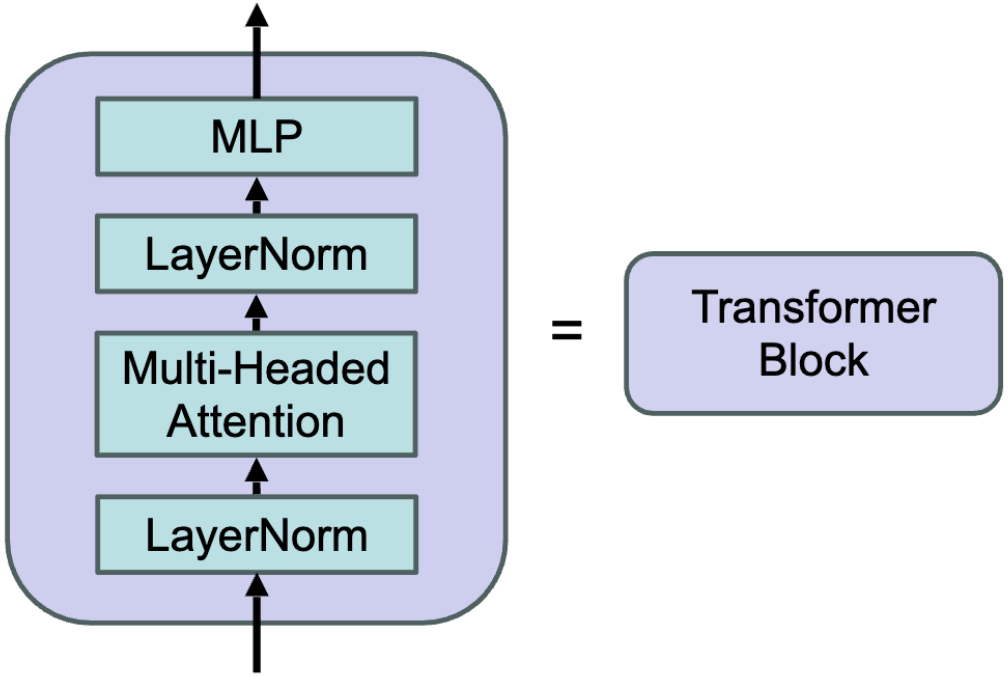


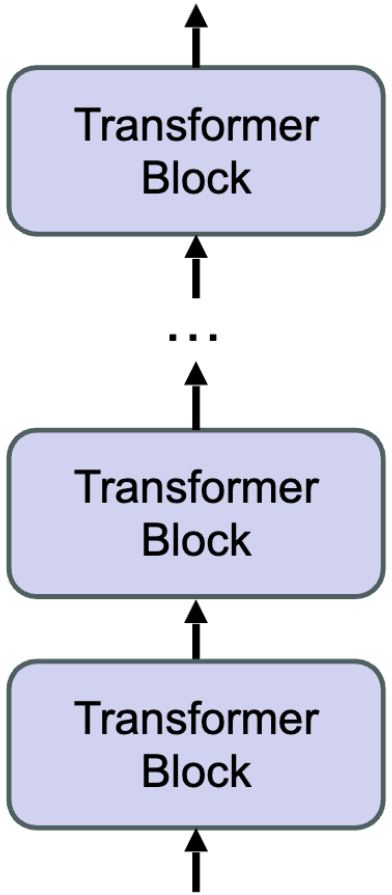
Multi-head attention

- Instead of one attention head, we'll have lots of them
- Intuition: each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

$$
\begin{aligned}
\mathbf{q}_i^c = \mathbf{x}_i\mathbf{W}^{\mathbf{Qc}};\ \ \mathbf{k}_j^c = \mathbf{x}_j\mathbf{W}^{\mathbf{Kc}};\ \ \mathbf{v}_j^c &= \mathbf{x}_j\mathbf{W}^{\mathbf{Vc}};\quad \forall\, c\ \ 1 \le c \le h \\
\text{score}^c(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\mathbf{q}_i^c \cdot \mathbf{k}_j^c}{\sqrt{d_k}} \\
\alpha_{ij}^c &= \text{softmax}(\text{score}^c(\mathbf{x}_i, \mathbf{x}_j))\ \ \forall j \le i \\
\mathbf{head}_i^c &= \sum_{j \le i} \alpha_{ij}^c \mathbf{v}_j^c \\
\mathbf{a}_i &= (\mathbf{head}^1 \oplus \mathbf{head}^2 \ldots \oplus \mathbf{head}^h)\mathbf{W}^O \\
\text{MultiHeadAttention}(\mathbf{x}_i, [\mathbf{x}_1, \cdots, \mathbf{x}_N]) &= \mathbf{a}_i
\end{aligned}
$$

Transformer Architecture





南方科技大学

MAT8034: Machine Learning

Generalization and Regularization

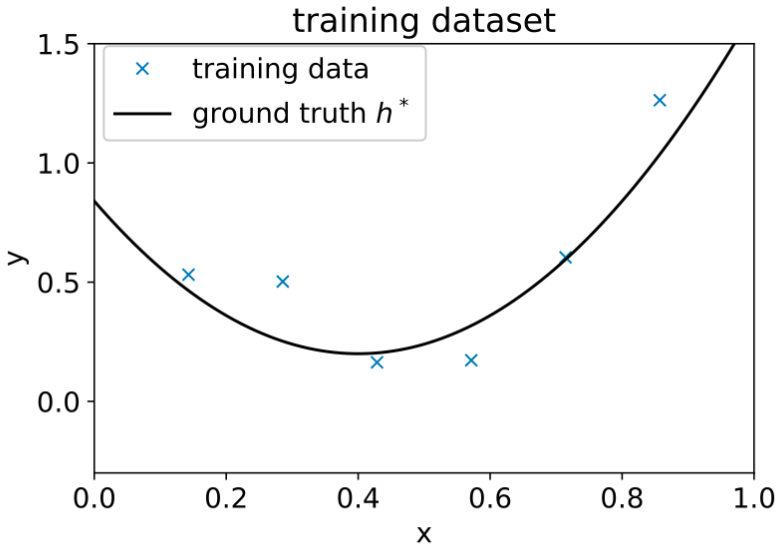
Fang Kong

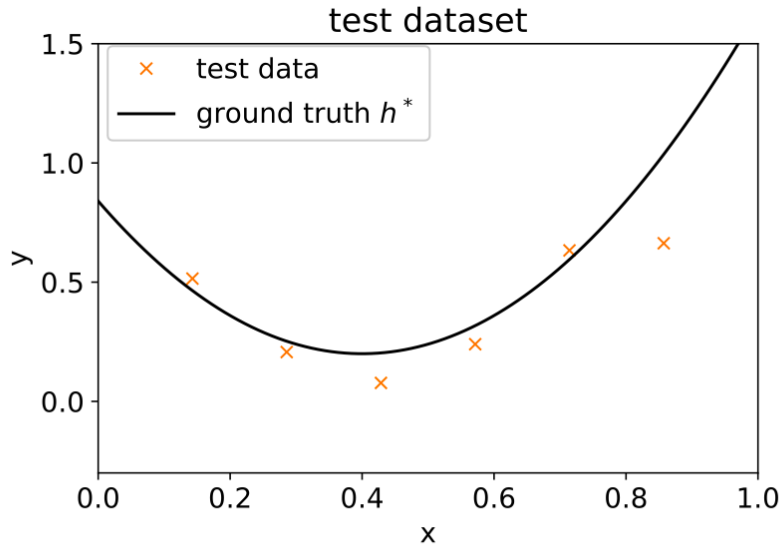
https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

Bias-variance tradeoff

Problem setting





- The training inputs are randomly chosen
- The outputs are generated by $y^{(i)} = h^{\star}(x^{(i)}) + \xi^{(i)}$
  - $h^*(\cdot)$: a quadratic function
  - $\xi^{(i)} \sim N(0, \sigma^2)$: noise
- Our goal is to recover the function $h^*(\cdot)$

How about fitting a linear model?

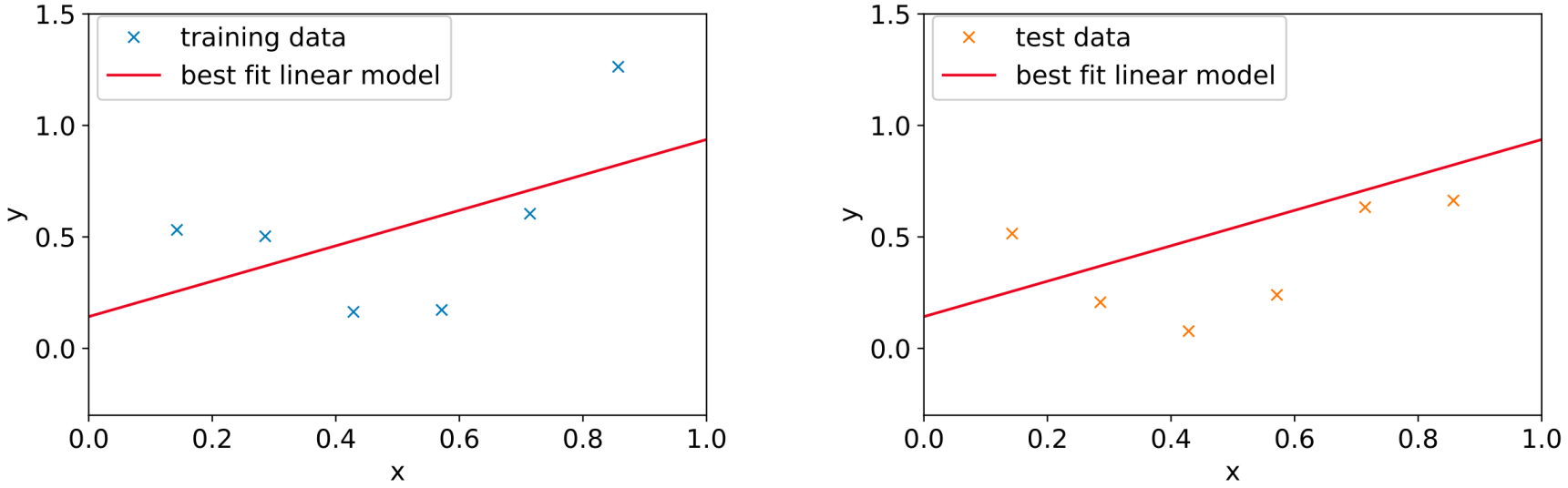


Figure 8.2: The best fit linear model has large training and test errors.

- 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)

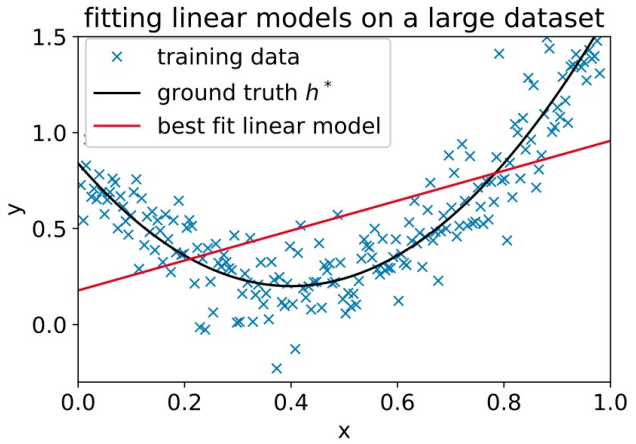


Figure 8.3: The best fit linear model on a much larger dataset still has a large training error.

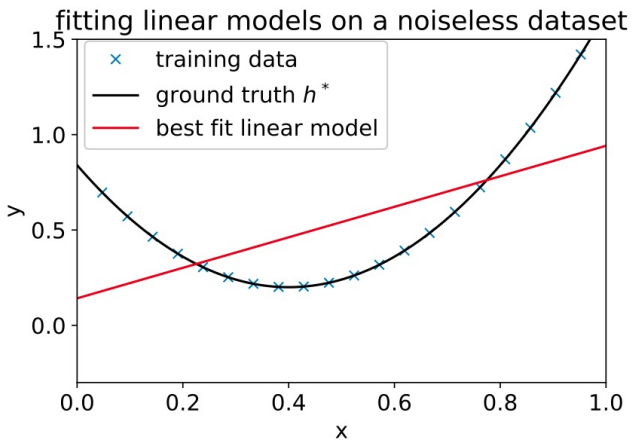


Figure 8.4: The best fit linear model on a noiseless dataset also has a large training/test error.

- 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?

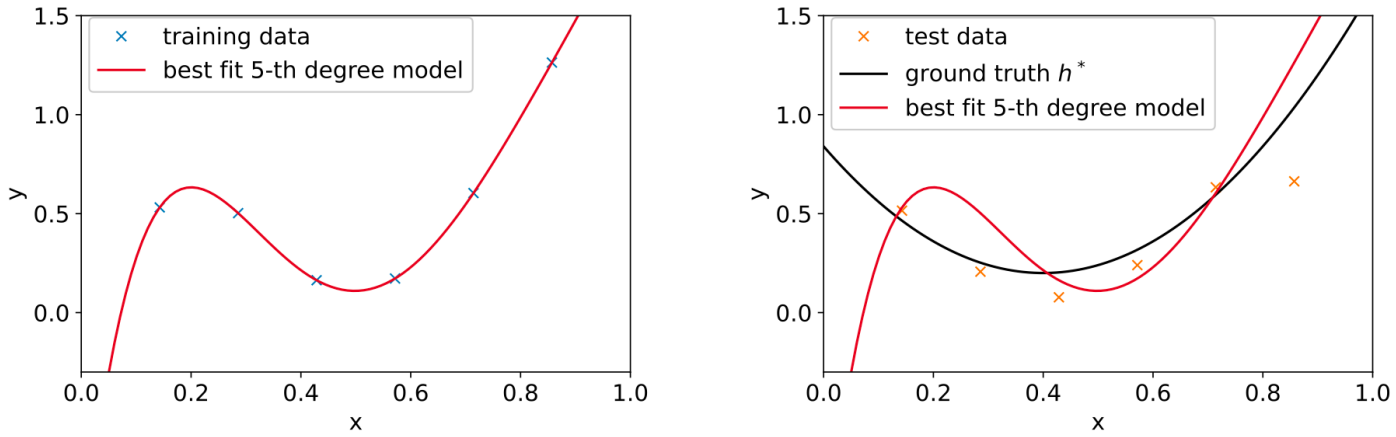


Figure 8.5: Best fit 5-th degree polynomial has zero training error, but still has a large test error and does not recover the the ground truth. This is a classic situation of overfitting.

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

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

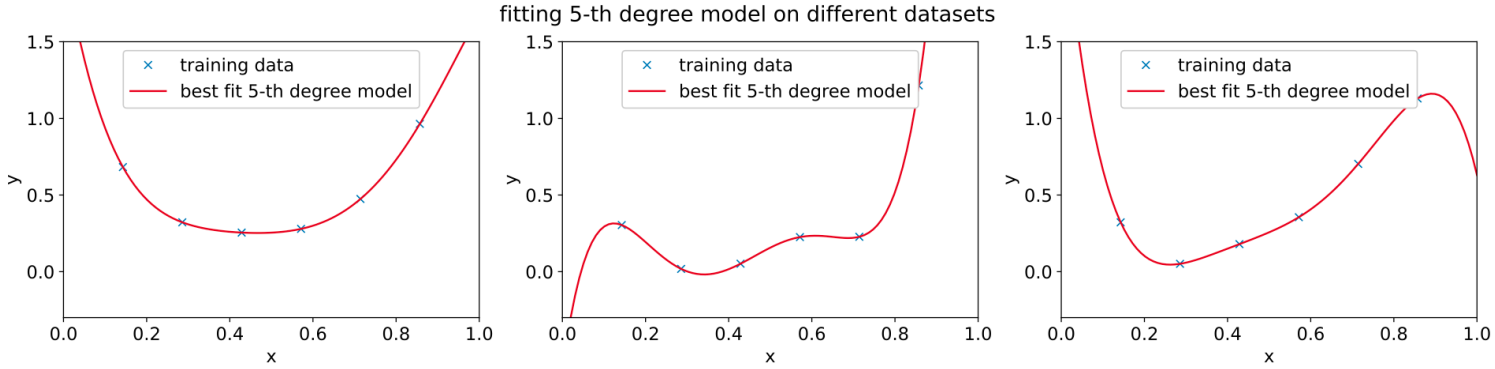


Figure 8.7: The best fit 5-th degree models on three different datasets generated from the same distribution behave quite differently, suggesting the existence of a large variance.

- 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)

A mathematical decomposition (for regression)

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

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

Sample complexity bounds

Useful lemmas

- **Lemma.** (The union bound). Let $A_1, A_2, \ldots, A_k$ be k different events (that may not be independent). Then

$$P(A_1 \cup \cdots \cup A_k) \leq P(A_1) + \ldots + P(A_k).$$

- **Lemma.** (Hoeffding inequality) Let $Z_1, \ldots, Z_n$ be n independent and identically distributed (iid) random variables drawn from a Bernoulli(ϕ) distribution. I.e., $P(Z_i = 1) = \phi$, and $P(Z_i = 0) = 1 - \phi$. Let $\hat{\phi} = (1/n) \sum_{i=1}^{n} Z_i$ be the mean of these random variables, and let any $\gamma > 0$ be fixed. Then

$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 n)$$

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

Problem setting (cont'd)

- Consider the linear classification $h_\theta(x) = 1\{\theta^\top x \geq 0\}$
- Objective: minimize the training error

$$\hat{\theta} = \arg\min_\theta \hat{\varepsilon}(h_\theta)$$

$$\hat{h} = h_{\hat{\theta}}$$

empirical risk minimization

- In learning theory, it will be useful to abstract away from the specific parameterization of hypotheses
- Define the hypothesis class $\mathcal{H}$, for linear classification

$$\mathcal{H} = \{h_\theta : h_\theta(x) = 1\{\theta^T x \geq 0\}, \theta \in \mathbb{R}^{d+1}\}$$

Problem setting (cont'd)

- ERM becomes finding $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$
- For simplicity, first consider the finite hypothesis set

$$\mathcal{H} = \{h_1, \ldots, h_k\}$$

- Now, show the guarantee for the generalization error of $\hat{h}$
  - 1. $\forall h, \hat{\varepsilon}(h)$ is a reliable estimate of $\varepsilon(h)$
  - 2. $\hat{h}$ guarantees good generalization error

Guarantee for a fixed hypothesis function

- Fix any hypothesis function $h_i \in \mathcal{H}$
- Define $Z_j = 1\{h_i(x^j) \neq y^j\}$
- The training error is

$$\hat{\varepsilon}(h_i) = \frac{1}{n}\sum_{j=1}^{n} Z_j$$

- The empirical mean of n random variables with expectation $\varepsilon(h_i)$
- Applying Hoeffding inequality,

$$P(|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) \leq 2\exp(-2\gamma^2 n)$$

Guarantee for any hypothesis function

- $$\begin{aligned} P(\exists h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) &= P(A_1 \cup \cdots \cup A_k) \\ &\leq \sum_{i=1}^{k} P(A_i) \\ &\leq \sum_{i=1}^{k} 2\exp(-2\gamma^2 n) \\ &= 2k\exp(-2\gamma^2 n) \end{aligned}$$

- Thus $$\begin{aligned} P(\neg\exists h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| > \gamma) &= P(\forall h \in \mathcal{H}.|\varepsilon(h_i) - \hat{\varepsilon}(h_i)| \leq \gamma) \\ &\geq 1 - 2k\exp(-2\gamma^2 n) \end{aligned}$$

Questions

- How large must n be before we can guarantee that with probability at least $1 - \delta$, training error will be within γ of generalization error? (sample complexity)

- What is the distance between the training error and generalization error with training set size n and confidence δ?

Guarantee for the output hypothesis function

- Recall $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$
- Define the best hypothesis is $h^* = \arg\min_{h \in \mathcal{H}} \varepsilon(h)$
- Then
$$\begin{aligned} \varepsilon(\hat{h}) &\leq \hat{\varepsilon}(\hat{h}) + \gamma \\ &\leq \hat{\varepsilon}(h^*) + \gamma \\ &\leq \varepsilon(h^*) + 2\gamma \end{aligned}$$
- If uniform convergence occurs, then the generalization error of h is at most 2γ worse than the best possible hypothesis in $\mathcal{H}$!

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

Corollary of sample complexity

- **Corollary.** Let $|\mathcal{H}| = k$, and let any δ, γ be fixed. Then for $\varepsilon(\hat{h}) \leq \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$ to hold with probability at least $1 - \delta$, it suffices that

$$\begin{aligned} n &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right), \end{aligned}$$

Extension to infinite $\mathcal{H}$: Intuition

- Usually the hypothesis set is infinite
  - For example, the linear function set contains a infinite number of parameters

- Suppose $\mathcal{H}$ is parameterized by d real numbers
- The computer uses 64 bits to represent a floating point number
- $\mathcal{H}$ contains 2^{64d} different hypotheses
- Existing results show that with fixed γ, δ

$$n \geq O\left(\frac{1}{\gamma^2}\log\frac{2^{64d}}{\delta}\right) = O\left(\frac{d}{\gamma^2}\log\frac{1}{\delta}\right) = O_{\gamma,\delta}(d)$$

VC dimension

- Shatter

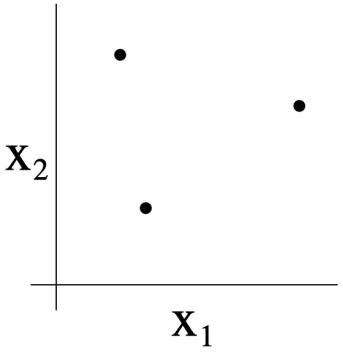
Given a set $S = \{x^{(i)}, \ldots, x^{(\mathbf{D})}\}$ (no relation to the training set) of points $x^{(i)} \in \mathcal{X}$, we say that $\mathcal{H}$ **shatters** S if $\mathcal{H}$ can realize any labeling on S. I.e., if for any set of labels $\{y^{(1)}, \ldots, y^{(\mathbf{D})}\}$, there exists some $h \in \mathcal{H}$ so that $h(x^{(i)}) = y^{(i)}$ for all $i = 1, \ldots \mathbf{D}$.

- VC dimension

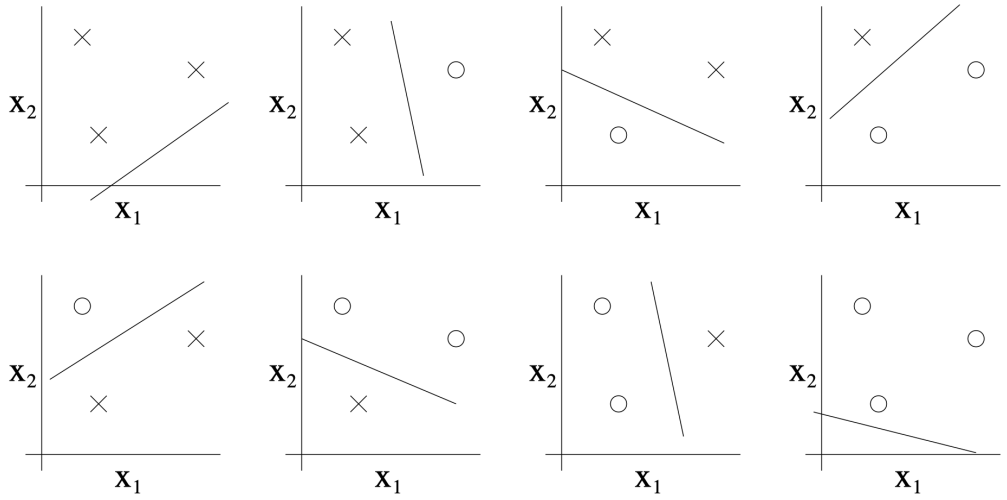
Given a hypothesis class $\mathcal{H}$, we then define its **Vapnik-Chervonenkis dimension**, written $\mathrm{VC}(\mathcal{H})$, to be the size of the largest set that is shattered by $\mathcal{H}$. (If $\mathcal{H}$ can shatter arbitrarily large sets, then $\mathrm{VC}(\mathcal{H}) = \infty$.)

VC dimension: illustration

- Can the set $\mathcal{H}$ of linear classifiers in two dimensions shatter the set below?

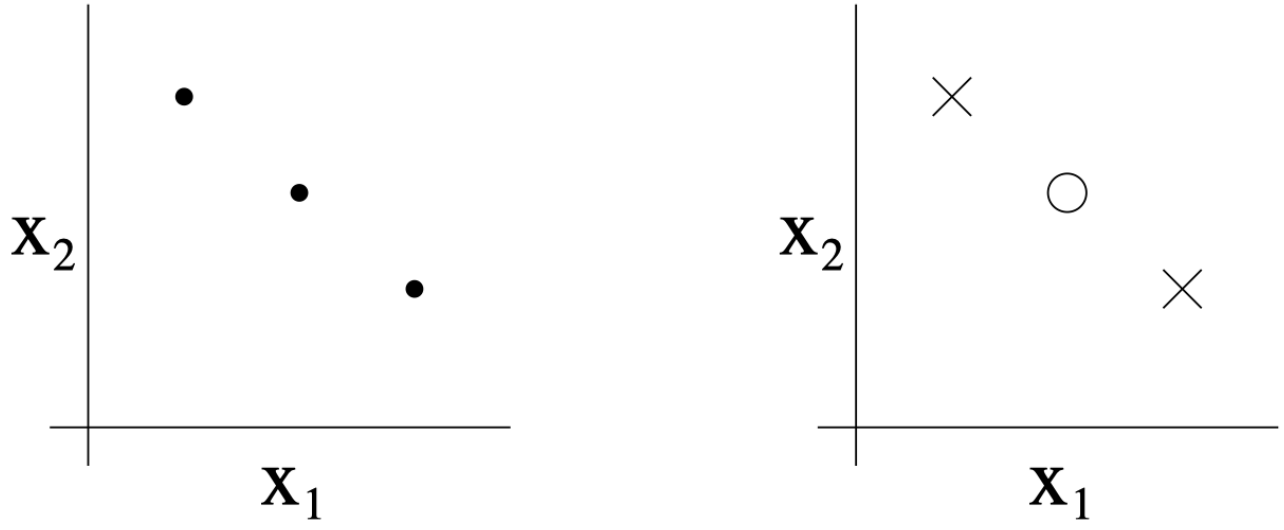


- For any labeling, $\mathcal{H}$ can correctly classify



VC dimension: illustration (cont'd)

- In order to prove that VC($\mathcal{H}$) is at least D, we need to show only that there's **at least one** set of size D that H can shatter (not every set of size D)



Convergence results

- **Theorem.** Let $\mathcal{H}$ be given, and let $\mathbf{D} = \mathrm{VC}(\mathcal{H})$. Then with probability at least $1 - \delta$, we have that for all $h \in \mathcal{H}$,

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq O\left(\sqrt{\frac{\mathbf{D}}{n}\log\frac{n}{\mathbf{D}} + \frac{1}{n}\log\frac{1}{\delta}}\right).$$

  Thus, with probability at least $1 - \delta$, we also have that:

$$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + O\left(\sqrt{\frac{\mathbf{D}}{n}\log\frac{n}{\mathbf{D}} + \frac{1}{n}\log\frac{1}{\delta}}\right).$$

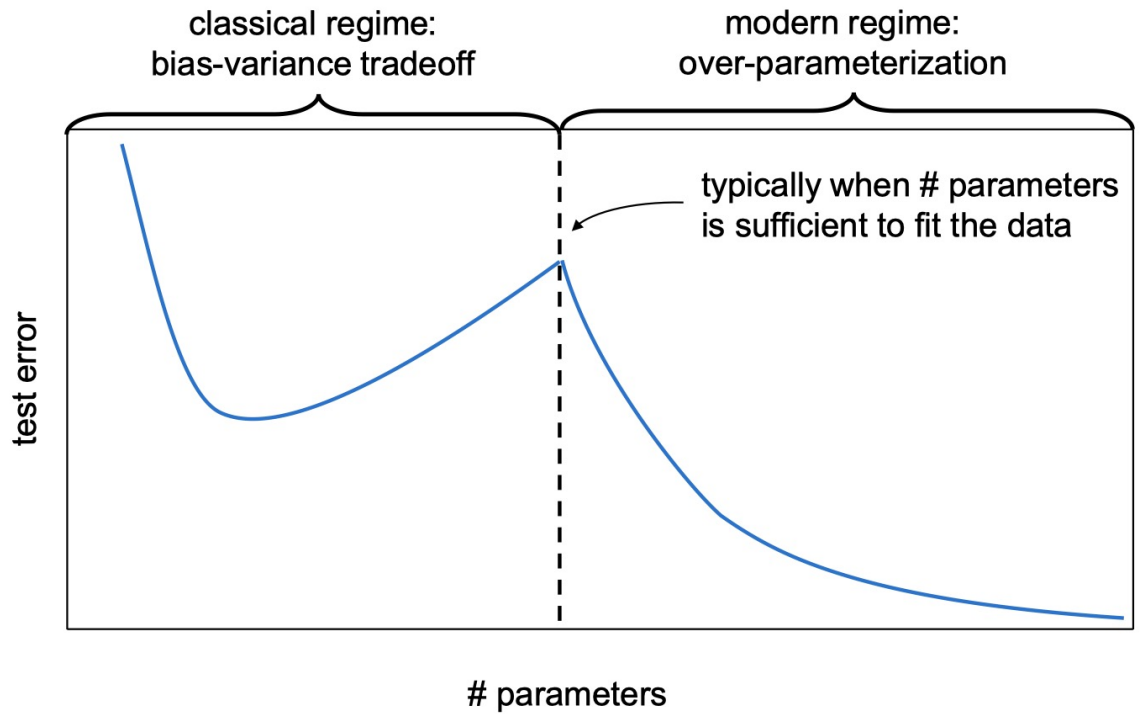
Usually the VC dimension is roughly linear in the number of parameters

- **Corollary.** For $|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \gamma$ to hold for all $h \in \mathcal{H}$ (and hence $\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma$) with probability at least $1 - \delta$, it suffices that $n = O_{\gamma,\delta}(\mathbf{D})$.

The double descent phenomenon

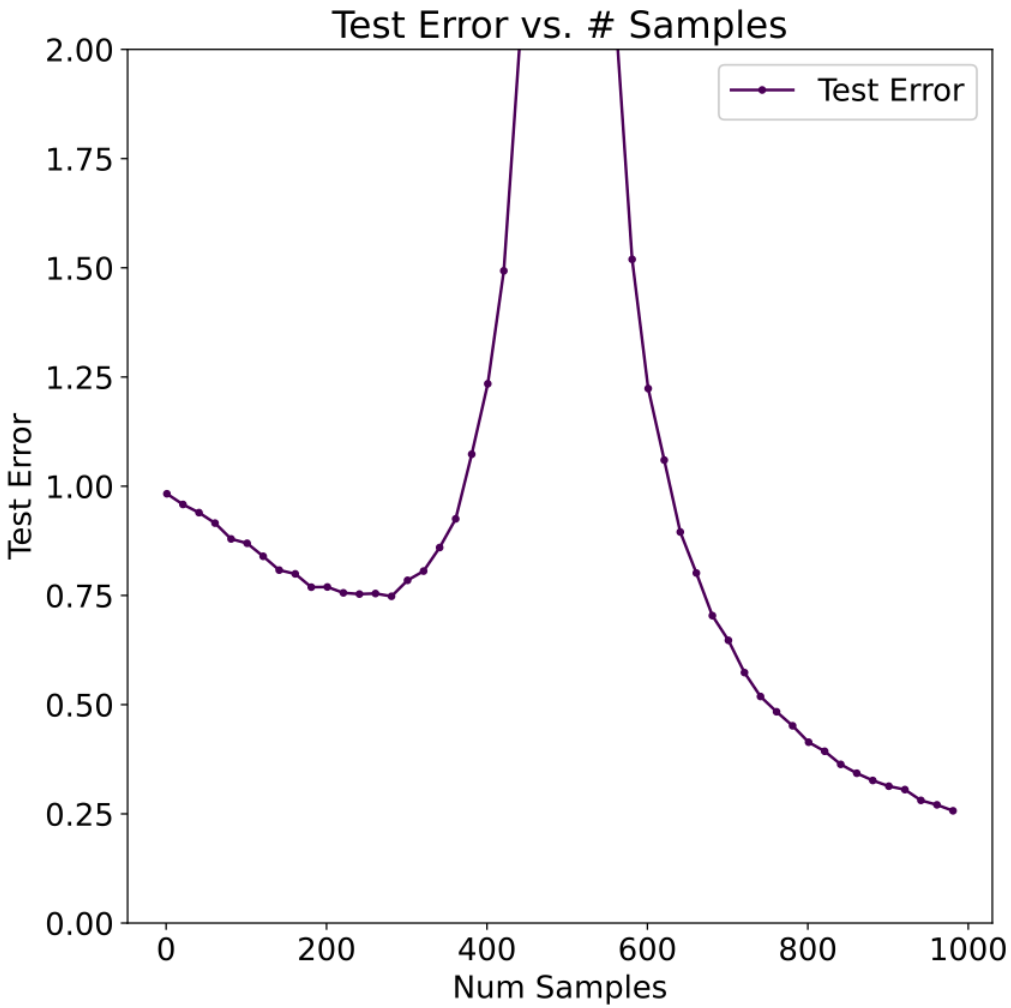
Model-wise double descent

- 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



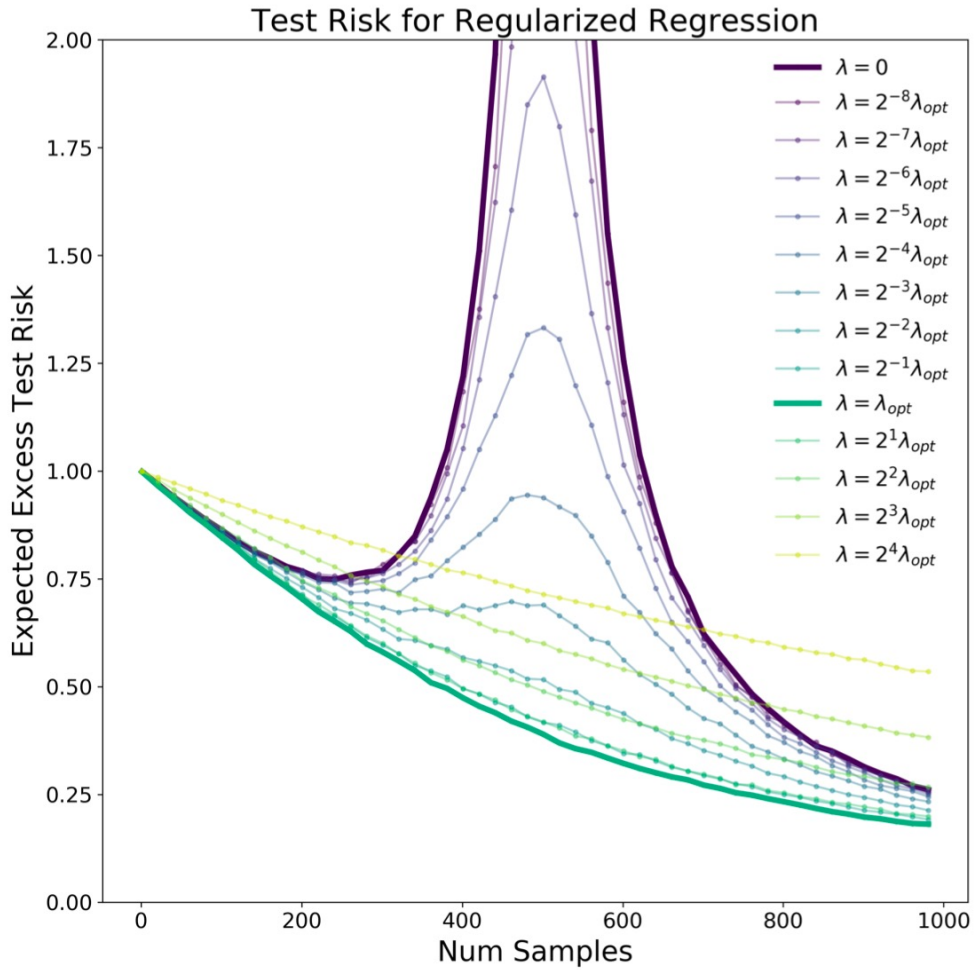
Sample-wise double descent

- Recent work observes that the test error is **not monotonically decreasing** when the sample size increases
  - The test error first decreases
  - Then increases and peaks around when the number of examples is similar to the number of parameters ($n \approx d$)
  - And then decreases again
  - Sample-wise double descent and model-wise double descent are essentially describing similar phenomena—the test error is peaked when $n \approx d$



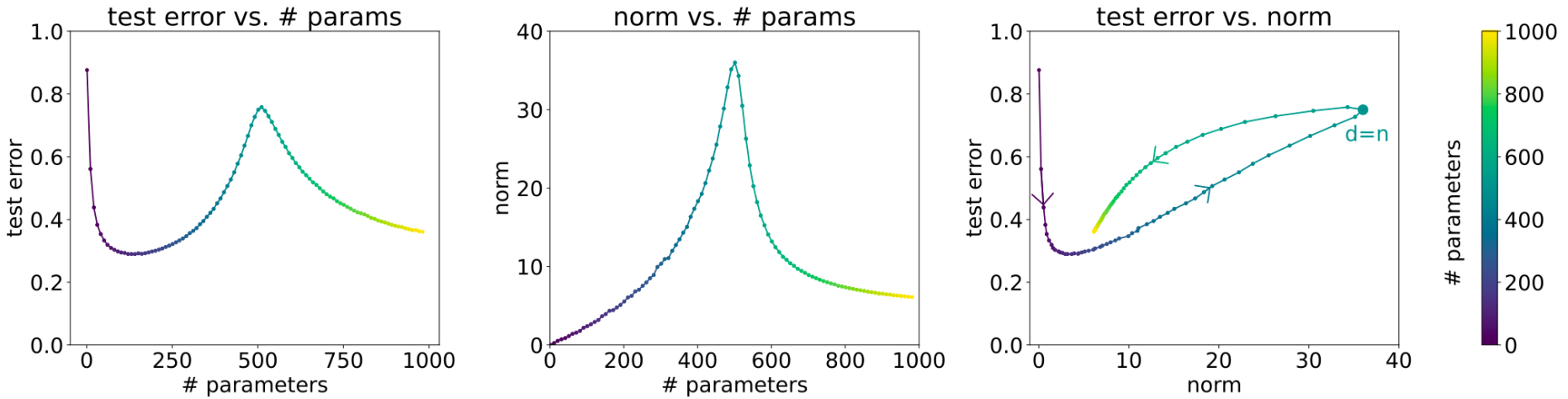
Regularization

- Using the optimal regularization parameter λ (optimally tuned for each n, shown in green solid curve) mitigates double descent



Complexity measure of the model

- The double descent phenomenon has been observed when the model complexity is measured by the number of parameters
- It is unclear if and when the number of parameters is the best complexity measure of a model



Implicit regularization effect

Explanation for overparameterization

- A typical explanation
  - Commonly-used optimizers such as gradient descent provide an implicit regularization effect
  - Intuition: even in the overparameterized regime and with an unregularized loss function, the model is still implicitly regularized, and thus exhibits a better test performance than an arbitrary solution that fits the data.

Intuition of implicit regularization effect

- In most classic settings
  - The optimal solution is unique
  - Any reasonable optimizer should converge to this point
- In deep learning
  - There are usually more than one (approximate) global minimum
  - Different optimizers may converge to different global minima
  - Though they have similar training loss
    - The solution may have dramatically different generalization performance

Implicitly regularization of GD

定理 8.3 (过参数线性回归的隐式正则化). 在过参数线性回归 (*MSE* 损失) 中，如果使用 *0* 初始化的梯度下降法且训练误差降为 *0*，则最终优化到的解 $\hat{\boldsymbol{w}}$ 是最小范数解，即在集合 $\{\boldsymbol{w} : X\boldsymbol{w} = Y\}$ 中，$\hat{\boldsymbol{w}}$ 具有最小的二范数。

- Initialization: $w_0 = 0 = X^\top 0$
- Update: $w_{t+1} = w_t - \eta X^\top (Y - X^\top \mathrm{w}_t)$
- Final convergence: $\widehat{w} = X^\top v$
- Training loss = 0 $\Longrightarrow Y = X\widehat{w} = XX^\top v \Longrightarrow \widehat{w} = X^\top (XX^\top)^{-1} Y$
- For all other solution w' satisfying $Y = Xw'$
  - $X(\mathrm{w}' - \widehat{w}) = 0 \implies \widehat{w}(\mathrm{w}' - \widehat{w}) = 0$
  - It holds that $\|w'\| - \|\widehat{w}\| \geq 0$

https://www.tengjiaye.com/book/TML.pdf#page=60.15

南方科技大学

MAT8034: Machine Learning

Clustering and the K-means Algorithm

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

Unsupervised learning

- In previous lectures, we consider the supervised learning with training set

$$S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$$

- Now, consider the unsupervised learning with training set

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

- Hope to group the data into a few cohesive “clusters”

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

- Is the k-means algorithm guaranteed to converge?

- The procedure of K-means (in each loop):
  - Fix cluster centroids μ, minimize the distance between $x^{(i)}$ and $\mu^{c(i)}$ by optimizing $c(i)$
  - Fix $c(i)$, minimize the distance between $x^{(i)}$ and $\mu^{c(i)}$ by optimizing μ

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

Convergence analysis (cont'd)

- Define the distortion function

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

  - J is a non-convex function, and so coordinate descent on J is not guaranteed to converge to the global minimum
  - K-means can be susceptible to local optima
  - It typically performs well
  - Tricks: run many with different initial values, pick the one with the lowest distortion J

南方科技大学

MAT8034: Machine Learning

EM Algorithms

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credit: Stanford CS229

Outline

- EM for the mixture of Gaussians
- Jensen's inequality
- General EM algorithms

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

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

Relaxation: If we know the class label (cont'd)

- The log-likelihood becomes

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

- Zero the derivatives and get

$$\begin{aligned}
\phi_j &= \frac{1}{n}\sum_{i=1}^{n} 1\{z^{(i)} = j\}, \\
\mu_j &= \frac{\sum_{i=1}^{n} 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{n} 1\{z^{(i)} = j\}}, \\
\Sigma_j &= \frac{\sum_{i=1}^{n} 1\{z^{(i)} = j\}(x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} 1\{z^{(i)} = j\}}
\end{aligned}$$

How to solve with unknown z^i?

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

Tool: Jensen's inequality

Convex functions

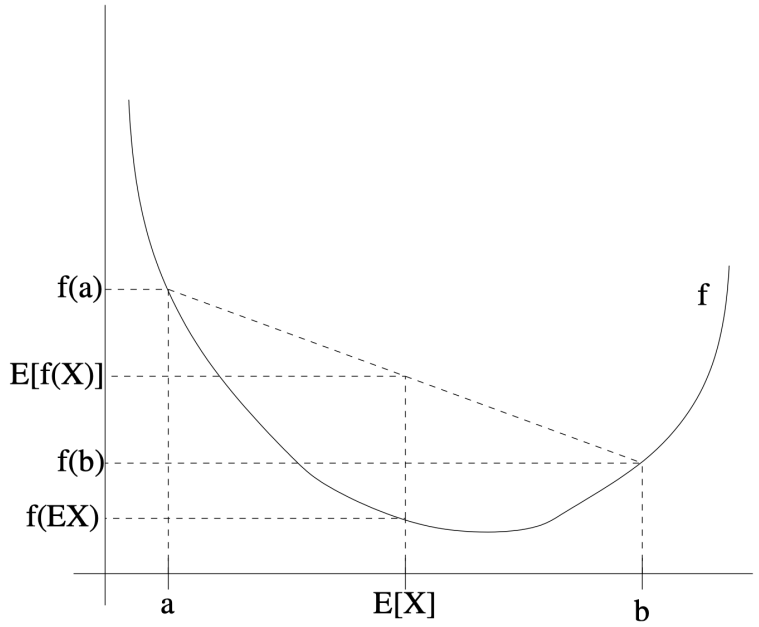
- Definition (convex functions)
  - f is a convex function if $f''(x) \geq 0$ (for all $x \in R$)
  - f is a strictly convex function if $f''(x) > 0$ (for all $x \in R$)
  - If taking vector-valued inputs, f is a convex function if its hessian H is positive semi-definite

Jensen's inequality

- **Theorem.** Let f be a convex function, and let X be a random variable. Then:

$$\mathrm{E}[f(X)] \geq f(\mathrm{E}X).$$

Moreover, if f is strictly convex, then $\mathrm{E}[f(X)] = f(\mathrm{E}X)$ holds true if and only if $X = \mathrm{E}[X]$ with probability 1 (i.e., if X is a constant).



Concave functions

- Definition (concave functions)
  - f is [strictly] concave if and only if −f is [strictly] convex (i.e., $f''(x) \leq 0$ or $H \leq 0$).
  - Jensen's inequality also holds for concave functions f with $E[f(X)] \leq f(EX)$

General EM algorithms

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

- For any distribution Q, we have the lower bound

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

- How to choose Q?
  - Try to make the lower-bound tight at that value of θ
  - Hope the inequality hold with equality

How?

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

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

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

}

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

Formal procedure of EM (cont'd)

- Repeat until convergence {

When the change between θ^{t+1} and θ^{t} is small enough

(E-step) For each i, set

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

(M-step) Set

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

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

}

Other interpretation of EM/ELBO

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

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

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

南方科技大学

MAT8034: Machine Learning

Markov Decision Processes

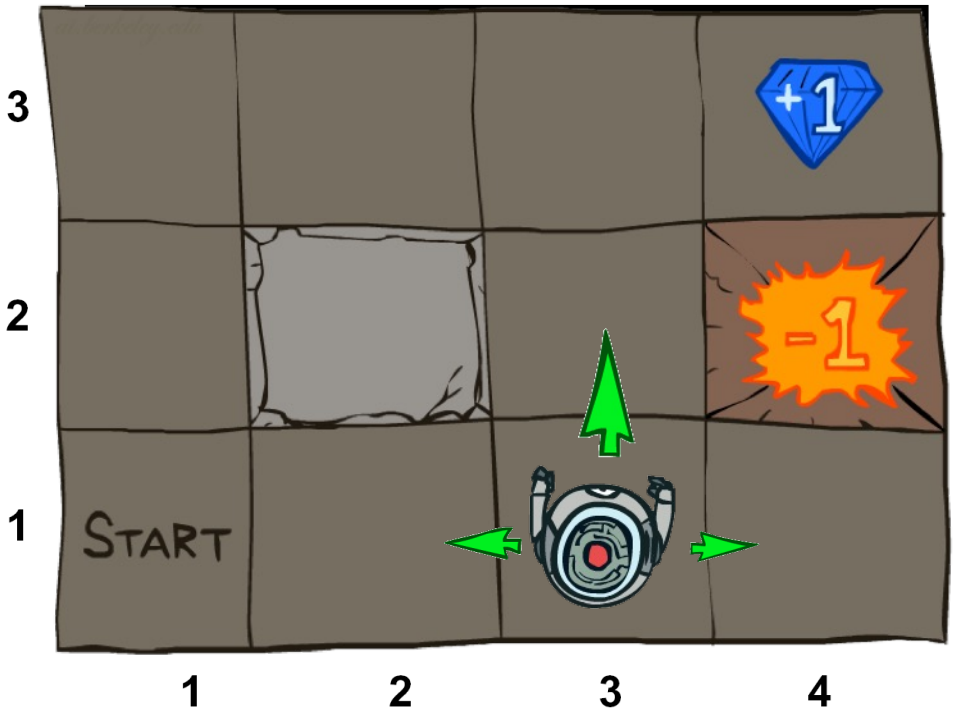
Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Slide credits: ai.berkeley.edu

Markov Decision Processes

- An MDP is defined by:
  - A set of states $s \in S$
  - A set of actions $a \in A$
  - A transition function T(s, a, s')
    - Probability that a from s leads to s', i.e., P(s' | s, a)
    - Also called the model or the dynamics
  - A reward function R(s, a, s')
    - Sometimes just R(s) or R(s')
  - A start state
  - Maybe a terminal state



[Demo – gridworld manual intro (L8D1)]

What is Markov about MDPs?

- "Markov" generally means that given the present state, the future and the past are independent

- For Markov decision processes, "Markov" means action outcomes depend only on the current state

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1}, \ldots S_0 = s_0)$$

$$=$$

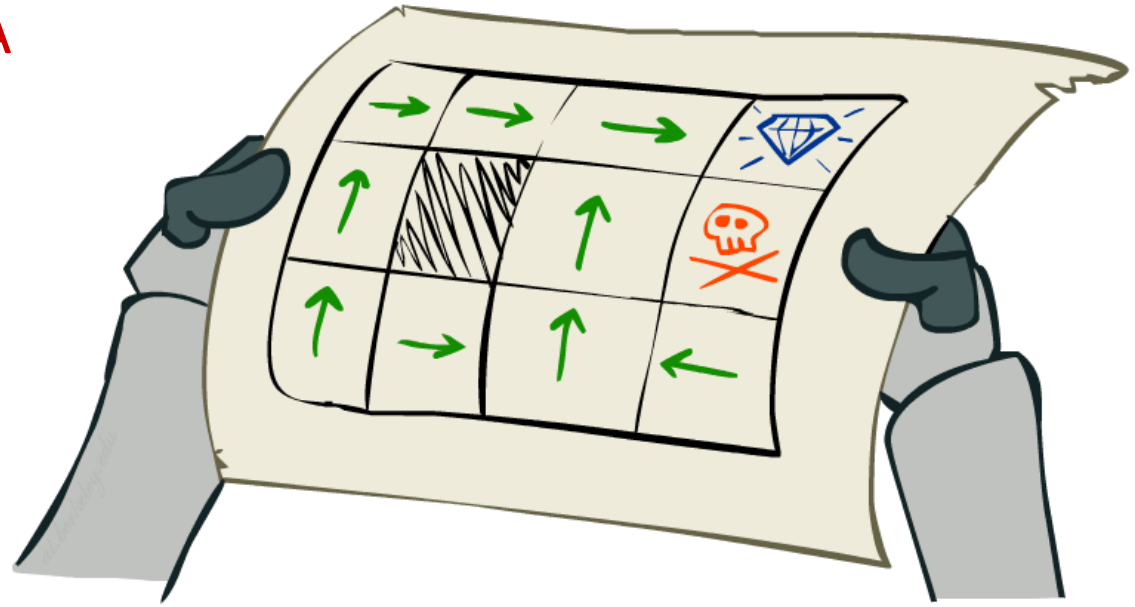
$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

- This is just like search, where the successor function could only depend on the current state (not the history)

Andrey Markov
(1856-1922)

Policies

- For MDPs, we want an optimal policy $\pi^*: S \rightarrow A$
  - A policy π gives an action for each state
  - An optimal policy is one that maximizes expected utility if followed



Optimal policy when R(s, a, s') = -0.03
for all non-terminals s

Optimal Quantities

- The value (utility) of a state s:

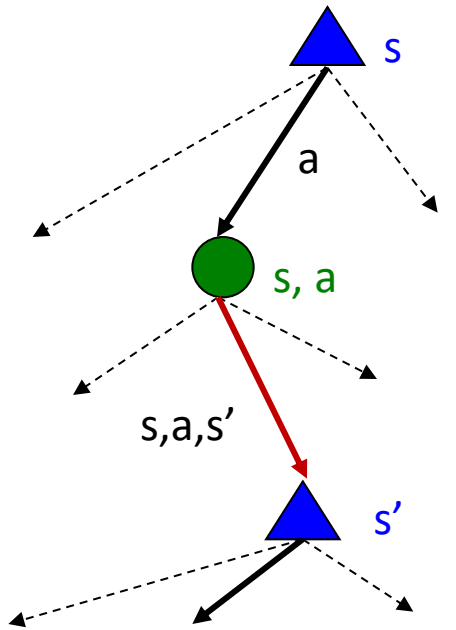
  $V^*(s)$ = expected utility starting in s and acting optimally

- 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)]

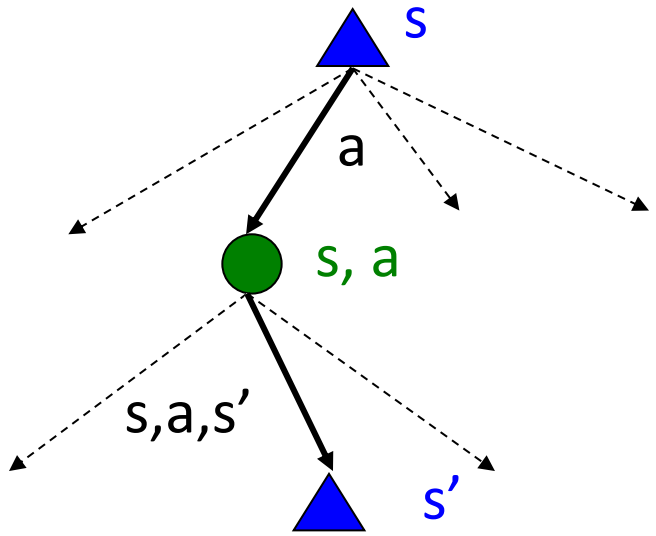
The Bellman Equations

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

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

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

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

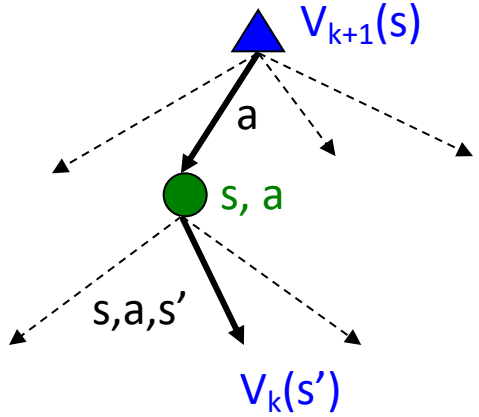


- These are the Bellman equations, and they characterize optimal values in a way we'll use over and over

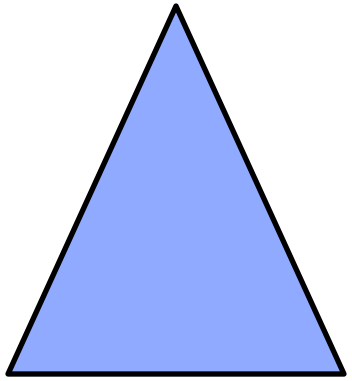
Value Iteration

- 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:

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

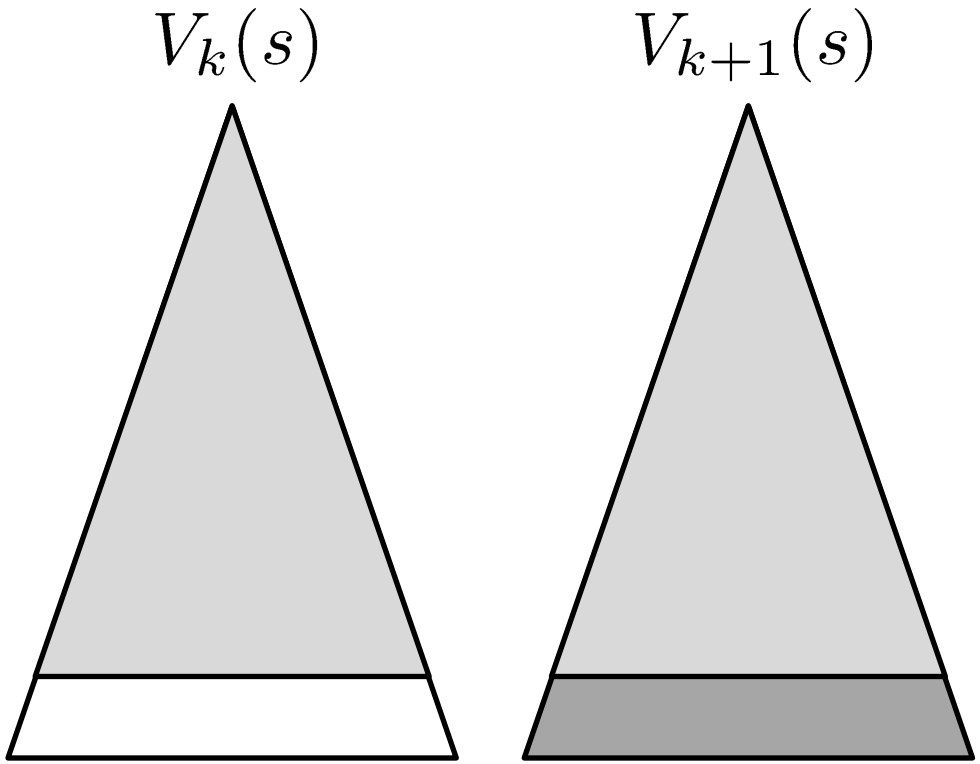


- Repeat until convergence
- Complexity of each iteration: $O(S^2A)$
- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do



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



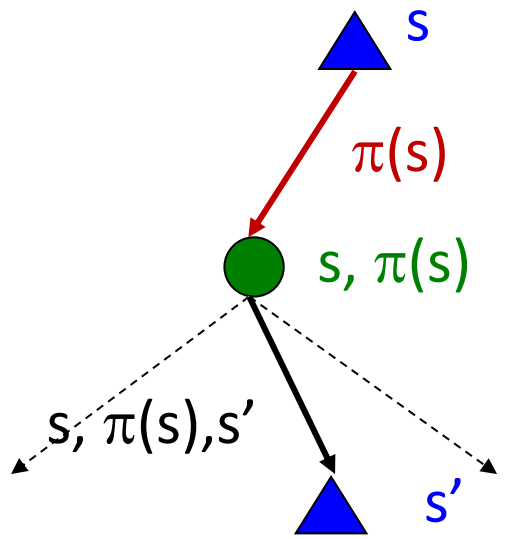
Policy Evaluation

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

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

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

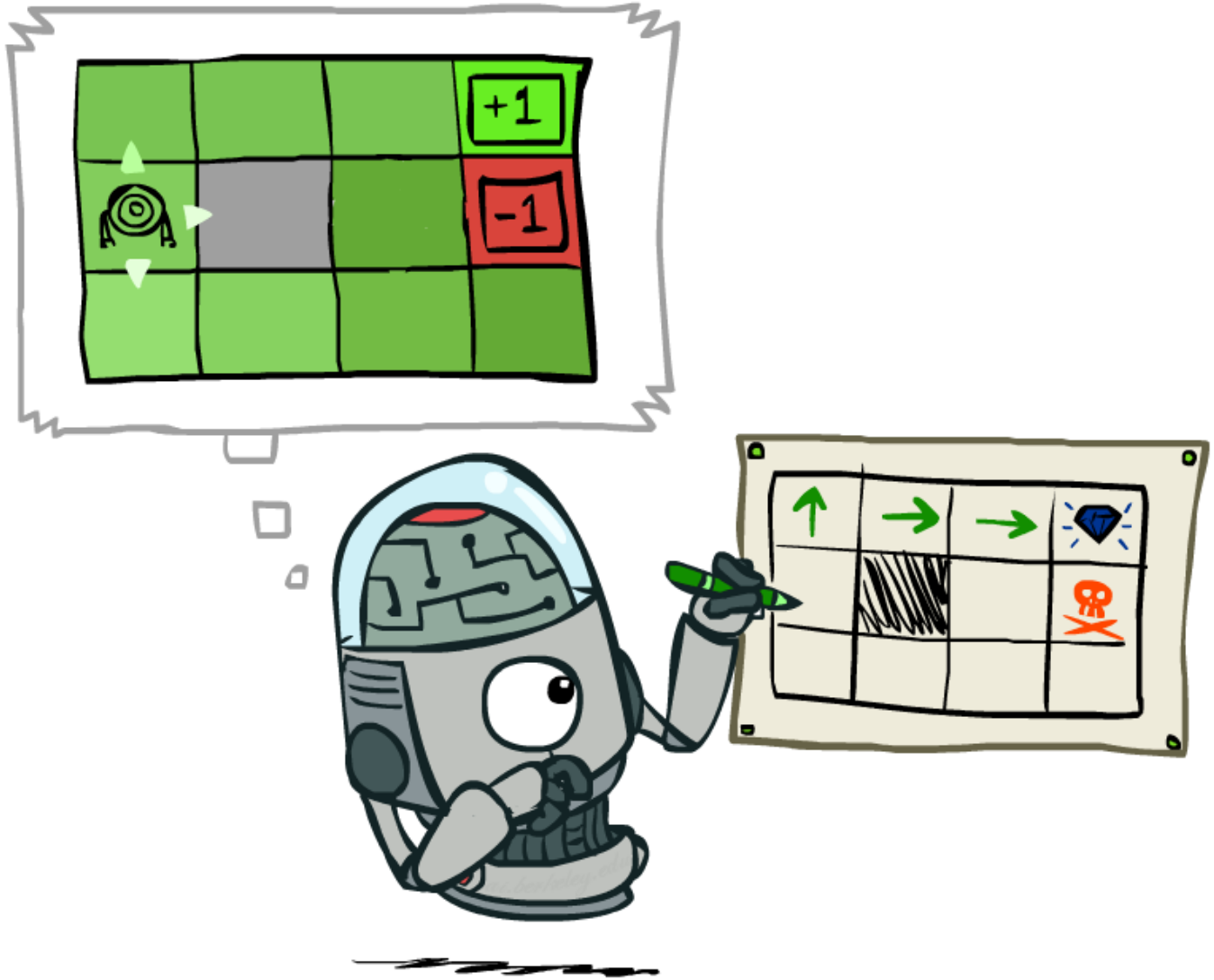
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V_k^\pi(s')]$$



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

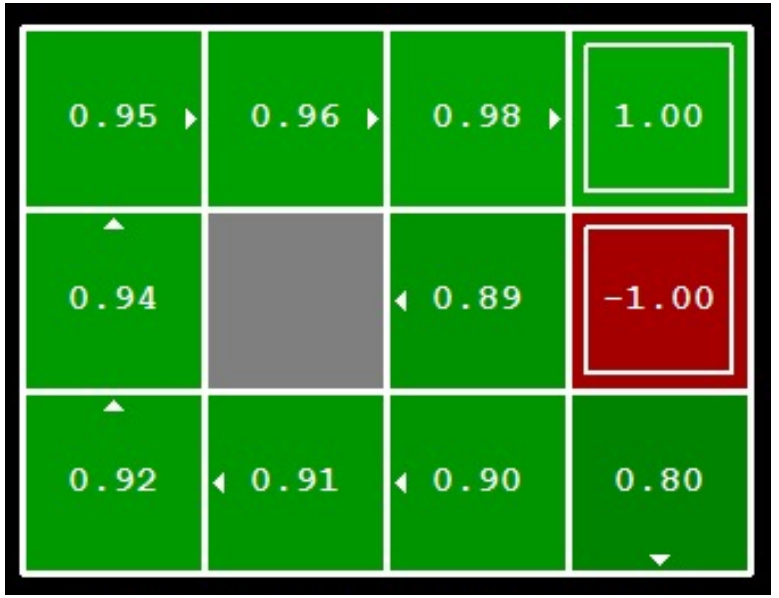
- Idea 2: Without the maxes, the Bellman equations are just a linear system
  - Solve with Matlab (or your favorite linear system solver)

Policy Extraction



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)

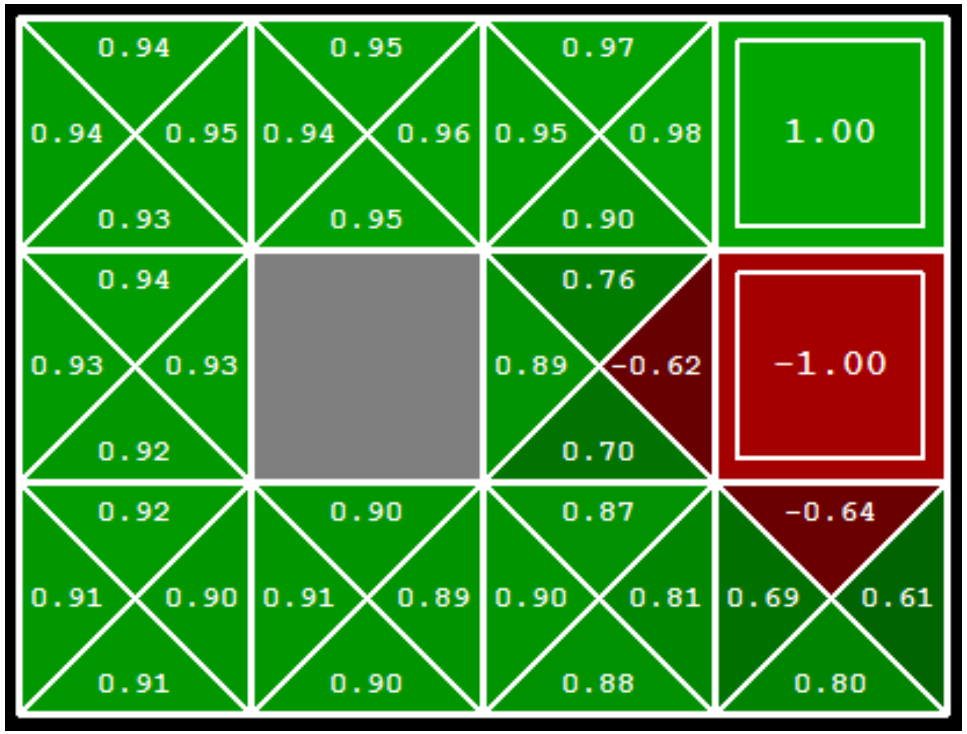


$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

- 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:

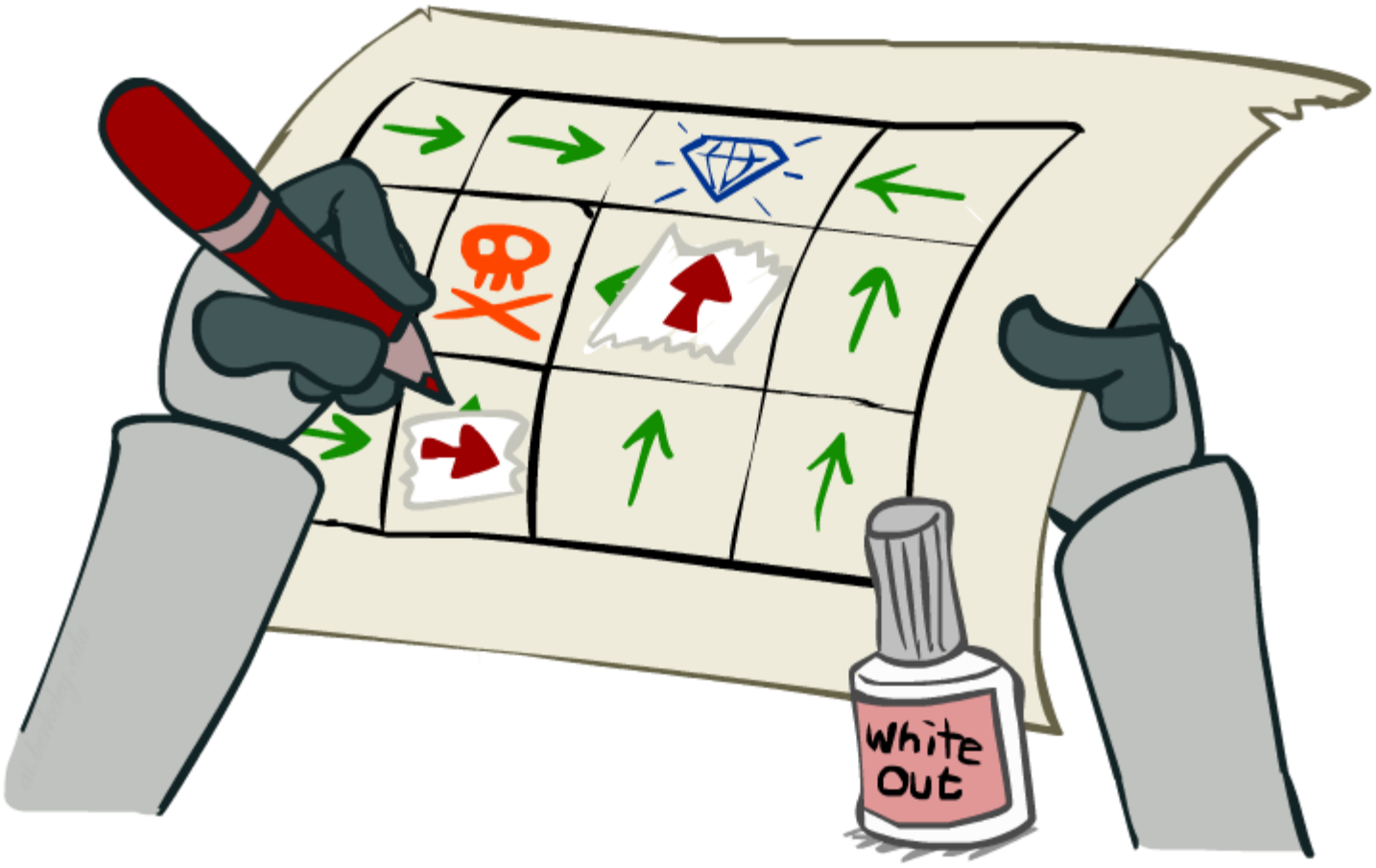


- How should we act?
  - Completely trivial to decide!

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

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

Policy Iteration



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?

南方科技大学

MAT8034: Machine Learning

Reinforcement Learning

Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Slide credits: ai.berkeley.edu

Reinforcement Learning

- Still assume a Markov decision process (MDP):
  - A set of states $s \in S$
  - A set of actions (per state) $A(s)$
  - A transition model $T(s,a,s')$
  - A reward function $R(s,a,s')$
- Still looking for a policy $\pi(s)$


Cool


Warm


Overheated

- New twist: don't know T or R
  - I.e. we don't know which states are good or what the actions do
  - Must explore new states and actions to discover how the world works

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

Pros and cons

- Pro:
  - Makes efficient use of experiences (low ***sample complexity***)
- Con:
  - May not scale to large state spaces
    - Solving MDP is intractable for very large $|S|$
  - RL feedback loop tends to magnify small model errors
  - Much harder when the environment is partially observable

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

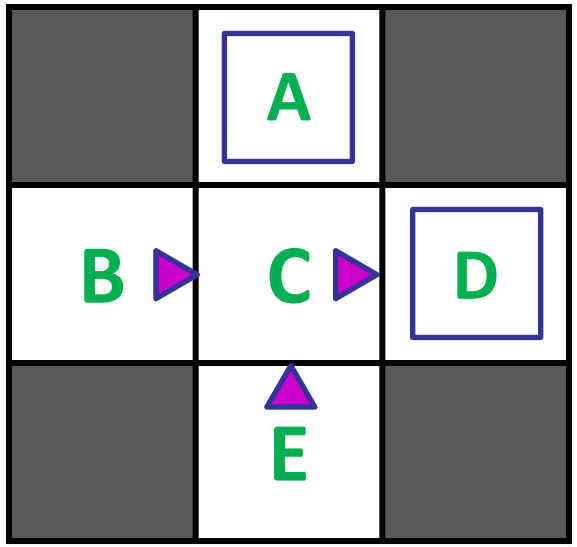
- Goal: Estimate $V^{\pi}(s)$, i.e., expected total discounted reward from s onwards

- 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)

Example: Direct Estimation

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

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

Episode 2

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

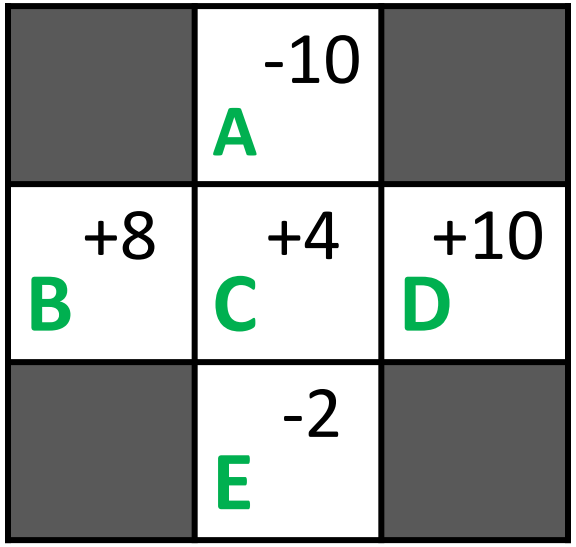
Episode 3

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

Episode 4

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

Output Values

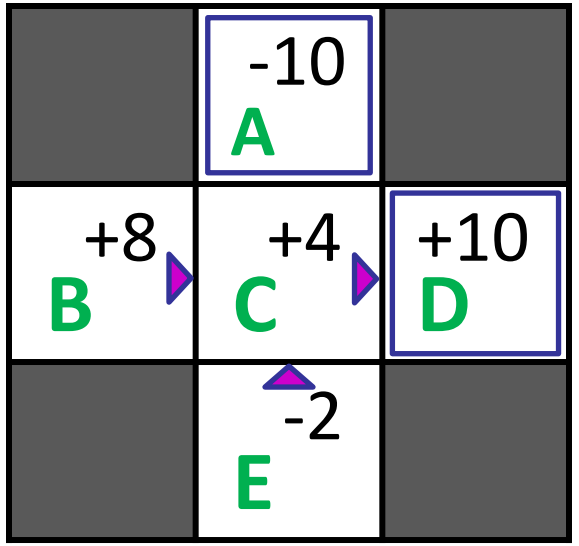


Problems with Direct Estimation

- 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

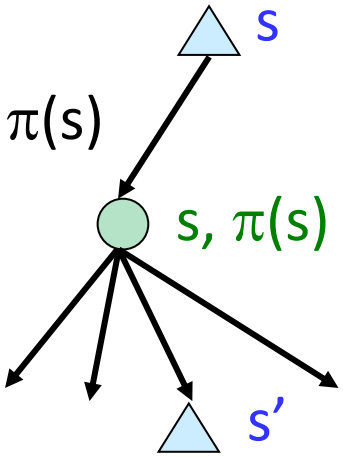
Output Values



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

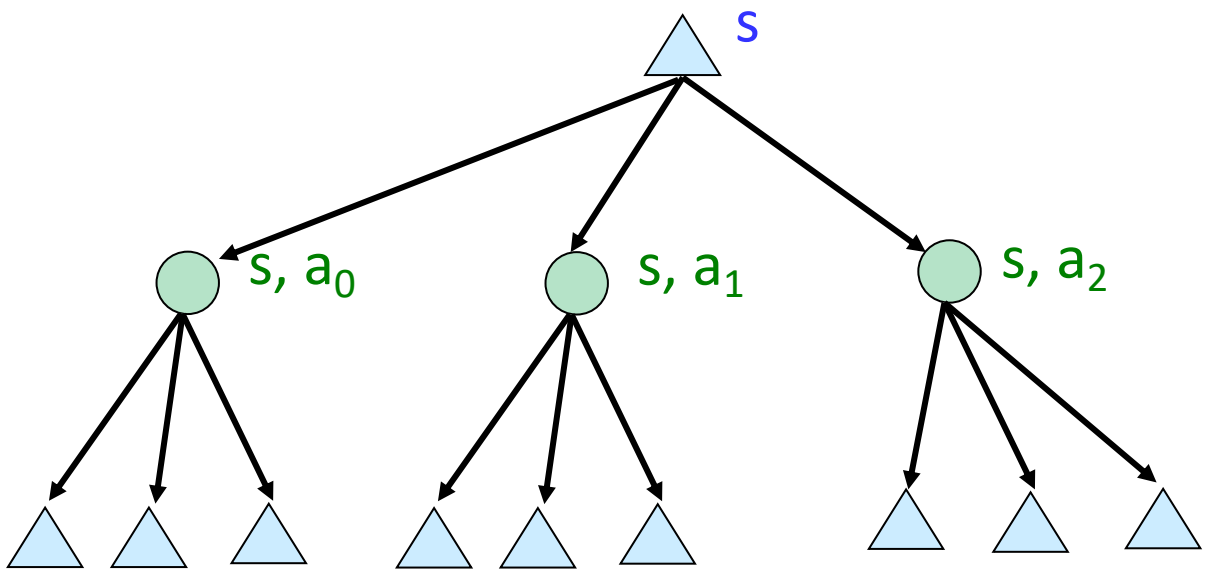
Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(sample - V^{\pi}(s))$

TD as approximate Bellman update

- Idea 3: Update values by maintaining a ***running average***
  - sample = $R(s,\pi(s),s') + \gamma V^{\pi}(s')$
  - $V^{\pi}(s) \leftarrow (1-\alpha) \cdot V^{\pi}(s) + \alpha \cdot \text{sample}$
  - $V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \cdot [\text{sample} - V^{\pi}(s)]$
  - This is the ***temporal difference learning rule***
  - $[\text{sample} - V^{\pi}(s)]$ is the "TD error"
  - α is the ***learning rate***
- Observe a sample, move $V^{\pi}(s)$ a little bit to make it more consistent with its neighbor $V^{\pi}(s')$

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)

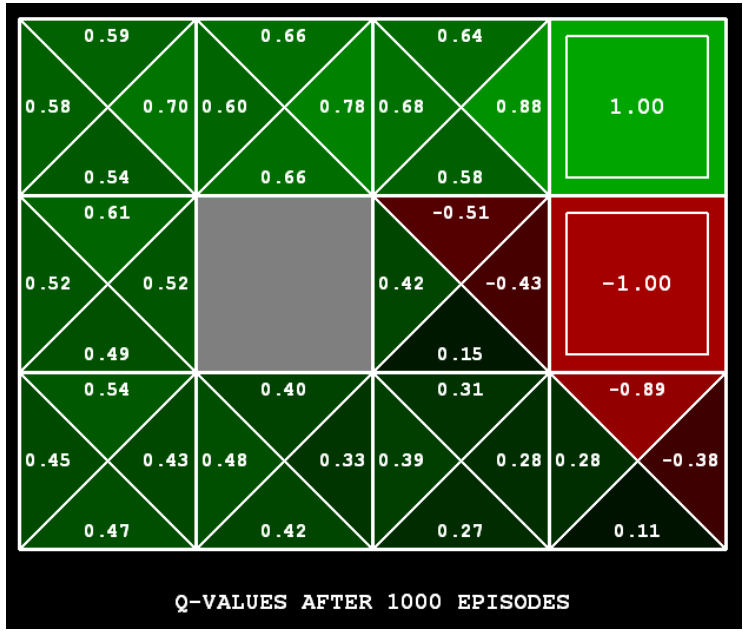
Q-Learning

- Learn Q(s,a) values as you go
  - Receive a sample (s,a,s',r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

    $sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$

  - Incorporate the new estimate into a running average:

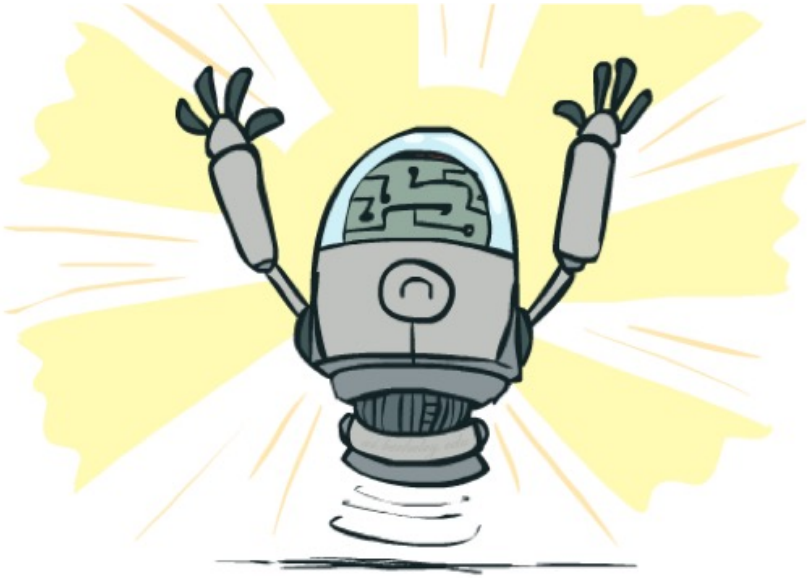
    $Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \cdot [sample]$



[Demo: Q-learning – gridworld (L10D2)]
[Demo: Q-learning – crawler (L10D3)]

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if samples are generated from a suboptimal policy!
- This is called off-policy learning
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough
  - … but not decrease it too quickly
  - Basically, in the limit, it doesn't matter how you select actions (!)



Policy Search

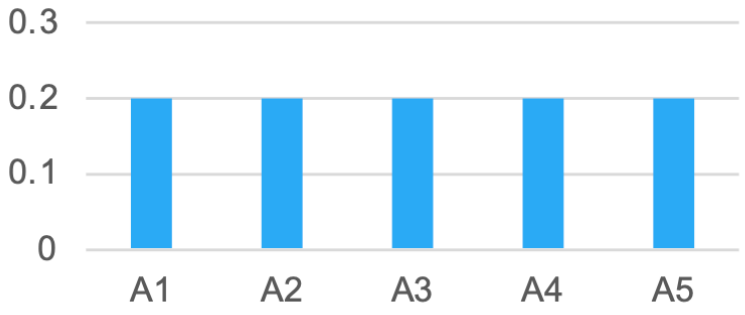
- Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate V / Q best
  - E.g. your value functions were probably horrible estimates of future rewards, but they still produced good decisions
  - Q-learning's priority: get Q-values close (modeling)
  - Action selection priority: get ordering of Q-values right (prediction)

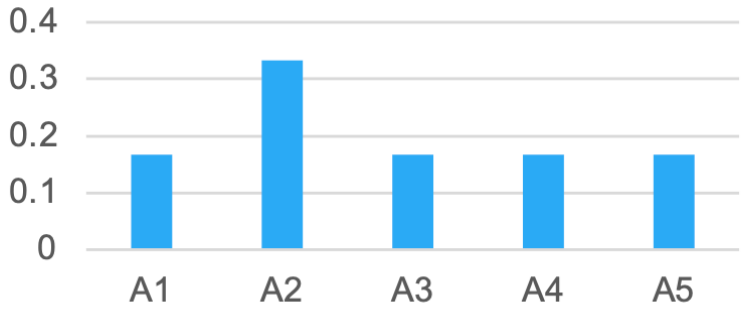
- Solution: learn policies that maximize rewards, not the values that predict them

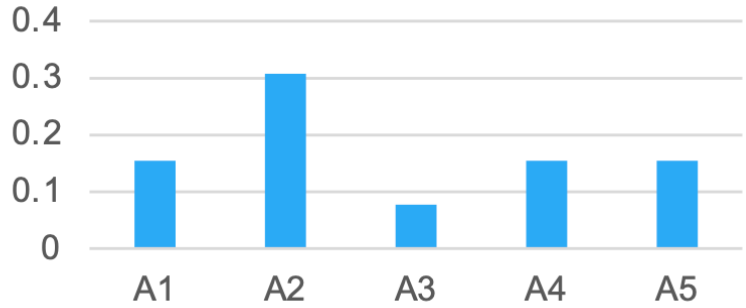
- Policy search: start with an ok solution (e.g. Q-learning) then fine-tune by **hill climbing** (or gradient ascent!) on feature weights

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

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

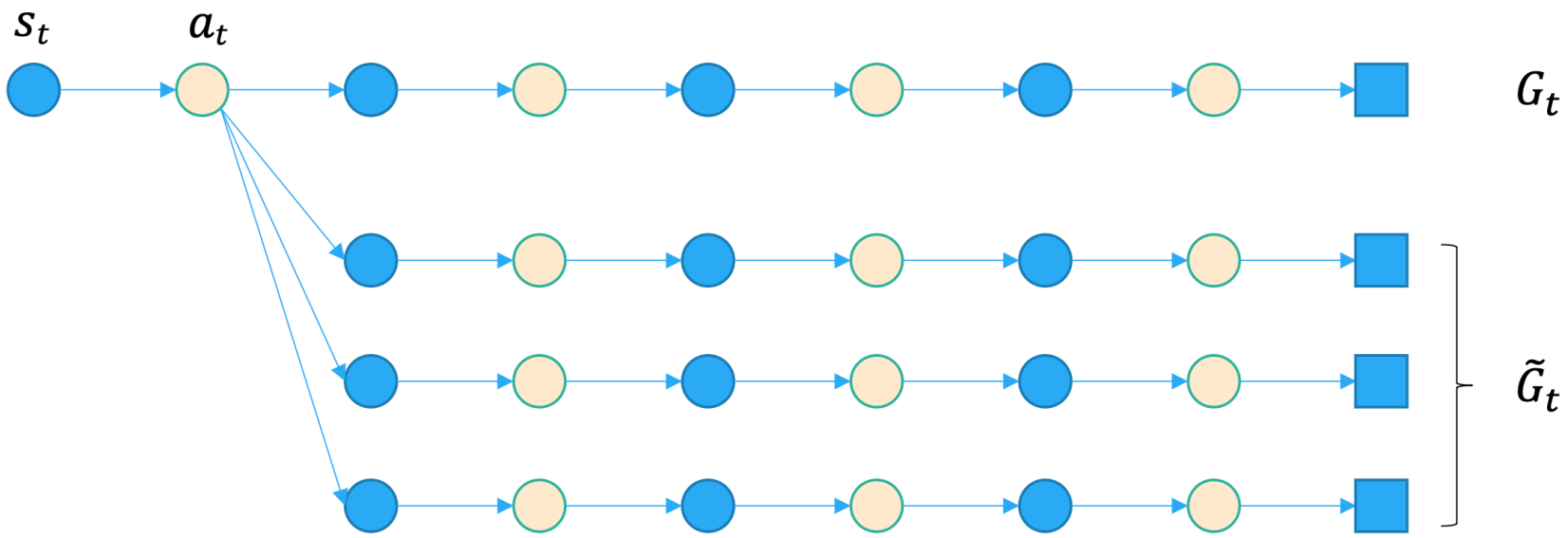
    end for

  end for

  return θ

REINFORCE Algorithm 2

- Can average multiple roll-out returns



$$\tilde{G}_t = \frac{1}{N}\sum_{i=1}^{n} G_t^{(i)}$$

Limitations of the REINFORCE Algorithm

- Episodic data requirement
  - REINFORCE typically requires tasks to terminate in order to compute the full return G_t
- Low data efficiency
  - In practice, REINFORCE needs a large amount of training data to achieve stable learning
- High variance in training (most critical issue)
  - The estimated returns from sampled trajectories can have very high variance, making gradient estimates noisy and unstable

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

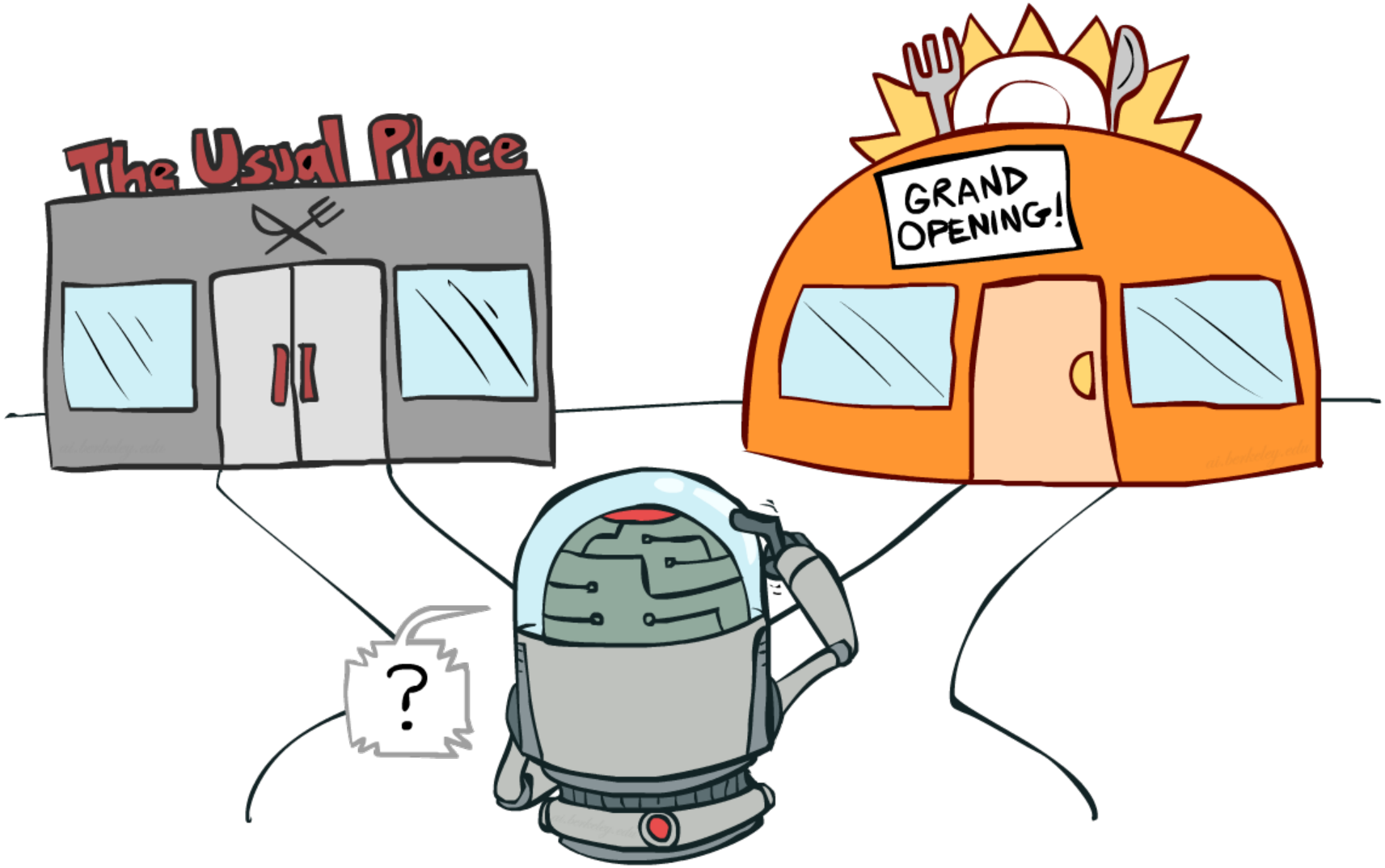
$$Q_\Phi(s, a) \simeq r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a), a' \sim \pi_\theta(a'|s')}[Q_\Phi(s', a')]$$

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

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

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

Exploration vs. Exploitation



Exploration vs. Exploitation

- ***Exploration***: try new things
- ***Exploitation***: do what's best given what you've learned so far
- Key point: pure exploitation often gets ***stuck in a rut*** and never finds an optimal policy!

Multi-armed bandits

- Multi-armed bandit is a tuple of $(\mathcal{A}, \mathcal{R})$
- $\mathcal{A}$: known set of m actions (arms)
- $\mathcal{R}^a(r) = \mathbb{P}[r \mid a]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- Goal: Maximize cumulative reward $\sum_{\tau=1}^{t} r_\tau$

Greedy algorithm

- We consider algorithms that estimate $\hat{Q}_t(a) \approx Q(a) = \mathbb{E}[R(a)]$
- Estimate the value of each action by Monte-Carlo evaluation

$$\hat{Q}_t(a) = \frac{1}{N_t(a)} \sum_{i=1}^{t} r_i \mathbb{1}(a_i = a)$$

- The **greedy** algorithm selects the action with highest value

$$a_t^* = \arg\max_{a \in \mathcal{A}} \hat{Q}_{t-1}(a)$$

Greedy algorithm: an example

1. Sample each arm once
   - Take action a^1 ($r \sim$Bernoulli(0.95)), get 0, $\hat{Q}(a^1) = 0$
   - Take action a^2 ($r \sim$Bernoulli(0.90)), get $+1$, $\hat{Q}(a^2) = 1$
   - Take action a^3 ($r \sim$Bernoulli(0.1)), get 0, $\hat{Q}(a^3) = 0$
2. Will the greedy algorithm ever find the best arm in this case?

Greedy can lock onto suboptimal action

Regret

- **Action-value** is the mean reward for action a

$$Q(a) = \mathbb{E}[r \mid a]$$

- **Optimal value** V^*

$$V^* = Q(a^*) = \max_{a \in \mathcal{A}} Q(a)$$

- **Regret** is the opportunity loss for one step, where the expectation is taken over the decision policy used to select a_t

$$l_t = \mathbb{E}[V^* - Q(a_t)]$$

Cumulative regret

- Over the horizon t, cumulative regret is the total opportunity loss

$$Reg_t = \mathbb{E}[\sum_{\tau=1}^{t} V^* - Q(a_\tau)]$$

- Maximize cumulative reward $\Leftrightarrow$ Minimize the cumulative regret

Regret decomposition

- **Count** $N_t(a)$ is number of times action a has been selected at time step t
- **Gap** Δ_a is the difference in value between action a and optimal action a^*, $\Delta_i = V^* - Q(a_i)$
- Regret is a function of gaps and counts

$$
\begin{aligned}
Reg_t &= \mathbb{E}\left[\sum_{\tau=1}^{t} V^* - Q(a_\tau)\right] \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}[N_t(a)](V^* - Q(a)) \\
&= \sum_{a \in \mathcal{A}} \mathbb{E}[N_t(a)]\Delta_a
\end{aligned}
$$

Objective: small counts for large (unknown) gap

Hoeffding inequality

Lemma. (Hoeffding inequality) Let $Z_1, \ldots, Z_n$ be n independent and identically distributed (iid) random variables drawn from a Bernoulli(ϕ) distribution. I.e., $P(Z_i = 1) = \phi$, and $P(Z_i = 0) = 1 - \phi$. Let $\hat{\phi} = (1/n) \sum_{i=1}^{n} Z_i$ be the mean of these random variables, and let any $\gamma > 0$ be fixed. Then

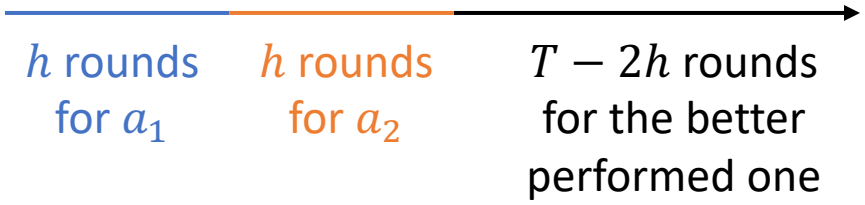
$$P(|\phi - \hat{\phi}| > \gamma) \leq 2 \exp(-2\gamma^2 n)$$

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

- There are $K = 2$ actions/arms
- Suppose
  - $Q(a_1) > Q(a_2)$
  - $\Delta = Q(a_1) - Q(a_2)$

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:

$$Reg(T) = T \cdot Q(a_1) - \mathbb{E}\left[\sum_{t=1}^{T} Q(a_t)\right]$$

Sample mean

$$= h\Delta + (T-2h) \cdot \Delta \cdot \mathbb{P}\left(\hat{Q}(a_1) < \hat{Q}(a_2)\right)$$

$$= h\Delta + (T-2h) \cdot \Delta \cdot \mathbb{P}\left(\left(\hat{Q}(a_2) - Q(a_2)\right) - \left(\hat{Q}(a_1) - Q(a_1)\right) > \Delta\right)$$

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

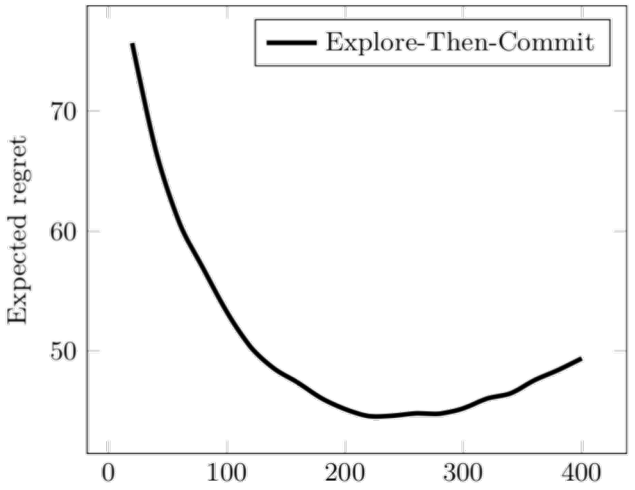
Hoeffding's inequality

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

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

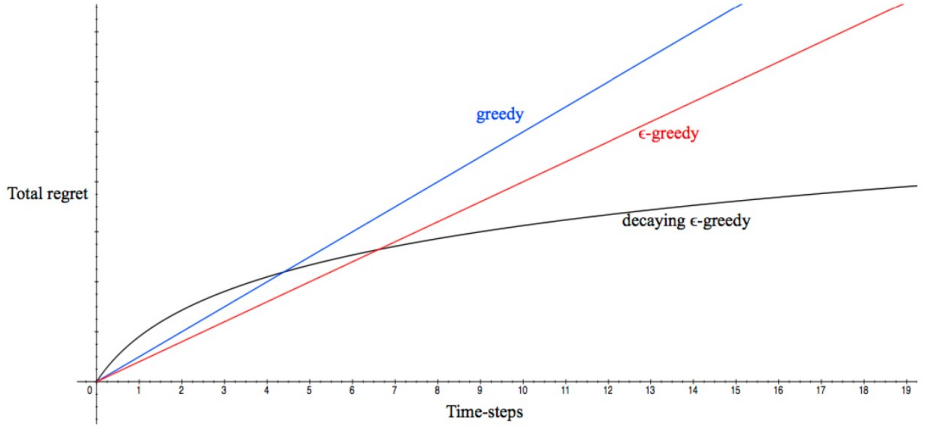
require the knowledge of Δ

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



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

A soft version: ε-greedy



- For each round t
  - $\varepsilon_t \in (0,1)$
  - With probability ε_t, exploration (uniformly random select arms)
  - With probability $1 - \varepsilon_t$, exploitation (select the best performed arm so far)

- When $\varepsilon_t = \min\left\{1, \frac{c}{t\Delta^2}\right\}$, $Reg(T) = O\left(\frac{\log T}{\Delta}\right)$

Demo: https://cse442-17f.github.io/LinUCB/ Figure: CS234

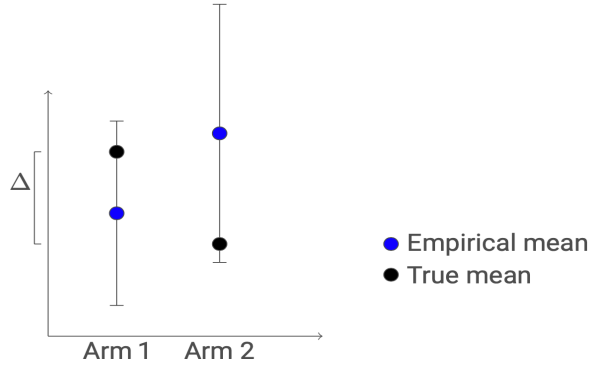
Upper confidence bound (UCB) [Auer et al., 2002]

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

$$Q(a_j) \in \left[\hat{Q}(a_j) - \sqrt{\frac{\log 1/\delta}{N_j}}, \hat{Q}(a_j) + \sqrt{\frac{\log 1/\delta}{N_j}}\right]$$

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

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

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

$$Q(a_1) \leq \mathrm{UCB}_1 \leq \mathrm{UCB}_j \leq Q(a_j) + 2\sqrt{\frac{\log 1/\delta}{N_j(t)}}$$

  - $\Rightarrow 2\sqrt{\frac{\log 1/\delta}{N_j(t)}} \geq \Delta_j := Q(a_1) - Q(a_j)$
  - $\Rightarrow N_j(t) \leq O\left(\frac{\log 1/\delta}{\Delta_j^2}\right)$

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

- Assume arm a_1 is the best arm
- If sub-optimal arm a_j is selected
  - w/ high probability
    $Q(a_1) \leq \mathrm{UCB}_1 \leq \mathrm{UCB}_j \leq Q(a_j) + 2\sqrt{\frac{\log 1/\delta}{N_j(t)}}$
  - $\Rightarrow 2\sqrt{\frac{\log 1/\delta}{N_j(t)}} \geq \Delta_j := Q(a_1) - Q(a_j)$
  - $\Rightarrow N_j(t) \leq O\left(\frac{\log 1/\delta}{\Delta_j^2}\right)$
- By choosing $\delta = 1/T$, cumulative regret:
  $O\left(\sum_{j\neq 1} \frac{\log T}{\Delta_j^2} \cdot \Delta_j\right) = O(K \log T/\Delta)$

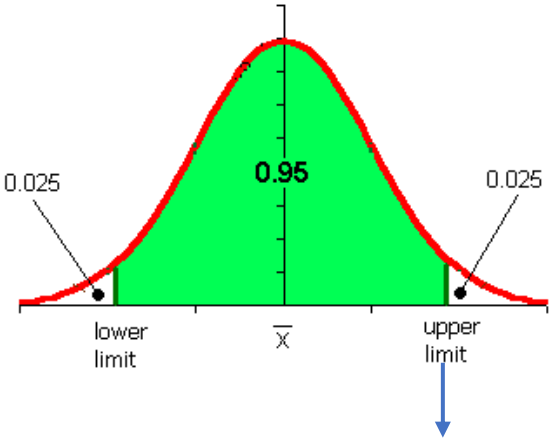
Thompson sampling (TS) [Agrawal and Goyal, 2013]

- Assume each arm has prior $\text{Gaussian}(0,1)$
- Sample an estimate $\tilde{Q}_j$ from the posterior distribution

$$\tilde{Q}_j \sim \text{Gaussian}\left(\hat{Q}_j, \frac{1}{1+N_j(t)}\right)$$

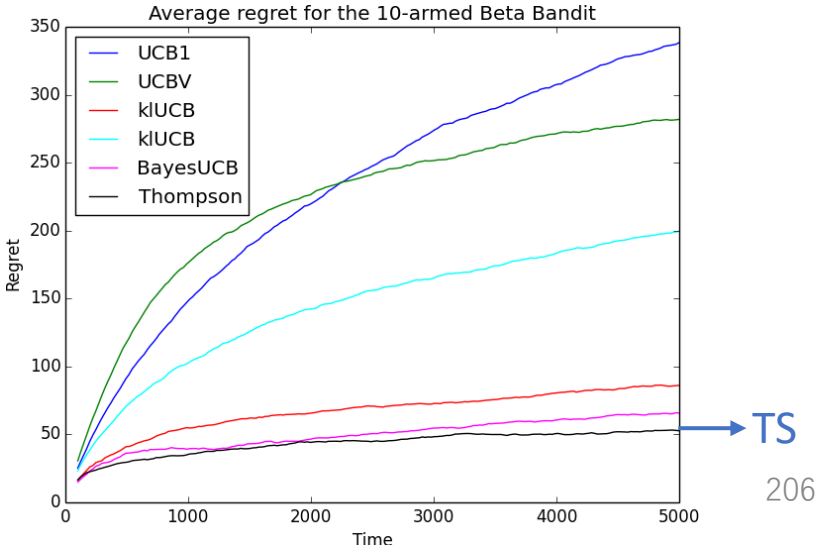
Exploitation

Exploration



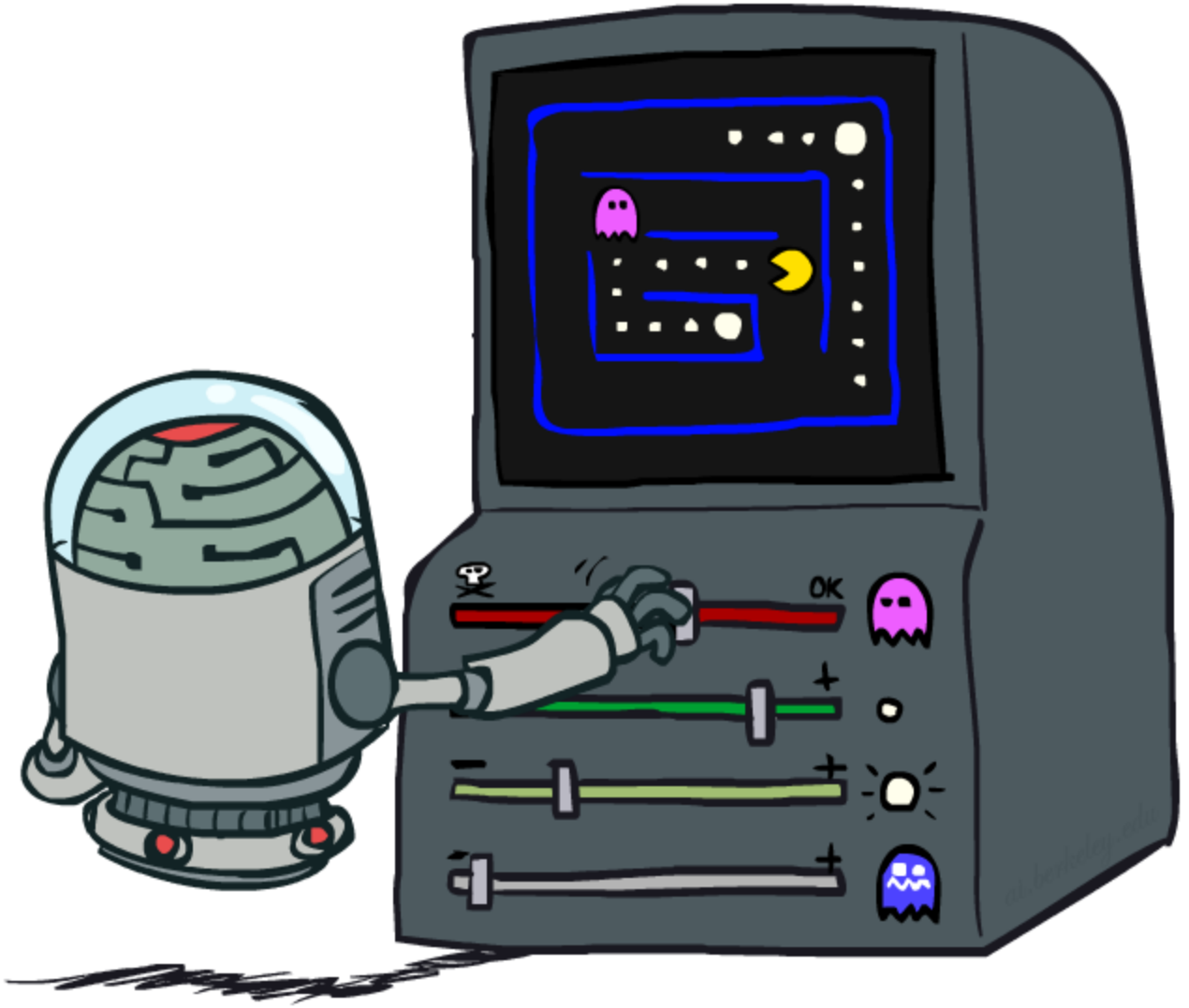
- Select the arm $A(t) \in \text{argmax}_{j \in [K]} \tilde{Q}_j$

- Also have $O(K \log T/\Delta)$ regret
- Usually outperforms UCB



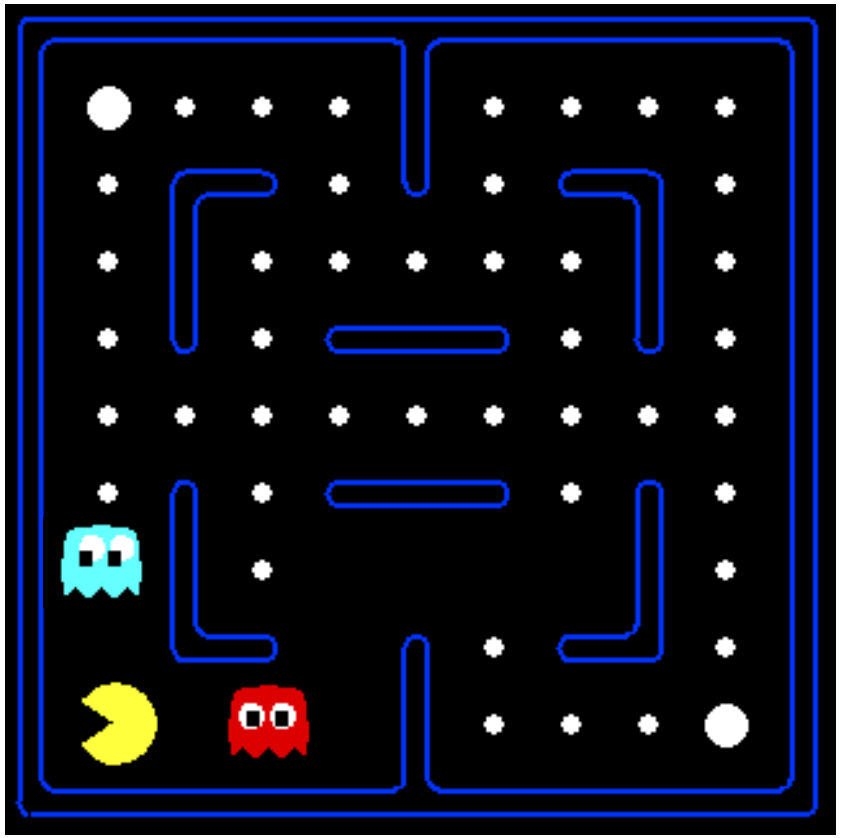
https://learnforeverlearn.com/bandits/

Approximate Q-Learning



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 and 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

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

- 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

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

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

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

Intuitive interpretation

- Original Q-learning rule tries to reduce prediction error at s,a:
  - $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
- Instead, we update the weights to try to reduce the error at s,a:
  - $\theta_i \leftarrow \theta_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] \partial Q_\theta(s,a)/\partial \theta_i$

    $= \theta_i + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)] f_i(s,a)$
- Intuitive interpretation:
  - Adjust weights of active features
  - If something bad happens, blame the features we saw; decrease value of states with those features. If something good happens, increase value!

南方科技大学

MAT8034: Machine Learning

Deep Reinforcement Learning

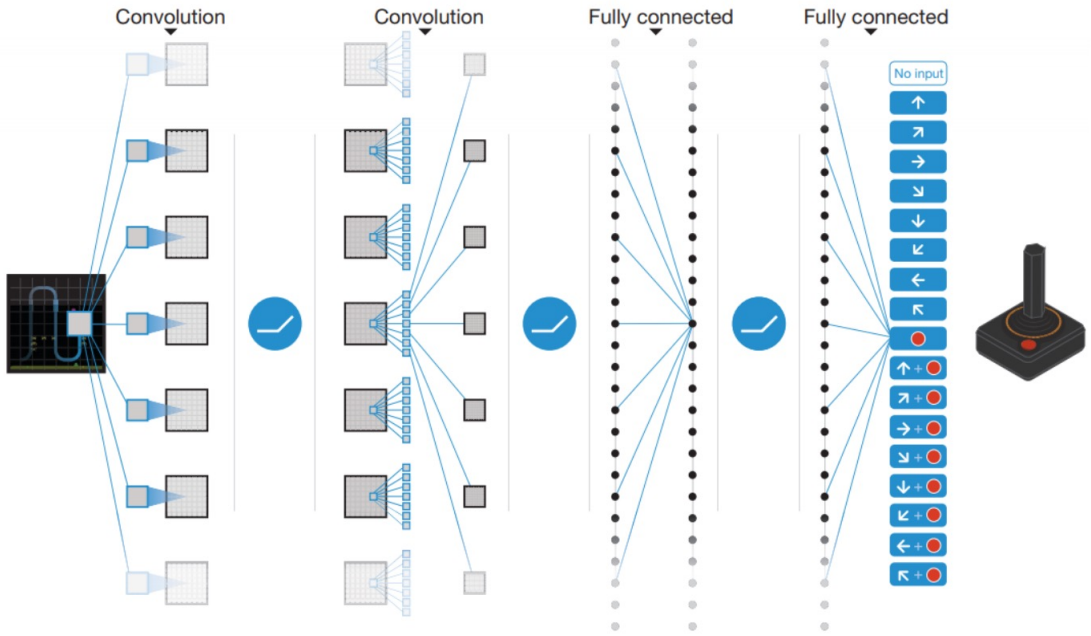
Fang Kong

https://fangkongx.github.io/Teaching/MAT8034/Spring2026/index.html

Part of slide credits: Weinan Zhang

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: Loss function

- Intuition: Use a deep neural network to approximate Q(s,a)
- Loss Function?

$$\mathbb{E}_{s_t, a_t, s_{t+1}, r_t} \left[\frac{1}{2} \left(r_t + \gamma \max_{a'} Q_\theta (s_{t+1}, a') - Q_\theta (s_t, a_t) \right)^2 \right]$$

DQN (cont.)

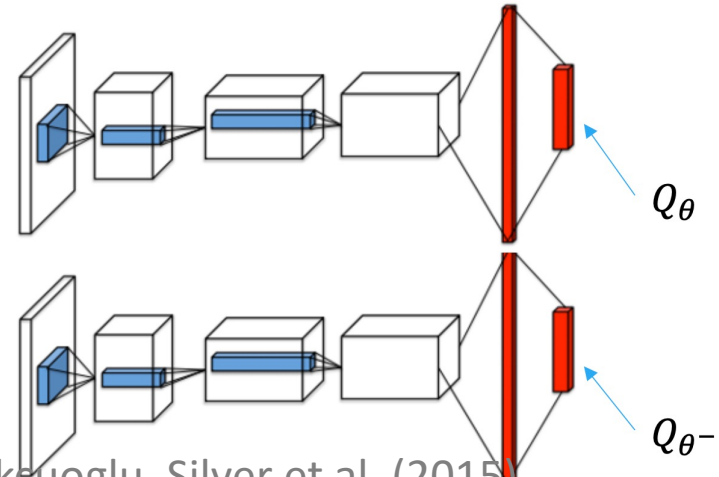
- Instability arises in the learning process
  - Samples $\{(s_t, a_t, s_{t+1}, r_t)\}$ are collected sequentially and do not satisfy the i.i.d. assumption
  - Frequent updates of Q(s,a) cause instability
- Solutions: Experience replay
  - Store transitions $e_t = (s_t, a_t, s_{t+1}, r_t)$ in a replay 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

- Assume the action-value function follows a distribution:

$$Q(s,a) \sim \mathcal{N}(\mu, \sigma)$$

- Then: $V(s) = \mathbb{E}[Q(s,a)] = \mu$ $\quad$ $Q(s,a) = \mu + \varepsilon(s,a)$

- How do we describe $\varepsilon(s,a)$?

$$\varepsilon(s,a) = Q(s,a) - V(s)$$

  - This term is also known as the Advantage function

"Dueling Network Architectures for Deep Reinforcement Learning", Wang et al. (2016)

Dueling DQN (cont.)

- 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

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

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

Deep RL – Policy-based methods

Review: The policy gradient theorem

- The policy gradient theorem generalizes the derivation of likelihood ratios to the multi-step MDP setting.
- It replaces the immediate reward r_t with the expected long-term return $Q^{\pi}(s, a)$.

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

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

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

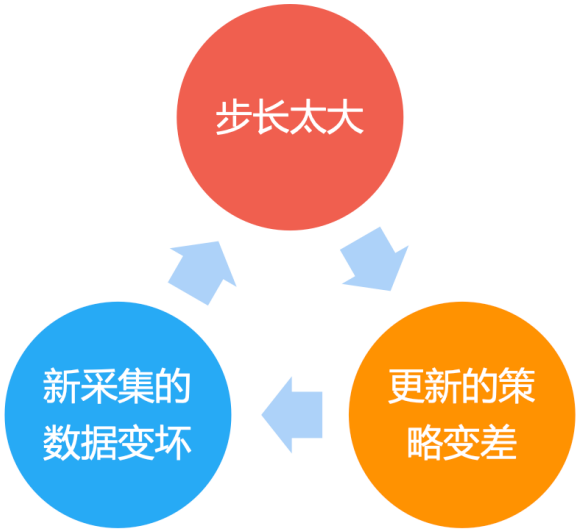
- Gradient of the policy network

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

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

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

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

Important:
Performance difference lemma

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

Sampling inconvenience

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

How to understand this lemma?

- Recall the policy improvement step in the PI algorithm

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

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

  - We proved that
  - $Q^{\pi_i}\left(s, \pi_{i+1}(s)\right) \geq Q^{\pi_i}\left(s, \pi_i(s)\right), \forall s \implies V^{\pi_{i+1}}(s) \geq V^{\pi_i}(s), \forall s$

How to deal with sample inconvenience? 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

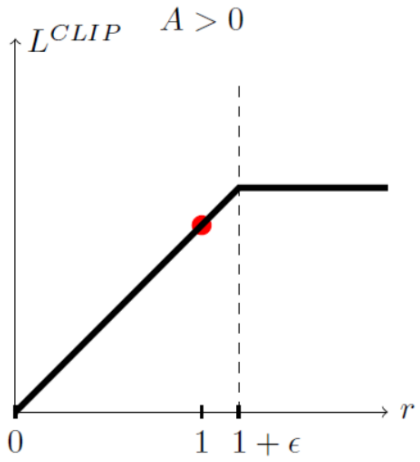
Proximal Policy Optimization (PPO)

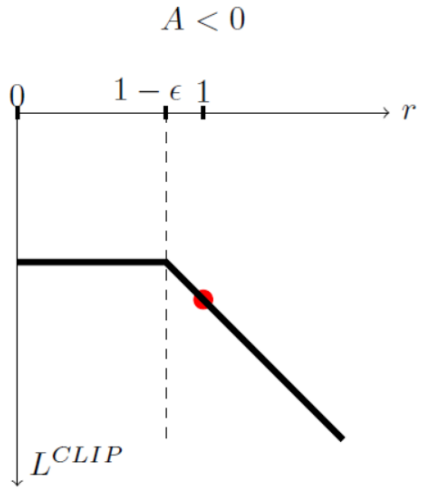
- Clipped Surrogate Objective

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: Adaptive penalty version

- Another version: adaptive penalty version

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

  - 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

RLHF and DPO

Bradley-Terry Model (1952)

- First consider simpler setting of k-armed bandits: K actions $b_1, b_2, \ldots b_k$. No state transition
- Assume a human makes noisy pairwise comparisons, where the probability she prefers $b_i \succ b_j$ is

$$P(b_i \succ b_j) = \frac{\exp(r(b_i))}{\exp(r(b_i)) + \exp(r(b_j))} = p_{ij}$$

$$= \sigma\left(r(b_i) - r(b_j)\right)$$

- Transitive: p_{ik} is determined from p_{ij} and p_{jk}

Which one is the best?

- **Condorcet Winner**

  An item b_i is a **Condorcet winner** if

  $$P(b_i \succ b_j) \geq 0.5, \quad \forall\, j \in [K],\ j \neq i.$$

Copeland Winner

A **Copeland winner** is defined as

$$\arg\max_{i \in [K]} \sum_{j \neq i} \mathbf{1}\{P(b_i \succ b_j) > 0.5\}.$$

Borda Winner

A **Borda winner** is defined as

$$\arg\max_{i \in [K]} \sum_{j \neq i} P(b_i \succ b_j).$$

Under the BT model, three winners are consistent

Train the Bradley-Terry Model: The bandit setting

- Assume have N tuples of form (b_i, b_j, μ) where $\mu = 1$ if the human marked $b_i \succ b_j$, $\mu = 0.5$ if the human marked $b_i = b_j$, else 0 if $b_i \prec b_j$
- Maximize likelihood with cross entropy

$$loss = -\sum_{(b_i, b_j, \mu) \in \mathcal{D}} \mu \quad \log P(b_i \succ b_j) + (1 - \mu \quad) \log P(b_j \succ b_i)$$

Recall logistic regression

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,

$$L(\theta) = P(y \mid X; \theta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^{n} h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1 - y^{(i)}}$$ exponents encode "if-then"

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

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

Train the Bradley-Terry Model: The RL setting

- Can also do this for trajectories

  Consider two trajectories, $\tau^1(s_0, a_7, s_{14}, \ldots)$ and $\tau^2(s_0, a_6, s_{12}, \ldots)$

  Let $R^1 = \sum_{i=0}^{T-1} r_i^1$ be the (latent, unobserved) sum of rewards for trajectory τ^1 and similarly for R^2.

  Define the probability that a human prefers $\tau^1 \succ \tau^2$ as

$$\hat{P}\left[\tau^1 \succ \tau^2\right] = \frac{\exp \sum_{i=0}^{t-1} r_i^1}{\exp \sum_{i=0}^{t-1} r_i^1 + \exp \sum_{i=0}^{t-1} r_i^2},$$

- Use learned reward model, and do PPO with this model

RLHF: putting it all together

- Finally, we have:
  - A pretrained (possibly instruction-finetuned) reference model π_{ref}
  - A reward model r_ϕ that produces scalar rewards for LM outputs
  - A method for optimizing LM parameters towards reward functions
- Now to do RLHF:
  - Initialize a copy of the model π_θ with parameters θ to optimize
  - Optimize the following reward with RL

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y|x)||\pi_{\mathrm{ref}}(y|x)\right]$$

RLHF: preference->reward->policy

- RLHF
  - First train the reward model by minimizing negative log likelihood:

$$J_{RM}(\phi) = -\mathbb{E}_{(s^w, s^l)\sim D}\left[\log \sigma(RM_\phi(s^w) - RM_\phi(s^l))\right]$$

  - Then learn a policy that maximizes the reward (with small distance to the pretrained model)

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y|x)||\pi_{\mathrm{ref}}(y|x)\right]$$

- Can we remove the reward step?

Direct Preference Optimization

RLHF Objective

(get **high reward**, stay **close** to reference model)

any reward function

$$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi(y|x)} \left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}(\pi(\cdot \mid x)\|\pi_{\mathrm{ref}}(\cdot \mid x))$$

Closed-form Optimal Policy

(write **optimal policy** as function of **reward function**; from prior work)

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

with $Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)$

Note **intractable sum** over possible responses; can't immediately use this

Rearrange

(write **any reward function** as function of **optimal policy**)

Ratio is **positive** if policy likes response more than reference model, **negative** if policy likes response less than ref. model

$$r(x,y) = \underbrace{\beta\log\frac{\pi^*(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta\log Z(x)}_{\text{some parameterization of a reward function}}$$

DPO: Putting it together

A loss function on reward functions

+

A transformation between reward functions and policies

=

A loss function on policies

Derived from the Bradley-Terry model of human preferences:

$$\mathcal{L}_R(r, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma(r(x, y_w) - r(x, y_l))\right]$$

$$r_{\pi_\theta}(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

When substituting, the **log Z term cancels**, because the loss only cares about **difference** in rewards

Reward of **preferred** response

Reward of **dispreferred** response

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

GRPO in deepseek

Group Relative Policy Optimization In order to save the training costs of RL, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), which foregoes the critic model that is typically the same size as the policy model, and estimates the baseline from group scores instead. Specifically, for each question q, GRPO samples a group of outputs $\{o_1, o_2, \cdots, o_G\}$ from the old policy $\pi_{\theta_{old}}$ and then optimizes the policy model π_θ by maximizing the following objective:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right), \tag{1}$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \tag{2}$$

where ε and β are hyper-parameters, and A_i is the advantage, computed using a group of rewards $\{r_1, r_2, \ldots, r_G\}$ corresponding to the outputs within each group:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \cdots, r_G\})}{\text{std}(\{r_1, r_2, \cdots, r_G\})}. \tag{3}$$