Inference and Learning

Wenxiao Yang*

*Department of Mathematics, University of Illinois at Urbana-Champaign

2022

Contents

1 Statistical Inference

1.1 Basics

Given an observation $x \in X$, we want to estimate an unknown state $\theta \in S$ (not necessarily random). The θ can form x with $P_\theta(x)$. We use decision rule $\delta(x)$ to form an action (estimation of θ) $a = \hat{\theta}$.

Example:

(1) Binary hypothesis testing (detection) when $S = \{0, 1\}$ e.g. $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$

(2) Multiple hypothesis testing (classification) when $S = \{1, 2, ..., n\}$

(3) (Estimation) when $S = \mathbb{R}$ e.g. $P_\theta \in N(\theta, \sigma^2)$

1.2 Decision Rule Examples

Binary HT Example

For the example Binary HT, $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$: decision rule $\delta : \mathbb{R} \to \{0, 1\}$

We can find a τ such that $\delta(x) = \begin{cases} 1, & x \geq \tau \\ 0, & else \end{cases} = \mathbf{1}_{x \geq \tau}$. Howe to choose τ?

Type-I error probability: probability that θ is 0 but receive $\delta(x) = 1$.

$$P_I = P_0\{\delta(x) = 1\} = P_0\{x \geq \tau\} = Q\left(\frac{\tau}{\sigma}\right)$$

Type-II error probability: probability that θ is 1 but receive $\delta(x) = 0$.

$$P_{II} = P_1\{\delta(x) = 0\} = P_1(x < \tau) = Q(\frac{\mu - \tau}{\sigma})$$

Both P_I and P_{II} depends on τ. $Q(t) = \int_t^\infty \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} dx$

For $\tau = \frac{\mu}{2}$, $P_I = P_{II} = Q\left(\frac{\mu}{2\sigma}\right)$

Multiple HT Example

Consider three state $S = \{1, 2, 3\}$. We can find a τ such that $\delta(x) = \begin{cases} 1, & x < \tau_1 \\ 2, & \tau_1 \leq x \leq \tau_2 \\ 3, & x > \tau_2 \end{cases} = \mathbf{1}_{x \geq \tau}$.

Conditional Error Probabilities: probability that θ is i but receive $\delta(x) = j$ (6 types in this example)

$$P_i\{\delta(x) = j\}, \forall i \neq j$$

Estimation Example

Ex: $P_\theta \sim N(\theta, \sigma^2)$. Perform $\delta(x) = \hat{\theta}$ by using mean-squared error (MSE):

$$MSE = \mathbb{E}_\theta \left[(\delta(x) - \theta)^2 \right], \theta \in \mathbb{R}$$

1.3 Maximum-Likelihood Principle (state is norandom)

Maximum-Likelihood Principle

$$\hat{\theta} = \underset{\theta \in S}{\operatorname{argmax}} P_\theta(x) = \underset{\theta \in S}{\operatorname{argmax}} \ln P_\theta(x)$$

Applied to the binary example: $P_0 \sim N(0, \sigma^2), P_1 \sim N(\mu, \sigma^2)$.
$P_0(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}, P_1(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$. $\ln P_0(x) = c - \frac{x^2}{2\sigma^2}$, $\ln P_1(x) = c - \frac{(x-\mu)^2}{2\sigma^2}$.
Then, the rule can become

$$\hat{\theta} = \begin{cases} 0, & x^2 < (x-\mu)^2 \\ 1, & else \end{cases} = \mathbf{1}_{x^2 \geq (x-\mu)^2} = \mathbf{1}_{x \geq \frac{\mu}{2}}$$

Vector Observations

Observations $X = (x_1, x_2, ..., x_n)$, where i.i.d. $x_i \sim P_\theta$. Then

$$P_\theta(X) = \prod_{i=1}^{n} P_\theta(x_i), \ \ln P_\theta(X) = \sum_{i=1}^{n} \ln P_\theta(x_i)$$

$\ln P_0(x) = cn - \frac{\sum_{i=1}^n x_i^2}{2\sigma^2}$, $\ln P_1(x) = cn - \frac{\sum_{i=1}^n (x_i-\mu)^2}{2\sigma^2}$.

Then, the rule can become

$$\hat{\theta} = \begin{cases} 0, & \sum_{i=1}^n x_i^2 < \sum_{i=1}^n (x_i - \mu)^2 \\ 1, & else \end{cases} = \mathbf{1}_{\sum_{i=1}^n x_i^2 \geq \sum_{i=1}^n (x_i-\mu)^2} = \mathbf{1}_{\bar{x} \geq \frac{\mu}{2}}$$

where $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$. Under both H_0 and H_1, $\bar{x} \sim N(0, \frac{\sigma^2}{n})$.

Then, type I error prob and type II error prob are the same

$$P_I = P_0\{\bar{x} \geq \frac{\mu}{2}\} = P_{II} = P_1\{\bar{x} < \frac{\mu}{2}\} = Q\left(\frac{\mu\sqrt{n}}{2\sigma}\right)$$

Esitimation $S = \mathbb{R}$

To estimate θ when $S = \mathbb{R}$

$$\begin{aligned}
&\max_{\theta\in\mathbb{R}} \sum_{i=1}^{n} \ln P_\theta(x_i) \\
&\Leftrightarrow \max_{\theta\in\mathbb{R}} \left[cn - \frac{\sum_{i=1}^{n}(x_i-\theta)^2}{2\sigma^2} \right] \\
&\Leftrightarrow \max_{\theta\in\mathbb{R}} \sum_{i=1}^{n}(x_i-\theta)^2 \Rightarrow \hat{\theta} = \bar{x}
\end{aligned}$$

Then, with $\bar{x} \sim N(\theta, \frac{\sigma^2}{n})$, the

$$MSE_\theta = \mathbb{E}_\theta \left(\bar{x} - \theta\right)^2 = \frac{\sigma^2}{n}$$

1.4 Bayesian Decision Rule (state is random)

1.4.1 Rules

Prior probability distribution $\pi(\theta)$,

Loss/cost function with action (estimation) a is $l(a, \theta)$. e.g.

1. (binary HT) Hamming/zero-one loss $l(a = \hat{\theta}, \theta) = \mathbf{1}_{a\neq\theta}$
2. (estimation) Squared error loss $l(a = \hat{\theta}, \theta) = (a-\theta)^2$; Absolute error loss $l(a, \theta) = |a - \theta|$.

Risk of decision rule δ:

$$R(\delta) = \mathbb{E}\left(l(\delta(X), \theta)\right)$$

where (X, θ) are random with prob $\pi(\theta), P_\theta$

Note: to help be consistent with machine learning notations, we use y to substitute θ.

The joint probability $P(x, y) = \pi(y)P_y(x) = P(x)\pi(y|x)$.

Example 1. *The risk of decision $\delta(x)$ in Hamming/zero-one loss $l(a = \hat{y}, y) = \mathbf{1}_{a\neq y}$*

$$\begin{aligned}
R(\delta) &= \mathbb{E}(\mathbf{1}_{\delta(x)\neq y}) = \mathbb{E}[\delta(x) \neq y] \\
&= P(y=0)P[\delta(x) \neq 0|y=0] + P(y=1)P[\delta(x) \neq 1|y=1] \\
&= P(y=0)P[\delta(x) = 1|y=0] + P(y=1)P[\delta(x) = 0|y=1]
\end{aligned}$$

Bayes rule

$$\delta_B = \operatorname*{argmin}_\delta R(\delta)$$

Derive Bayes rule

$$\begin{aligned} R(\delta) &= \int_x \int_y P(x,y) l(\delta(x), y) dy dx \\ &= \int_x P(x) \int_y \pi(y|x) l(\delta(x), y) dy dx \end{aligned}$$

Solve optimization problem:

$$\min_{\delta} \int_x P(x) \int_y \pi(y|x) l(\delta(x), y) dy dx$$

this problem can be transformed into optimization problems for each $x \in S$

$$\min_{\delta(x)} \int_y \pi(y|x) l(\delta(x), y) dy$$

The problem becomes to compute $\pi(y|x)$. From $P(x,y) = \pi(y)P_y(x) = P(x)\pi(y|x)$, we know

$$\pi(y|x) = \frac{\pi(y)P_y(x)}{P(x)}$$

1.4.2 Maximum A Posteriori (MAP) Decision Rule (Binary example)

Example 2. *Hamming/zero-one loss* $l(a,y) = \mathbf{1}_{a \neq y}$

Maximum A Posteriori (MAP) Decision Rule:

Optimization problem is

$$\delta(x) = \operatorname*{argmin}_a \sum_{y=0,1} \pi(y|x) \mathbf{1}_{a \neq y} dy = \operatorname*{argmax}_{y \in \{0,1\}} \pi(y|x)$$

$$\Rightarrow \sum_{y=0,1} \pi(y|x) \mathbf{1}_{\delta(x) \neq y} dx = \min_a \sum_{y=0,1} \pi(y|x) \mathbf{1}_{a \neq y} dy = \min\{\pi(1|x), \pi(0|x)\}$$

Likelihood ratio: $L(x) = \frac{P_1(x)}{P_0(x)}$

Likelihood ratio test: threshold $\tau = \frac{\pi(0)}{\pi(1)}$. If $L(x) > \tau$ accept H_1 (equivalent to $P_1(x)\pi(1) > P_0(x)\pi(0)$ which is also equivalent to comparing $\pi(y|x)$).

In this rule the whole optimization problem also goes to

$$\begin{aligned} R(\delta_{MAP}) &= \int_x P(x) \sum_{y=0,1} \pi(y|x) \mathbf{1}_{\delta(x) \neq y} dx \\ &= \int_x P(x) \min\{\pi(1|x), \pi(0|x)\} dx \end{aligned}$$

1.4.3 Minimum Mean Squared Error (MMSE) Rule ($\mathbb{R}^n$ example)

Example 3. *(estimation) Squared error loss* $l(a,y) = (a-y)^2$.

Minimum Mean Squared Error (MMSE) Rule:

Optimization problem is $\delta(x) = \text{argmin}_a \int_y \pi(y|x)(a-y)^2 dy$

$$0 = \int_y \pi(y|x)(\delta_B(x) - y)dy = \delta_B(x) - \mathbb{E}[Y|X=x]$$

$$\Rightarrow \delta_B(x) = \mathbb{E}[Y|X=x]$$

which is called **conditional mean estimation**.

In this rule the whole optimization problem also goes to

$$R(\delta_{MMSE}) = \int_x P(x) \int_y \pi(y|x)(y - \mathbb{E}[Y|X=x])^2 dydx = \mathbb{E}_X Var[Y|X=x]$$

Gaussian case: If $X \in \mathbb{R}^n$ and (Y, X) are jointly Gaussian, then the conditional mean is a linear function of x, also called linear MMSE estimator.

$$\mathbb{E}[Y|X=x] = \mathbb{E}[Y] + Cov(Y,X)Cov(X)^{-1}(x - \mathbb{E}[X])$$

and the posterior risk is independent of x:

$$Var[Y|X=x] = Var[Y] - Cov(Y,X)Cov(X)^{-1}Cov(X,Y)$$

1.5 Comparison

Maximum-Likelihood Principle (state is norandom): $\delta_{ML}(x) = \text{argmax}_y P_y(x)$.

Maximum A Posteriori (MAP) Decision Rule (state is random): $\delta_{MAP}(x) = \text{argmax}_y \pi(y|x) = \text{argmax}_y\{\pi(y|x), P_y(x)\}$

2 Machine Learning in Inference

Instead of given a prior distribution of Y, we are given a **training set** $T = (X_i, Y_i)_{i=1}^n$ where i.i.d. $(X_i, Y_i) \sim P$. (Distribution P is unknown).

Risk: $R(\delta) = \mathbb{E}_P[l(\delta(X), Y)]$

The ture optimal decision rule is

$$\delta_B = \underset{\delta}{\text{argmin}}\, \mathbb{E}_P[l(\delta(X), Y)]$$

which is can't be computed since we don't know how actually P is.

2.1 Empirical Risk Minimization (ERM)

Instead of computing optimal decision rule with P, we compute the optimal decisioin rule in the training set:

$$\hat{\delta}_n = \underset{\delta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l(\delta(X_i), Y_i)$$

The corresponding risk is $R(\hat{\delta}_n) = \mathbb{E}_P \left[l(\hat{\delta}_n(X), Y) \right]$. $\Delta R(\hat{\delta}_n) = R(\hat{\delta}_n) - R(\delta) > 0$ always holds.

Consistency: if $\Delta R(\hat{\delta}_n) \to 0$ as $n \to \infty$.

2.1.1 Example: Linear MMSE (LMMSE) estimator

Use the decision role in the class of $\delta(x) = wx$. To find the linear MMSE (LMMSE) estimation $\delta^*(x) = w^* x$:

$$w^* = \underset{w}{\operatorname{argmin}} \mathbb{E}_P \left[(wX - Y)^2 \right] = \frac{\mathbb{E}[XY]}{\mathbb{E}[X^2]}$$

The rule that minimizes the **empirical risk** is

$$\hat{w} = \underset{w}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (wX_i - Y_i)^2 = \frac{\frac{1}{n}\sum_{i=1}^{n} X_i Y_i}{\frac{1}{n}\sum_{i=1}^{n} X_i^2}$$

The risk of the optimal rule $\delta^* = w^* x$ is $R(\delta^*)$ and the empirical risk under rule $\hat{\delta}(x) = \hat{w}x$ is $R(\hat{\delta}(x))$. $R(\hat{\delta}) > R(\delta^*)$ always holds, and

$$R(\hat{\delta}) \to R(\delta^*) \text{ as } n \to \infty$$

According to CLT:

$$\sqrt{n} \left(\frac{1}{n} \sum_i X_i Y_i - \mathbb{E}(XY) \right) \xrightarrow{d} N(0, \sigma^2)$$

$$\sqrt{n} \left(\frac{1}{n} \sum_i X_i^2 - \mathbb{E}(X^2) \right) \xrightarrow{d} N(0, \sigma^2)$$

Then,

$$\frac{1}{n} \sum_i X_i^2 = \mathbb{E}(X^2) + O(\frac{1}{\sqrt{n}})$$

$$\frac{1}{n} \sum_i X_i Y_i = \mathbb{E}(XY) + O(\frac{1}{\sqrt{n}})$$

which means the error of estimators

$$\hat{w} = \frac{\mathbb{E}(X^2) + O(\frac{1}{\sqrt{n}})}{\mathbb{E}(XY) + O(\frac{1}{\sqrt{n}})} = w^* + O(\frac{1}{\sqrt{n}})$$

$$\hat{w} - w^* = O(\frac{1}{\sqrt{n}})$$

and the error of the risks:

$$\begin{aligned} R(\hat{\delta}) - R(\delta^*) &= \mathbb{E}_P[\hat{w}X - Y]^2 - \mathbb{E}_P[w^*X - Y]^2 \\ &= \mathbb{E}_P[(\hat{w} - w^*)X + w^*X - Y]^2 - \mathbb{E}_P[w^*X - Y]^2 \\ &= \mathbb{E}_P[(\hat{w} - w^*)X]^2 + 2(\hat{w} - w^*)\mathbb{E}_P[X(w^*X - Y)] \\ &= (\hat{w} - w^*)\mathbb{E}_P(X^2) = O\left(\frac{1}{n}\right) \end{aligned}$$

Complexity:

Definition 1. *A sequence* $f(n)$ *is* $O(1)$ *if* $\lim_{n\to\infty} f(n) < \infty$.

Definition 2. *A sequence* $f(n)$ *is* $O(g(n))$ *if* $\frac{f(n)}{g(n)}$ *is* $O(1)$.

Definition 3. *A sequence* $f(n)$ *is* $o(1)$ *if* $\lim_{n\to\infty} \sup f(n) = 0$.

Definition 4. *A sequence* $f(n)$ *is* $o(g(n))$ *if* $\lim_{n\to\infty} \sup \frac{f(n)}{g(n)} = 0$.

Definition 5. *A sequence* $f(n)$ *is asymptotic to* $g(n)$ *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. *(This is denoted by* $f(n) \sim g(n)$ *as* $a \to \infty$*)*

2.1.2 Penalized ERM

$\delta(x) = \sum_{j=1}^{J} w_j x^j$

Pick $J = d$ and use ERM with d dimensional w:

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} [w^T X_i - Y_i]^2$$

Approach 1: Fix $d << n$, use ERM.

Approach 2: (**Penalized ERM**)

$$\min_{\delta} [R_{emp}(\delta) + J(\delta)]$$

($J(\delta)$ is regularization (penalty) term)

2.2 Stochastic Approximation

Robbins and Monro (95)

Problem: Find a root of function $h(x)$. $(f(x) = 0)$

We do not observe $h(x)$ directly, but we observe $Y \sim P_x$ with

(1) $\mathbb{E}[Y|X = x] = h(x)$

(2) $(Y|X = x) - h(x)$ is bounded

Example 4. *$Y = X + Z$ with $\mathbb{E}[Z] = 0$ and Z is bounded.*

Assumptions: 1. $h'(x^*) > 0$; 2. x^* is the unique root of h.

SA Algorithm

- Pick Sequence $\{a_n\}$ such that $\sum_{n=1}^{\infty} a_n = \infty$ and $\sum_{n=1}^{\infty} a_n^2 < \infty$ (should converge to 0 but not too quick) e.g. $a_n = n^{-\alpha}$ when $\alpha \in (\frac{1}{2}, 1]$.
- Initialize X_1
- Update for $n = 1, 2, ...$, $Y_n \sim P(\cdot|X = X_n)$

$$X_{n+1} = X_n - a_n Y_n$$

  until convergence.

Theorem 1. *Under these assumptions*

$$X_n \xrightarrow{m.s.} x^* \text{ as } n \to \infty$$

i.e., $\mathbb{E}(X_n - x^*)^2 \to 0$ *as* $n \to \infty$.

- **Performance Measure** (Convergence rate): the root mean squared error (RMSE) $e_n = \sqrt{\mathbb{E}[(X_n - x^*)^2]}$.
- Projections: If x is constrained to live in an interval I, the update rule becomes

$$X_{n+1} = \text{Proj}_x[X_n - a_n Y_n]$$

- Averaging:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i = \frac{X_n}{n} + \bar{X}_{n-1}\frac{n-1}{n}$$

  (nicer graph) (The benefits of this smoothing operation are mostly seen in the initial stages of the SA recursion, and do not improve the convergence rate.)

Example 5. *Let $h(x) = x$, in which case $x^* = 0$. $Y_n = h(X_n) + Z_n = X_n + Z_n$ where noise Z_n is independent of Y_n with $\mathbb{E}[Z_n] = 0, Var(Z_n) = 1$ and Z_n is bounded.*

Then,

$$\begin{aligned} X_{n+1} &= X_n - a_n(X_n + Z_n) \\ &= (1 - a_n)X_n - a_n Z_n \end{aligned}$$

The MSE,

$$\begin{aligned} e_{n+1}^2 &= \mathbb{E}(X_{n+1} - x^*)^2 = \mathbb{E}(X_{n+1})^2 \\ &= \mathbb{E}[(1 - a_n)X_n - a_n Z_n]^2 \\ &= (1 - a_n)^2 \mathbb{E}X_n^2 + a_n^2 \mathbb{E}Z_n^2 \\ &= (1 - a_n)^2 e_n^2 + a_n^2 \end{aligned}$$

Pick $a_n = n^{-\alpha}$, where $\alpha \in (\frac{1}{2}, 1]$

$$\Rightarrow e_{n+1}^2 = (1 - n^{-\alpha})^2 e_n^2 + n^{-2\alpha}$$

Guess: $e_n = \sqrt{c}n^{-\beta} + H.O.T$

$$c(n+1)^{-2\beta} + H.O.T = (1-n^{-\alpha})^2 cn^{-2\beta} + n^{-2\alpha} + H.O.T$$

(where $(n+1)^{-2\beta} = n^{-2\beta}(1+\frac{1}{n})^{-2\beta} = n^{-2\beta}[1 - 2\beta n^{-1} + O(n^{-2})]$, by Taylor)

$$cn^{-2\beta} - 2c\beta n^{-1-2\beta} + H.O.T = (1-n^{-\alpha})^2 cn^{-2\beta} + n^{-2\alpha} + H.O.T$$

$$-2c\beta n^{-1-2\beta} + H.O.T = -2cn^{-\alpha-2\beta} + n^{-2\alpha} + H.O.T$$

(For $\alpha < 1$), $-2c\beta n^{-1-2\beta}$ is not dominant term.

$$H.O.T = -2cn^{-\alpha-2\beta} + n^{-2\alpha} + H.O.T$$

Identify Power: $2\alpha = \alpha + 2\beta \Rightarrow \beta = \frac{\alpha}{2}$ and $c = \frac{1}{2}$

(For $\alpha = 1$), there are three dominant terms.

$$-2c\beta n^{-1-2\beta} + H.O.T = -2cn^{-1-2\beta} + n^{-2} + H.O.T$$

Identify Power: $2 = 1 + 2\beta \Rightarrow \beta = \frac{1}{2}$ and $-2c\beta = -2c + 1 \Rightarrow c = 1$

$$e_n^2 \sim cn^{-2\beta}$$

To let the convergence rate as fast as possible, we want the β to be as large as possible.

Since $\beta = \frac{\alpha}{2}$, we pick the highest $\alpha = 1 \Rightarrow \beta = \frac{1}{2}, c = 1$.

$$e_n = O(n^{-\frac{1}{2}}) \text{ with } a_n \sim \frac{1}{n}$$

Example 6. *Let $h(x) = x^3$, in which case $x^* = 0$. $Y_n = h(X_n) + Z_n = X_n^2 + Z_n$ where noise Z_n is independent of Y_n with $\mathbb{E}[Z_n] = 0, Var(Z_n) = 1$ and Z_n is bounded.*

Then,

$$\begin{aligned} X_{n+1} &= X_n - a_n(X_n^3 + Z_n) \\ &= X_n - a_n X_n^3 - a_n Z_n \end{aligned}$$

Pick $a_n = n^{-\alpha}$, $\alpha \in (\frac{1}{2}, 1] \Rightarrow \beta = \frac{1}{6}, \alpha = \frac{2}{3} \Rightarrow e_n \sim O(n^{-\frac{1}{6}})$

2.3 Stochastic Gradient Descent (SGD)

Solve $\min_{x\in\mathbb{R}^n} f(x)$.

We only use a **noisy version** $g(x,z)$ of $f(x)$, where $\mathbb{E}_z[g(x,z)] = f(x)$.

$$\mathbb{E}_z[\nabla_x g(x,z)] = \nabla_x \mathbb{E}_z[g(x,z)] = \nabla f(x)$$

Also pick sequence $\{a_n\}$ such that $\sum_{n=1}^{\infty} a_n = \infty$ and $\sum_{n=1}^{\infty} a_n^2 < \infty$.

SGD

- Initialize X_1
- Update for $n = 1, 2, ...,$

$$X_{n+1} = X_n - a_n \nabla g(X_n, Z_n)$$

Example 7. *$f(x) = \frac{1}{2}x^2, x \in \mathbb{R}$. Let Z be a random variable with $\mathbb{E}(Z) = 0, Var(Z) = 1$.*

$$g(x,Z) = \frac{1}{2}(x+Z)^2 - \frac{1}{2}$$

$$\mathbb{E}[g(x,Z)] = \frac{1}{2}x^2 = f(x)$$

$$\nabla_x g(x,Z) = x + Z \Rightarrow \mathbb{E}[\nabla_x g(x,Z)] = \nabla f(x)$$

$$X_{n+1} = X_n - a_n(X_n + Z_n)$$

which is the same as the stochastic approximation.

Main Results: (Suppose the unique minimum is x^*)

(1) Convergence: $e_n \to 0$ as $n \to \infty$.

(2) Convergence Rate: To achieve $\mathbb{E}[f(X_n)] - f(x^*) < \varepsilon$, we need $n = O(\frac{1}{\varepsilon})$ if f is twice continuously differentiable and strongly convex.

GD has linear convergence $\Rightarrow e_n = O(e^{-cn})$; Solve $\varepsilon = O(e^{-cn}) \Rightarrow n = O(\ln \frac{1}{\varepsilon})$. (**SGD is much worse than GD**, cost more.)

2.4 SGD Application to Empirical Risk Minimization (ERM)

ERM problem is

$$\min_{w\in\mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} L(\delta_w(X_i), Y_i)$$

$R_{emp}(w) = \frac{1}{n}\sum_{i=1}^{n} L(\delta_w(X_i), Y_i)$ is the empirical risk (e.g. $\delta_w(x) = w^T x$, $L(\hat{y}, y) = (\hat{y} - y)^2$) To make w more visible, we can write

$$R_{emp}(w) = \frac{1}{n}\sum_{i=1}^{n} L(\delta_w(X_i), Y_i) = \frac{1}{n}\sum_{i=1}^{n} Q(X_i, Y_i, w)$$

For penalized ERM we would similarly have

$$\min_{w\in\mathbb{R}^d} \frac{1}{n}\sum_{i=1}^{n} Q(X_i, Y_i, w) + J(w)$$

$J(w)$ is the penalty (regularization) term.

In the problem $\min_{W\in\mathbb{R}^n} R_{emp}(w) = \frac{1}{n}\sum_{i=1}^{n} Q(X_i, Y_i, w)$

2.4.1 Different Gradient Descent for ERM

GD

- Initialize W_1
- Update for $k \geq 1$, Update: $W_{k+1} = W_k - a_k \frac{1}{n}\sum_{i=1}^{n} \nabla Q(X_i, Y_i, W_k)$

Computational cost: The computational cost of GD is $O(dn)$ operations per iteration. Since GD has exponential convergence, the number of iterations needed to reach an optimization error of ρ is $O(\log \frac{1}{\rho})$. Hence GD incurs a total computational cost of $O(dn \log \frac{1}{\rho})$ to reach a solution W_k such that $R_{emp}(W_k) \leq \min_W R_{emp}(W) + \rho$

SGD

- Initialize W_1
- Update for $k \geq 1$,

  Step 1: Pick i uniformly over $\{1, ..., n\}$

  Step 2: $W_{k+1} = W_k - a_k \nabla Q(X_i, Y_i, W_k)$

Computational cost: After k iterations, $\mathbb{E}[R_{emp}(W_k)] \leq \min_W R_{emp}(W) + \rho$, for $k = O(\frac{1}{\rho})$ and f twice differentiable and strongly convex. The cost per iteration is $O(d)$ (independent of n), so the total computational cost is $O(\frac{d}{\varepsilon})$.

2.4.2 Constraints on Learning Problem

Why achieving a low value of ρ is useful (low error of $R_{emp}(\cdot)$), since the cost function $R_{emp}(\cdot)$ is only a surrogate for the actual risk $R(\cdot)$?

Typically, $d = O(n^b)$ where $0 < b < 1$.

For any numerical algorithm producing a decision rule $\tilde{\delta_n}$, the excess risk (compared to Bayes rule δ_B) can be expressed as the sum of three terms:

$$\begin{aligned}\Delta R(\tilde{\delta_n}) &= R(\tilde{\delta_n}) - R(\delta_B) \\ &= [R(\tilde{\delta_n}) - R(\hat{\delta_n})] + [R(\hat{\delta_n}) - R(\delta^*)] + [R(\delta^*) - R(\delta_B)]\end{aligned}$$

where

$$\begin{aligned}\delta_B &= \text{ Bayes rule} \\ \delta^* &= \text{ best rule in } D = \underset{\delta \in D}{\operatorname{argmin}} R(\delta) \\ \hat{\delta_n} &= \underset{\delta \in D}{\operatorname{argmin}} R_{emp}(\delta) \\ \tilde{\delta_n} &= \text{solution of the algorithm after } k \text{ iterations}\end{aligned}$$

(Note: D is the set of all available decision rule in approximation (e.g. all linear parameters $\{W, b\}$), which can't be better than Bayes rule)

The expected excess risk

$$\epsilon = \mathbb{E}[\Delta R(\tilde{\delta_n})] = \underbrace{\mathbb{E}[R(\tilde{\delta_n}) - R(\hat{\delta_n})]}_{\text{Comp. Error}=\rho} + \underbrace{\mathbb{E}[R(\hat{\delta_n}) - R(\delta^*)]}_{\text{Est. Error}=O(\frac{d}{n})} + \underbrace{\mathbb{E}[R(\delta^*) - R(\delta_B)]}_{\text{Approx. Error}=O(d^{-\beta})}$$

Estimation error increases as d increases, but *approximation error* decreases as d increases. To minimize the excess risk, we want to balance the last two items, that is $O(\frac{d}{n}) = O(d^{-\beta})$: solve

$$\frac{d}{n} = d^{-\beta} \Rightarrow d^{1+\beta} = n \Rightarrow d = n^{\frac{1}{1+\beta}}$$

$$\Rightarrow \text{ the last two items } O(\frac{d}{n}) = O(d^{-\beta}) = O(n^{-\gamma})$$

where $\gamma = \frac{\beta}{1+\beta} \in (0, 1]$ is a constant.

To balance the three items, we want

$$\rho = O(n^{-\gamma}) \Rightarrow n = O(\rho^{-\frac{1}{\gamma}}) \text{ and } d = O(n^{\frac{1}{1+\beta}})$$

The update rule $W_{k+1} = W_k - a_k \nabla Q(X_k, Y_k, W_k)$, where $i \sim \text{Uniform}\{1, 2, ..., n\}$

Relation to Online Learning: When the training data are made available sequentially (instead of in a batch as assumed here), online learning can be used to sequentially learn the decision rules (or the weights that parameterize the decision rule).

Variations on Basic SGD: mini batch: replace S by a subset B and n by $|B|$

$$\frac{1}{|B|} \sum_{i \in B} Q(X_i, Y_i, W_k)$$

Averaging SGD:

$$\bar{W}_n = \frac{1}{n}\sum_{i=1}^{n} W_i = \frac{W_n}{n} + \bar{W}_{n-1}\frac{n-1}{n}$$

SVRG (Stochastic Variance Randomed Gradient): R. Johnson and T. Zhang, "Accelerating Stochastic Gradient Descent using Predictive Variance Reduction," *Proc. NIPS* 2013.

Unsupervised learning: If no explanatory variable X is present, the problem reduces to

$$\min_w \frac{1}{n}\sum_{i=1}^{n} Q\left(Y_i, w\right)$$

which finds applications to various unsupervised learning problems. For instance the k-means clustering algorithm partitions n data points $y_i, 1 \le i \le n$ in $\mathbb{R}^d$ into k clusters with centroids $w_j, 1 \le j \le k$, in a way that minimizes the within-cluster sum-of-squares: WCSS $= \frac{1}{n}\sum_{i=1}^{n} \min_{1\le j\le k} \|y_i - w_j\|^2$. Using the formalism of (7), we have $Q(y, w) = \min_{1\le j\le k} \|y - w_j\|^2$ where $y \in \mathbb{R}^d$ and $w = \{w_j\}_{j=1}^k \in \mathbb{R}^{d\times k}$ is a matrix whose k columns are the centroid vectors.

3 Stachastic Integradtion Methods

Integral $I = \int_X f(x)dx$.

3.1 Deterministic Methods (Better in Low Dimension)

3.1.1 Riemann Integration

Riemann integral: approximation integral I of $f(x)$ in $[a, b]$ with

$$\hat{I}_n = \sum_{i=1}^{n} \underbrace{(x_i - x_{i-1})}_{\frac{b-a}{n}} f(x_i)$$

where $x_i = a + \frac{b-a}{n}i$. We can also denote $\hat{f}(x) = f(x_i)$ if $x \in (x_{i-1}, x_i]$

The error $|\hat{I}_n - I| = \int_a^b |\hat{f}(x) - f(x)|dx$

Assume f is differentiable and $\max_x |f'(x)| = c < \infty$, then $|\hat{f}(x) - f(x)| \le \frac{b-a}{n}c$

$$\Rightarrow |\hat{I}_n - I| \le \int_a^b \frac{b-a}{n}c\, dx = \frac{(b-a)^2}{n}c$$

That is $n \sim O\left(\varepsilon^{-1}\right)$

3.1.2 Trapezoidal Rule

Using average can be better.

$$\hat{I}_n = \sum_{i=1}^{n} (\underbrace{x_i - x_{i-1}}_{\frac{b-a}{n}}) \frac{f(x_i) + f(x_{i-1})}{2}$$

The upper bound of error is $|\hat{I}_n - I| \leq \frac{C}{n^2}$ for some constant C.

That is $n \sim O\left(\varepsilon^{-\frac{1}{2}}\right)$

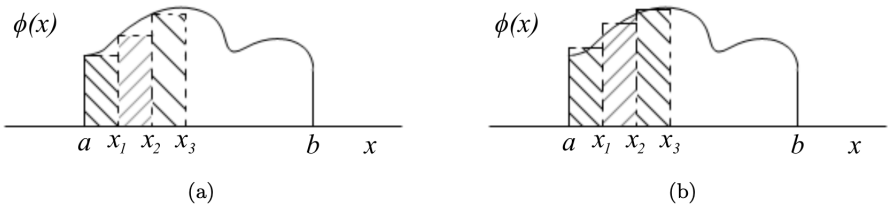


Figure 1: (a) Riemann approximation; (b) Trapezoidal approximation.

3.1.3 Multidimensional Integration

When we want to do intergral in high dimention, it will be really hard.

For $d-$dimensional integrals, the trapezoidal rule yields an approximation error $|\hat{I}_n - I| \leq \frac{C}{n^{\frac{2}{d}}}$ for some constant C. That is $n \sim O\left(\varepsilon^{-\frac{d}{2}}\right)$. n needs to increase exponentially with d to achieve a target approximation error ε. This phenomenon is known as the curse of dimensionality.

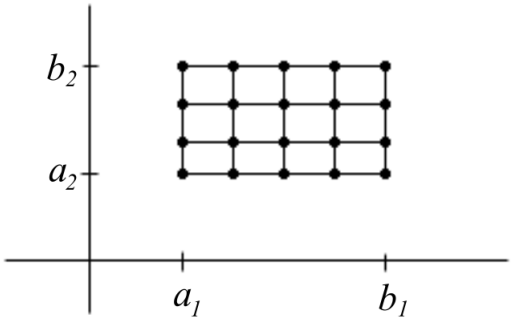


Figure 2: Two-dimensional integration using regular grid.

3.2 Stachastic Methods (Better in High Dimension)

3.2.1 Classical Monte Carlo Integration

Compute the expetation

$$\xi = \mathbb{E}_p[h(x)] = \int_X \underbrace{p(x)h(x)}_{f(x)} dx$$

The methods described below can be used to solve the following problems: (1) General $\int_X f$; (2) Compute the probability of falling into a subset $a \subset X : P(a) = \int_a p(x)dx$, where $h(x) = \mathbf{1}_{x \in a}$

The Monte Carlo approach is as follows: Given $X_1, X_2, \cdots, X_n$ drawn i.i.d from the pdf p, estimate ξ by the empirical average

$$\hat{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} h(X_i)$$

$\mathbb{E}_p[\hat{\xi}_n] = \mathbb{E}_p[h(X)] = \xi$. $\hat{\xi}_n \xrightarrow{a.s.} \xi$ as $n \to \infty$ by SLLW.

$Var(\hat{\xi}_n - \xi) = Var(\hat{\xi}_n) = \frac{1}{n} Var[h(x)] = O\left(\frac{1}{n}\right) \Rightarrow sd(\hat{\xi}_n) = \frac{\sqrt{Var[h(x)]}}{\sqrt{n}}$

That is $n \sim O\left(n^{-\frac{1}{2}}\right)$

The stochastic methods **outperform** when the deterministic ones for dimensions $\underline{d > 4}$ and are **worse** for $\underline{d < 4}$.

3.2.2 Importance Sampling

Draw $X_i, i = 1, ..., n$ i.i.d from pdf q

$$\hat{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} \frac{p(X_i)}{q(X_i)} h(X_i)$$

It is an unbiased estimator of ξ

$$\mathbb{E}[\hat{\xi}_n] = \mathbb{E}_q[\frac{p(X_i)}{q(X_i)} h(X_i)] = \int_X p(x)h(x)dx = \xi$$

$\hat{\xi}_n \xrightarrow{a.s.} \xi$ as $n \to \infty$ by SLLW.

Its variance is

$$\begin{aligned} Var_q(\hat{\xi}_n) &= \frac{1}{n} Var_q \left[\frac{p(X_i)}{q(X_i)} h(X_i) \right] \\ &= \frac{1}{n} \left(\int_X \frac{p^2(x)}{q(x)} h^2(x)dx - \xi^2 \right) \end{aligned}$$

The idea of importance sampling is to find a good q such that

$$Var_q(\hat{\xi}_n) < Var_p(\hat{\xi}_n)$$

Error Meausre

The *relative error* of the importance-sampling estimator is defined as

$$\delta_{\text{rel}}\left(\hat{\xi}_n\right) \triangleq \frac{\sqrt{\text{Var}_q\left(\hat{\xi}_n\right)}}{\xi} = \sqrt{\frac{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}{\xi^2 n}}.$$

The *number* of simulations needed to achieve a relative error of δ is

$$n_{IS}(\delta) = \frac{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}{\xi^2\delta^2}.$$

The *gain* relative to a Monte Carlo simulation is defined as

$$\Gamma = \frac{n_{MC}(\delta)}{n_{IS}(\delta)} = \frac{\text{Var}_p[h(X)]}{\text{Var}_q\left[\frac{p(X)}{q(X)}h(X)\right]}$$

In the example of $\xi = P(a), h(x) = \mathbf{1}_{x\in a}$: Suppose $\xi = P(a) \approx 10^{-9}$ (small), $\hat{\xi}_n = \frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{X_i\in a}$. $\mathbf{1}_{X_i\in a}$ is $Bernoulli(\xi)$. We have $Var(\hat{\xi}_n) = \frac{\xi(1-\xi)}{n}$.

We can use relative error to measure

$$\delta_{\text{rel}}\left(\hat{\xi}_n\right) = \frac{\sqrt{Var(\hat{\xi}_n)}}{\xi} = \sqrt{\frac{1-\xi}{n\xi}}$$

and the number of simulation need to get relative error δ is

$$n_{IS}(\delta) = \frac{1-\delta}{\xi\delta^2}.$$

Find the optimal q:

$$\min_q \int_X \frac{p^2(x)}{q(x)}h^2(x)dx - \xi^2$$

write

$$\int_X \frac{p^2(x)}{q(x)}h^2(x)dx - \xi^2 = \mathbb{E}_q\left[\left(\underbrace{\frac{p(x)}{q(x)}h(x)}_{Z}\right)^2\right]$$

Since x^2 is convex function, by Jensen's inequality

$$\mathbb{E}_q\left[\left(\underbrace{\frac{p(x)}{q(x)}h(x)}_{Z}\right)^2\right] \geq \left(\mathbb{E}_q\left[\underbrace{\frac{p(x)}{q(x)}h(x)}_{Z}\right]\right)^2$$

This equality holds if and only if $\frac{p(x)}{q(x)}h(x) = \alpha, \forall x \in X$, α is a constant.

Since q is pdf., we can infer

$$q(x) = \frac{p(x)h(x)}{\int_X p(x)h(x)dx}$$

which is as hard as the original problem. In practice, one is content to find a "good" q that assigns high probability to the important region where $p(x)h(x)$ is large. Ideally the ratio $\frac{p(x)}{q(x)}h(x)$ would be roughly constant over X.

4 Bootstrap (not enough data)

Problem: analyze the performance of an estimator $\hat{\theta}_n(\vec{Y})$, $\vec{Y} = (Y_1, Y_2, ..., Y_n)$ taken i.i.d. from distribution P. e.g. $P_\theta = N(0,1), \hat{\theta}_n = \frac{1}{n}\sum_{i=1}^n Y_i$

Assume θ is a scalar parameter. Performance: (1) Bias $\mathbb{E}_\theta[\hat{\theta}_n(\vec{Y})] - \theta$; (2) Variance $\mathbb{E}_\theta[\hat{\theta}_n^2(\vec{Y})] - \mathbb{E}_\theta^2[\hat{\theta}_n(\vec{Y})]$; (3) CDF $G_n(t) = P(\hat{\theta}_n(\vec{Y}) < t), \forall t$

Approach #1 Monte-Carlo Simulations

Generate k vectors $\vec{Y}^{(i)}, i = 1, 2, ..., k$ (total kn random variables) (1) Bias $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y}^{(j)}) - \theta$; (2) Variance $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n^2(\vec{Y}^{(j)}) - \left(\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y}^{(j)})\right)^2$; (3) CDF $\hat{G}_n(t) = \frac{1}{k}\sum_{j=1}^k \mathbf{1}_{\hat{\theta}_n(\vec{Y}^{(j)})<t}, \forall t$

Approach #2 Bootstrap

(When data is not enough) Suppose we only have one data $\vec{Y} = (Y_1, ..., Y_n)$

Reuse $Y_1, ...Y_n$ to obtain resamples $\vec{Y}^* = (Y_1^*, ..., Y_n^*)$. Do this k times $\Rightarrow$ k resamples $\vec{Y^*}^{(1)}, ..., \vec{Y^*}^{(k)}$

(1) Bias $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y^*}^{(j)}) - \theta$; (2) Variance $\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n^2(\vec{Y^*}^{(j)}) - \left(\frac{1}{k}\sum_{j=1}^k \hat{\theta}_n(\vec{Y^*}^{(j)})\right)^2$; (3) CDF $\hat{G}_n(t) = \frac{1}{k}\sum_{j=1}^k \mathbf{1}_{\hat{\theta}_n(\vec{Y^*}^{(j)})<t}, \forall t$

Example: $\theta = med\{P\}$, P is an unknown distribution over $\{0, 1, ..., 9\}$. $\vec{Y} = (4, 8, 9, 6, 2)$.

4.1 Residual Bootstrap

The bootstrap principle is quite general and may also be used in problems where the data Y_i, $1 \le i \le n$, **are not i.i.d**.

Example: Linear

Observation $Y_i = a + b\frac{i}{n} + Z_i$, where $Z_i \sim N(0, \sigma^2)$ (i.i.d.) for $i = 1, 2, ..., n$

Parameter $\theta = (a, b)$. Linear Least Square Estimator:

$$(\hat{a}_n, \hat{b}_n) = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - a - b\frac{i}{n})^2$$

Given $\vec{Y}$, the residual (not i.i.d.)

$$E_i = Y_i = \hat{a}_n - \hat{b}_n \frac{i}{n} \approx Z_i$$

Generate k resamples of $\vec{E} = (E_1, E_2, ..., E_n)$

$\Rightarrow$ obtain $\vec{E^*}^{(1)}, \vec{E^*}^{(2)}, ..., \vec{E^*}^{(k)}$ by resampling

$\Rightarrow$ Compute pseudo-data $Y_i^{*(j)} = \hat{a}_n + \hat{b}_n \frac{i}{n} + E_i^{*(j)}$

$\Rightarrow$ Compute LS estimator

$$\hat{\theta}_n^{(j)} = (\hat{a}_n^{(j)}, \hat{b}_n^{(j)}) = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i^{*(j)} - a - b\frac{i}{n})^2$$

$\Rightarrow$ Evaluate bias

$$\widehat{Bias} = \frac{1}{k} \sum_{j=1}^{k} \hat{\theta}_n^{(j)} - \theta$$

Example: Nonlinear Markov Process

Observation $Y_i = F_\theta(Y_{i-1}) + Z_i$, where $Z_i \sim N(0, \sigma^2)$ (i.i.d.) for $i = 1, 2, ..., n$

Parameter $\theta = (a, b)$. Linear Least Square Estimator:

$$\hat{\theta}_n(\vec{Y}) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i - F_\theta(Y_{i-1}))^2$$

Given $\vec{Y}$, the residual (not i.i.d.)

$$E_i = Y_i = \hat{a}_n - F_{\hat{\theta}_n}(Y_{i-1}) \approx Z_i$$

Generate k resamples of $\vec{E} = (E_1, E_2, ..., E_n)$

$\Rightarrow$ obtain $\vec{E^*}^{(1)}, \vec{E^*}^{(2)}, ..., \vec{E^*}^{(k)}$ by resampling

$\Rightarrow$ Fix $Y_0^{*(j)} = Y_0$, compute pseudo-data $Y_i^{*(j)} = F_{\hat{\theta}_n}({Y^*_{i-1}}^{(j)}) + E_i^{*(j)}$

$\Rightarrow$ Compute LS estimator

$$\hat{\theta}_n^{(j)} = \underset{(a,b)}{\operatorname{argmin}} \sum_{i=1}^{n} (Y_i^{*(j)} - F_{\hat{\theta}_n}({Y^*_{i-1}}^{(j)}))^2$$

$\Rightarrow$ Evaluate bias

$$\widehat{Bias} = \frac{1}{k} \sum_{j=1}^{k} \hat{\theta}_n^{(j)} - \theta$$

5 Particle Filtering

Kalman filtering is used in tracking problems (dynamic models). Particle Filtering is an extension of Kalman filtering.

5.1 Kalman Filtering

1. Unknown state sequence $X_t \in \mathbb{R}^m, t = 0, 1, 2, ...$
2. Observations $Y_t \in \mathbb{R}^k, t = 0, 1, 2, ...$
3. $X_{t+1} = F_t X_t + U_t$, $F_t \in \mathbb{R}^{m \times m}, U_t \sim P_{U_t}$
4. $Y_t = H_t X_t + V_t$, $H_t \in \mathbb{R}^{k \times m}, V_t \sim P_{V_t}$

We want to solve two problems

1. **Estimation Problem:** Evaluate Linear MMSE (LMMSE) of X_t given $Y_{0:t}$.

$$\hat{X}_{t|t} = WY_{0:t} + b$$

2. **Prediction Problem:** Predict Linear MMSE (LMMSE) of $X_{t+1|t}$ given $Y_{0:t}$. (Really hard)

5.2 Particle Filtering

Particle filtering is a nonlinear form of Kalman filtering.
We consider a Nonlinear Dynamic System

$$X_{t+1} \sim q(\cdot|X_t)$$
$$Y_t \sim r(\cdot|X_t)$$
$$t = 0, 1, 2, ...$$

where $q(X_{t+1}|X_t)$ is the transition probability distribution, and $r(Y_t|X_t)$ is the conditional probability distribution for the observations. Hence, X_t is a Markov process and Y_t follows a Hidden Markov Model (HMM).

We also consider these two probelms.

1. **Estimation Problem:** Evaluate X_t given $Y_{0:t}$.
2. **Prediction Problem:** Predict $X_{t+1|t}$ given $Y_{0:t}$.

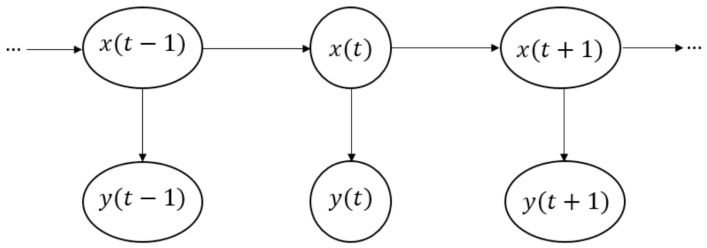


Figure 3: Hidden Markov Model

5.2.1 Bayesian Recursive Filtering

In this section we use Bayesian approach and use MMSE estimation $l(\hat{x}_t, x_t) = \|x_t - \hat{x}_t\|^2$

Estimation and prediction in conditional forms are:

$$\hat{X}_{t|t} = \mathbb{E}[X_t|Y_{0:t}] = \int_{\mathbb{R}^m} x_t P(X_t|Y_{0:t}) dx_t$$
$$\hat{X}_{t+1|t} = \mathbb{E}[X_{t+1}|Y_{0:t}] = \int_{\mathbb{R}^m} x_{t+1} P(X_{t+1}|Y_{0:t}) dx_{t+1}$$

Apparently the posterior p.d.f cannot be evaluated due to the curse of dimensionality as t increases. However, they can in principle be evaluated *recursively* using the following two-step procedure.

Step 1: Prediction. $P(X_{t+1}|Y_{0:t})$ can be expressed in term of $P(X_t|Y_{0:t})$:

$$\begin{aligned} P(X_{t+1}|Y_{0:t}) &= \int_{\mathbb{R}^m} P(X_{t+1}, X_t|Y_{0:t}) dx_t \\ &= \int_{\mathbb{R}^m} P(X_{t+1}|X_t, Y_{0:t}) P(X_t|Y_{0:t}) dx_t \\ &= \int_{\mathbb{R}^m} q(X_{t+1}|X_t) P(X_t|Y_{0:t}) dx_t \end{aligned}$$

Step 2: Update. We can also express $P(X_t|Y_{0:t})$ in terms of $P(X_t|Y_{0:t-1})$

$$\begin{aligned} P(X_t|Y_{0:t}) &= P(X_t|Y_t, Y_{0:t-1}) \\ &= \frac{P(Y_t|X_t, Y_{0:t-1}) P(X_t|Y_{0:t-1})}{P(Y_t|Y_{0:t-1})} \\ &= \frac{r(Y_t|X_t) P(X_t|Y_{0:t-1})}{\int_{\mathbb{R}^m} r(Y_t|X_t) P(X_t|Y_{0:t-1}) dx_t} \end{aligned}$$

5.2.2 Particle Filter (bootstrap filter)

Suppose we have n i.i.d. samples of X_t drawn from $p(x_t|Y_{0:t})$: $X_t(1), X_t(2), ..., X_t(n)$.

$$X_t(i) \sim p(\cdot|Y_{0:t}), 1 \leq i \leq n \qquad \text{(Sample 1)}$$

We can use above recursive filtering method to generate estimation of X_{t+1}.

Step 1: Prediction. Using the transition probability $q(\cdot|X_t(i)), 1 \leq i \leq n$ to generate n independent random variables

$$X^*_{t+1}(i) \sim q(\cdot|X_t(i)),\ 1 \leq i \leq n \tag{Sample 2}$$

Step 2: Update. Upon receiving a new measurement y_{t+1}, evaluate the *importance weights* (non-negative and summing to 1)

$$w_i = \frac{r(y_{t+1}|X^*_{t+1}(i))}{\sum_{j=1}^n r(y_{t+1}|X^*_{t+1}(j))},\ 1 \leq i \leq n$$

Then we resample n times from the set $\{X^*_{t+1}(i)\}_{i=1}^n$ with respective probabilities $\{w_i\}_{i=1}^n$, obtaining i.i.d samples $\{X_{t+1}(j)\}_{j=1}^n$ with probabilities

$$Pr[X_{t+1}(j) = X^*_{t+1}(i)] = w_i,\ 1 \leq i, j \leq n \tag{Sample 3}$$

By the **weighted bootstrap theorem**, as $n \to \infty$, the distribution of the resampled $\{X_{t+1}(j)\}_{j=1}^n$ converges to the desired posterior.

Potential issues: 1. n is not large enough. 2. Sample impoverishment

6

6.1 EM Algorithm

Problem of Expectation Maximization Observation $Y \sim P_\theta$ $\theta \in S$ is an unknown parameter.

ML: $\hat{\theta}_{ML} = \text{argmax}_{\theta \in S} \ln P_\theta(y)$

There is a complete data space Z and an incomplete data space Y. There is a many-to-one mapping $h(z) = y$. Z has p.d.f. $q_\theta(z)$ and Y has p.d.f. $p_\theta(y)$. We have

$$p_\theta(y) = \sum_{z \in h^{-1}(y)} q_\theta(z),\ \forall y$$

We can define the log-likelihood

$$l_{id}(\theta) \triangleq \ln p_\theta(y);\ l_{cd}(\theta) \triangleq \ln q_\theta(z)$$

EM Algorithm

1. Initialize $\hat{\theta}^0$

2. For $k = 0, 1, 2, ...$

   E-Step

   M-Step

Definition 6. *θ^* is a stable point of the EM algorithm if $\exists$ subsequence that converges to θ^*.*

e.g. $1, 3, \frac{1}{2}, 3, \frac{1}{3}, 3, ...\frac{1}{n}, 3, ...$

Example: Estimation of Variance of GRV

Observation $Y = S + N$, $S \sim \mathcal{N}(0, \theta)$ is independent of $N \sim \mathcal{N}(0, \theta) \Rightarrow Y \sim \mathcal{N}(0, \theta + 1)$

$p_\theta(y) = \frac{1}{\sqrt{2\pi(\theta+1)}} e^{-\frac{y^2}{2(\theta+1)}}$

$$\ln p_\theta(y) = -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\theta + 1) - \frac{y^2}{2(\theta + 1)}$$

take derivation of θ

$$0 = -\frac{1}{2(\theta + 1)} + \frac{y^2}{2(\theta + 1)^2}$$

We can get

$$\hat{\theta} = y^2 - 1$$

Then,

$$\hat{\theta}_{ML} = \begin{cases} 0, & y^2 \leq 1 \\ y^2 - 1, & y^2 > 1 \end{cases}$$

EM Algorithm Let $Z = (S, N)$, $y = s + n = h(z)$.

$$q_\theta(z) = q_\theta(s, n) = \frac{1}{\sqrt{2\pi\theta}} e^{-\frac{s^2}{2\theta}} \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}}$$

Then

$$l_{cd}(\theta) = \ln q_\theta(z) = \ln \frac{1}{\sqrt{2\pi}} e^{-\frac{n^2}{2}} - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\theta) - \frac{s^2}{2\theta}$$

7 Deep Learning

7.1 Parameters and Hyperparameters

Definition 7. ***Parameters*** *are values learned by the model given the data.*

e.g. β, W, b, θ.

Definition 8. ***Hyperparameters*** *are values supplied to tune the model and cannot be learned from data.*

e.g. Number of Hidden Layers, Neurons, and Epochs to Train. Learning Rate, and Batch Size.

7.2 Neural Network: Back Propagation Algorithm

Neural Network

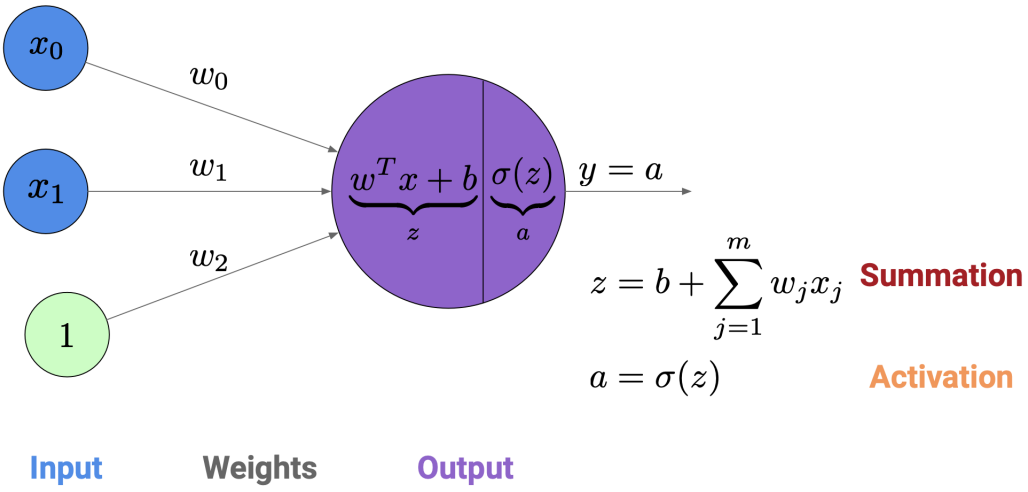


Figure 4: Simple Neural Network

Given a vector input x, we need to find the best estimator $\hat{y}$ which minimizes lost function. In the figure that has only one layer and one pathway, we find the parameter (ω, b) to form an input $\omega^T x + b$ to neuron σ (activation function). Then, the final output (estimator) of the network is $\hat{y} = \sigma(\omega^T x + b)$.

7.2.1 Activations

Definition 9. *Activation functions are element-wise gates for letting information propagate to future layers either transformed with non-linearities or left as-is.*

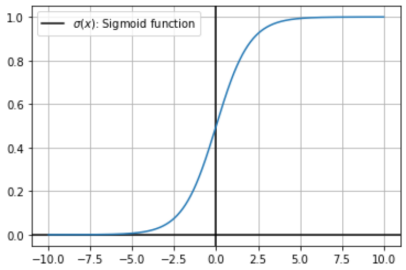
Example of activation function:

(1) **Identity:**

$$\text{identity}(x) = I(x) = x$$

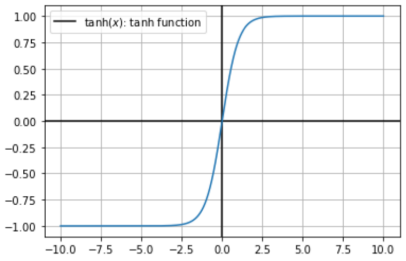
Sigmoid

$$\frac{1}{1+\exp(-x)}$$



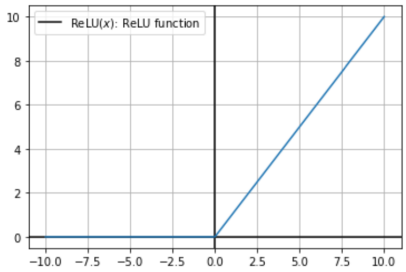
TanH

$$\frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$$



ReLU

$$\max(0,x)$$



Leaky ReLU

$$\max(0.1x,x)$$

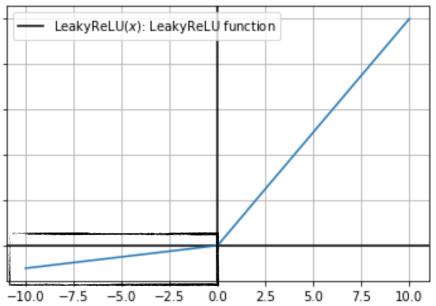


Figure 5: Activations

(2) **Binary:**

$$\text{binary}(x) = \text{step}(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

(3) **Sigmoid:**

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

$$\sigma(-x) = 1 - \sigma(x); \quad \frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$$

$$\frac{\partial \sigma(\vec{x})}{\partial \vec{x}} = \begin{bmatrix} \sigma(x_1)(1-\sigma(x_1)) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma(x_n)(1-\sigma(x_n)) \end{bmatrix} = \text{diag}(\sigma(\vec{x}) \cdot (1 - \sigma(\vec{x})))$$

(4) **TanH:**

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(TanH is a rescaled sigmoid)

$$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} = 1 - 2\sigma(-2x) = 2\sigma(2x) - 1$$

(5) **ReLU:**

$$g(x) = \max(0, x)$$

(6) **Leaky ReLU:**

$$g(x) = \max(0.1x, x)$$

(7) **Softmax:** $S_j(\vec{x}) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$,

$$S(\vec{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right]^T$$

$$\frac{\partial S_j(\vec{x})}{\partial x_j} = S_j(\vec{x})[1 - S_j(\vec{x})]$$

7.2.2 Multilayer Neural Network

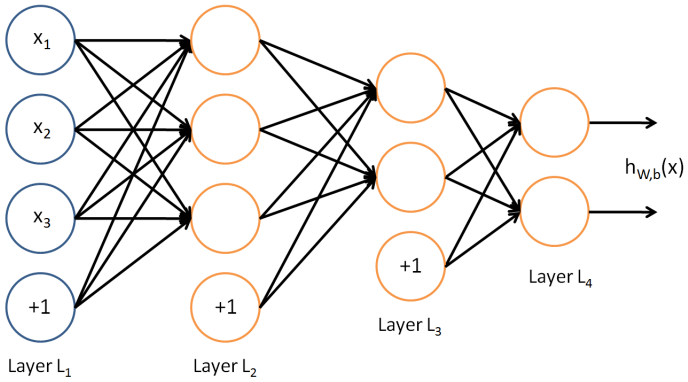


Figure 6: Multilayer Neural Network

- Number of neurons in each layer can be different.
- All weights on edge connecting layers $m-1$ and m is matrix $W^{(m)}$, with $w_{ij}^{(m)}$ being the weight connecting output j of layer $m-1$ with neuron i of layer m.
- Input to network is vector x; output of layer m is vector $y^{(m)}$

$$y_i^{(1)} = \sigma(x_i^{(1)}), \text{ with } x_i^{(1)} = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$y^{(1)} = \sigma(x^{(1)}), \text{ with } x^{(1)} = W^{(1)}x + b^{(1)}$$

$$y^{(2)} = \sigma(x^{(2)}), \text{ with } x^{(2)} = W^{(2)}y^{(1)} + b^{(2)}$$

$$\vdots$$

$$y^{(M)} = \sigma(x^{(M)}), \text{ with } x^{(M)} = W^{(M)}y^{(M-1)} + b^{(M)}$$

We want to find the weights $W^{(1)}, \cdots, W^{(M)}, b^{(1)}, \cdots, b^{(M)}$ so that the output of last layer

$$\hat{y} = y^{(M)} \approx f^*(x) = y$$

$f^*(x)$ is the unknown thing we need to predict.

We use labelled training data, i.e.

$$(x[1], y[1]), (x[2], y[2]), \cdots (x[N], y[N])$$

Minimize the "empirical" loss on training data.

$$J = \sum_{i=1}^{N} L(y[i], \hat{y}[i])$$

where $\bar{y}[i]$ is the output of NN whose input is $x[i]$.

- L is the function of $W^{(1)}, \cdots, W^{(M)}, b^{(1)}, \cdots, b^{(M)}$ to measure the loss. e.g. the square loss

$$L(y, \hat{y}) = (y - \hat{y})^2$$

- We wish to minimize J using a gradient descent procedure.

- To compute gradient we need:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} \text{ for each } l, i, j; \quad \frac{\partial L}{\partial b_i^{(l)}} \text{ for each } l, i.$$

7.2.3 A Simple Example of Back Propagation Algorithm

We can consider a simple example (one layer, two pathways)

$W^{(1)} = [w_{1,1}^{[1]}, w_{2,1}^{[1]}]^T$, $b^{(1)} = [0, 0]^T$, $\sigma_1(x) = x$.
$W^{(2)} = [w_{1,1}^{[2]}, w_{1,2}^{[2]}]$, $b^{(2)} = 0$, $\sigma_2(x) = x$.

$$[a_1^{[1]}, a_2^{[1]}]^T = \sigma_1(W^{(1)} x_1 + b^{(1)}) = [w_{1,1}^{[1]} x_1, w_{2,1}^{[1]} x_1]^T$$

$$\hat{y} = \sigma_2(W^{(2)} [a_1^{[1]}, a_2^{[1]}]^T + b^{(2)}) = (w_{1,1}^{[1]} w_{1,1}^{[2]} + w_{2,1}^{[1]} w_{1,2}^{[2]}) x_1$$

$$\frac{\partial J(\hat{y})}{\partial w_{2,1}^{[1]}} = \frac{\partial J(\hat{y})}{\partial \hat{y}} \cdot \frac{\hat{y}}{\partial a_2^{[1]}} \cdot \frac{a_2^{[1]}}{\partial w_{2,1}^{[1]}}$$

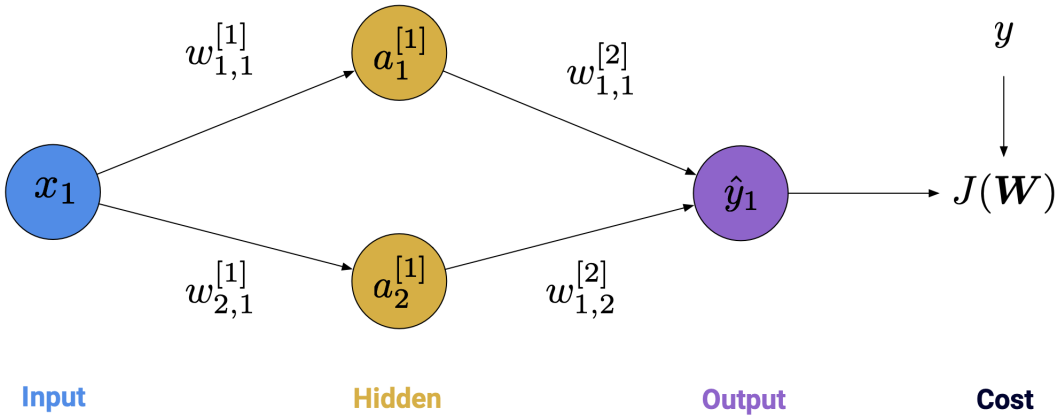


Figure 7: Two Independent Pathways

7.2.4 Back Propagation Algorithm

Recall $y_i^{(m)} = \sigma(x_i^{(m)})$, $x_i^{(m)} = \sum_j w_{ij}^{(m)} y_j^{(m-1)} + b_i^{(m)}$

$$\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial w_{ij}^{(m)}}$$

$$\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}}$$

For large M,

- $\frac{\partial L}{\partial y_i^{(M)}}$ is easy to compute.
- $\frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} = \frac{\partial \sigma(x_i^{(M)})}{\partial x_i^{(M)}} = \sigma'(x_i^{(M)})$, (assuming σ differentiable).
- $\frac{\partial x_i^{(M)}}{\partial w_{ij}^{(M)}} = y_j^{(M-1)}$

Thus,

$$\frac{\partial L}{\partial w_{ij}^{(M)}} = \frac{\partial L}{\partial y_i^{(M)}} \cdot \sigma'(x_i^{(M)}) \cdot y_j^{(M-1)}$$

Similarly,

$$\begin{aligned}\frac{\partial L}{\partial b_i^{(M)}} &= \frac{\partial L}{\partial y_i^{(M)}} \cdot \frac{\partial y_i^{(M)}}{\partial x_i^{(M)}} \cdot \frac{\partial x_i^{(M)}}{\partial b_i^{(M)}} \\ &= \frac{\partial L}{\partial y_i^{(M)}} \cdot \sigma'(x_i^{(M)})\end{aligned}$$

For $1 \leq m < M$, in this situation $\frac{\partial L}{\partial y_i^{(m)}}$ is not easy to compute. Note that $x^{(m+1)} = W^{(m+1)} y^{(m)} + b^{(m+1)}$.

$$\begin{aligned}
\frac{\partial L}{\partial y_i^{(m)}} &= \sum_k \frac{\partial L}{\partial x_k^{(m+1)}} \cdot \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} \\
&= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \frac{\partial y_k^{(m+1)}}{\partial x_k^{(m+1)}} \cdot \frac{\partial x_k^{(m+1)}}{\partial y_i^{(m)}} \\
&= \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \sigma'(x_k^{(m+1)}) \cdot w_{ki}^{(m+1)}
\end{aligned}$$

Then use this form to compute,

(We can set $\delta^{(m)} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)})$ to avoid duplicate computation.)

$$\begin{aligned}
\frac{\partial L}{\partial w_{ij}^{(m)}} &= \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial w_{ij}^{(m)}} \\
&= \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \cdot y_j^{(m-1)} \\
&= \delta^{(m)} \cdot y_j^{(m-1)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\frac{\partial L}{\partial b_i^{(m)}} &= \frac{\partial L}{\partial y_i^{(m)}} \cdot \frac{\partial y_i^{(m)}}{\partial x_i^{(m)}} \cdot \frac{\partial x_i^{(m)}}{\partial b_i^{(m)}} \\
&= \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \\
&= \delta^{(m)}
\end{aligned}$$

Summary

1. Compute $\frac{\partial L}{\partial y_i^{(M)}}$.
2. Use
$$\frac{\partial L}{\partial y_i^{(m)}} = \sum_k \frac{\partial L}{\partial y_k^{(m+1)}} \cdot \sigma'(x_k^{(m+1)}) \cdot w_{ki}^{(m+1)}$$
compute $\frac{\partial L}{\partial y_i^{(m)}}$ for $m = 1, 2..., M-1$.
3. Compute
$$\frac{\partial L}{\partial b_i^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) = \delta^{(m)}$$
for $m = 1, 2..., M$.
4. Compute
$$\frac{\partial L}{\partial w_{ij}^{(m)}} = \frac{\partial L}{\partial y_i^{(m)}} \cdot \sigma'(x_i^{(m)}) \cdot y_j^{(m-1)} = \delta^{(m)} \cdot y_j^{(m-1)}$$
for $m = 1, 2..., M$.

7.2.5 Other Methods

Stochastic Gradient Descent (SGD)

Subgradient Method

7.3 Perceptron Algorithm

Definition 10. *Binary linear classifiers distinguish between two categories through a linear function of the inputs.*

Definition 11. *Linearly separable refers to a line that can be drawn to perfectly split the two classes.*

The Perceptron algorithm is an efficient algorithm for learning a **linear separator** in $d-$dimensional space, with a mistake bound that depends on the margin of separation of the data.

7.3.1 General Idea

Given the training data

$$D = \left\{ \left\langle x^{(i)}, y^{(i)} \right\rangle, i = 1, ..., n \right\} \in (\mathbb{R}^m \times \{0, 1\})^n$$

we want to know the exact value of $y \in \{0, 1\}$.

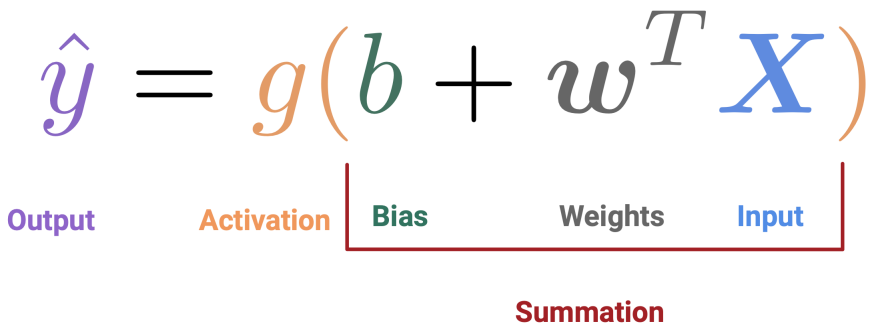


Figure 8: Perceptron Output

General idea:

- If the perceptron correctly predicts ($\hat{y} = y$):
  - Do nothing
- If the perceptron yields an incorrect prediction ($\hat{y} \neq y$):
  - If the prediction is 0 and truth is 1 ($\hat{y} = 0 | y = 1 \Rightarrow e = y - \hat{y} = 1$), add feature vector to weight vector.
  - If the prediction is 1 and truth is 0 ($\hat{y} = 1 | y = 0 \Rightarrow e = y - \hat{y} = -1$), subtract feature vector from the weight vector.

Since we want the prediction to be either 0 or 1, we usually use binary function as the activation function in perceptron.

7.3.2 Algorithm

Perceptron Algorithm:

- Initialize weights (including a bias term) to zero, e.g. $W = [w, b] = 0^{m+1}$.
- Under each training epoch: Compute for each sample $\langle x^{(i)}, y^{(i)} \rangle \in D$
  - A prediction $\hat{y}^{(i)} = g({x^{(i)}}^T W)$

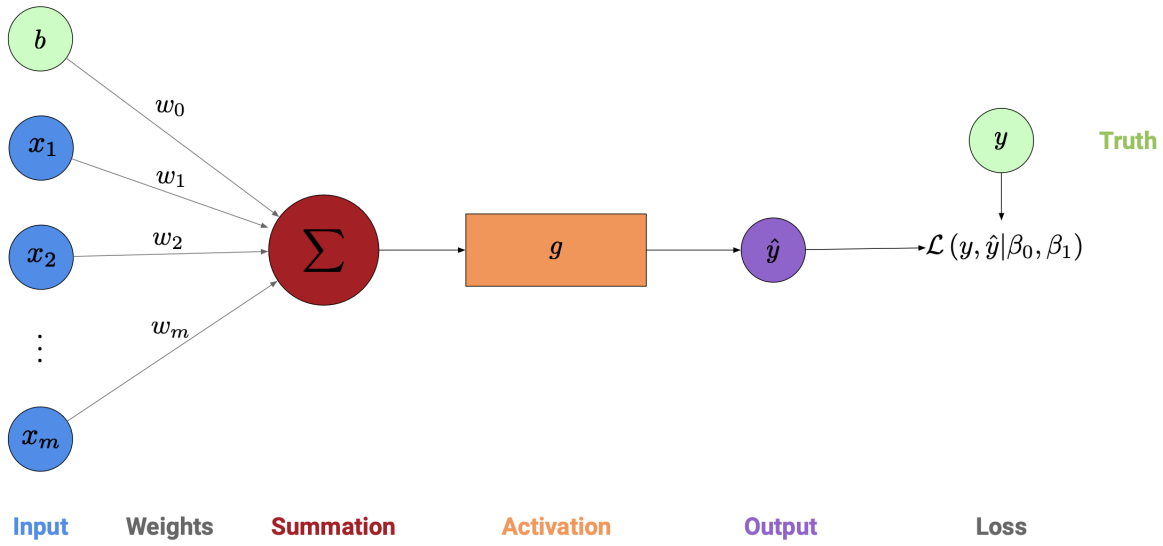


Figure 9: Perceptron

- Prediction error $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$
- Weighted update $W = W + \eta e^{(i)} x^{(i)}$

7.3.3 Limitations

(1) Only provides a linear classifier boundary.

(2) Only allows for **binary classifier** between two classes.

(3) **No convergence possible** if classes are not linearly separable.

(4) Perceptron will yield **multiple boundary/"optimal" solutions**.

(5) Boundaries found may **not** perform **equally well**.

7.4 ADAptive LInear NEuron (ADALINE)

7.4.1 General Idea

Except the activation function in perceptron, we can add a threshold function.
In perceptron, we generate the estimation $\hat{y}$ (after binary function) to help update weight $\{w_i\}_{i=0}^m$.
However, in ADALINE, we minimize MSE $z = x^T W$ to update weight $\{w_i\}_{i=0}^m$ before output estimation $\hat{y}$ (before binary function).

Before entering threshold (binary function), we want to minimize a mean- squared error (MSE) loss function to estimate weights.

e.g. suppose $g(x) = x$, let $z = x^T W$ be the input of threshold, for each y,

$$W^* = \underset{W}{\operatorname{argmin}} L(z, y) = (y - z)^2$$

$$\frac{\partial L(z, y)}{\partial w_i} = -2(y - z)\frac{\partial z}{\partial w_i} = -2(y - z)x_i$$

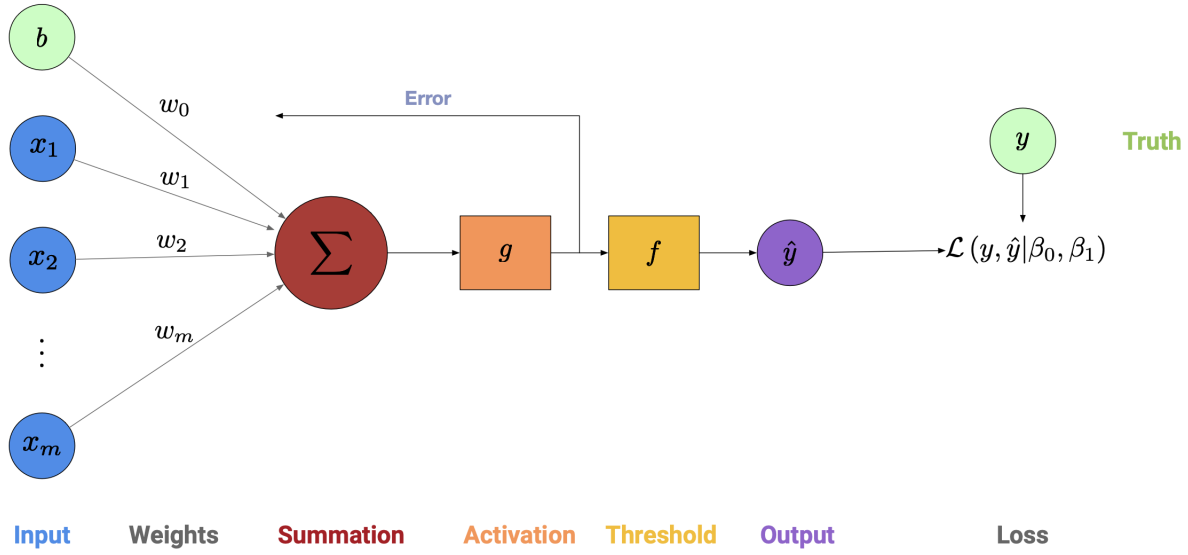


Figure 10: ADALINE

7.4.2 Widrow-Hoff Delta Rule

(Gradient Descent Rule for ADALINE)

- **Original:**

$$W = W + \eta(y^{(j)} - z)x^{(j)}$$

- **Unit-norm:**

$$W = W + \eta(y^{(j)} - z)\frac{x^{(j)}}{\|x^{(j)}\|}$$

where $\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2}$

The Perceptron and ADALINE use variants of the delta rule!

(1) **Perceptron**: Output used in delta rule is $\hat{y} = g(x^T W)$; $W = W + \eta(y^{(j)} - \hat{y}^{(j)})x^{(j)}$

(2) **ADALINE**: Output used to estimate weights is $z = x^T W$. $W = W + \eta(y^{(j)} - z)x^{(j)}$

7.5 Logistic Regression (Binary-class Output)

7.5.1 Generative and Discriminative Classifiers

The most important difference between naive Bayes and logistic regression is that logistic regression is a **discriminative classifier** while naive Bayes is a **generative classifier**.

Suppose we want to classify class A (dogs) and class B (cats) (More genearl form: assign a class $c \in C$ to a document $d \in D$):

(1) **Generative model:** A generative model would have the goal of understanding what dogs look like and what cats look like. You might literally ask such a model to 'generate', i.e., draw, a dog. e.g. **naive Bayes:** we do not directly compute the probability that the document d belongs to each class c, $P(c|d)$. We compute likelihood $P(d|c)$ and prior probability $P(c)$ to generate best estimation $\hat{c}$. (i.e., we want to know what should the distribution of a document d in class c be like.)

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(d|c)P(c)$$

(2) **Discriminative model:** A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them). That is we want to directly computing $P(c|d)$.

7.5.2 Sigmoid function

The goal of binary logistic regression is to train a classifier that can make a binary decision about the class of a new input observation.

The input observation is $x = [x_1, ..., x_m]^T$ and the output y is either 1 or 0. Instead of using the optimal weights of each feature x_i and binary activation function (**threshold**: $\hat{y} = 1$ if $z \geq 0$ and $\hat{y} = 0$ otherwise) to estimate in Perceptron and ADALINE, **we want to estimate the probability $P(y = 1|x)$.**

However, the weighted sum $z = x^T W = \sum_{i=1}^{m} w_i x_i + b$ ranges $-\infty$ to ∞. We want to **force the z to be a legal probability**, that is, to lie between 0 and 1.

The **sigmoid function** $\sigma(z) = \frac{1}{1+e^{-z}}$ can be used as acitivation for this purpose, $P(y = 1|x) = \sigma(x^T W)$. Since $1 - \sigma(x) = \sigma(-x)$, $P(y = 0|x) = \sigma(-x^T W)$.

7.5.3 Cross-entropy Loss Function

We choose the parameters W that maximize the log probability of the true y labels in the training data given the observations x. The conditional probability

$$p(y|x) = \begin{cases} \hat{y}, & y = 1 \\ 1 - \hat{y}, & y = 0 \end{cases} = \hat{y}^y (1 - \hat{y})^{1-y}$$

To maximize the probability, we log both sides:

$$\log p(y|x) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

Then, we want the $\hat{y}$ to maximize the probability (also the logarithm of the probability):

$$\begin{aligned} \hat{y}^* &= \underset{\hat{y} \in [0,1]}{\operatorname{argmax}} \log p(y|x) \\ &= \underset{\hat{y} \in [0,1]}{\operatorname{argmin}} - \log p(y|x) \\ &= \underset{\hat{y} \in [0,1]}{\operatorname{argmin}} -(y \log \hat{y} + (1 - y) \log(1 - \hat{y})) \end{aligned}$$

The right hand side is exactly the **cross-entropy loss function**:

$$L(y, \hat{y}) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

where $\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$

$$\frac{\partial L(y^{(i)}, \hat{y}^{(i)})}{\partial w_j} = \left(\sigma(w^T x^{(i)} + b) - y^{(i)}\right) x_j^{(i)} = (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$$

The risk (Binary Cross-Entropy Cost) of a weight W is

$$J(W) = -\frac{1}{n} \sum_{i=1}^{n} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$$

$$\frac{\partial J(w, b)}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\sigma(w^T x^{(i)} + b) - y^{(i)}\right) x_j^{(i)}$$

7.5.4 Algorithm

- Initialize weights (including a bias term) to zero, e.g. $W = [w, b] = 0^{m+1}$.
- Under each training epoch: Compute for each sample $\langle x^{(i)}, y^{(i)} \rangle \in D$
  - · A prediction $\hat{y}^{(i)} = g({x^{(i)}}^T W)$
  - · Prediction error $e^{(i)} = y^{(i)} - \hat{y}^{(i)}$
  - · Weighted update $W = W + \eta e^{(i)} x^{(i)} = W - \eta \nabla L(W)$

7.6 Softmax Regression (Multi-class Output)

7.6.1 Multi-Class Classification and Multi-Label Classification

Definition 12. *Multi-Class Classification is a process for assigning each sample exactly one class. In this case, classes are considered mutually exclusive (no intersection).*

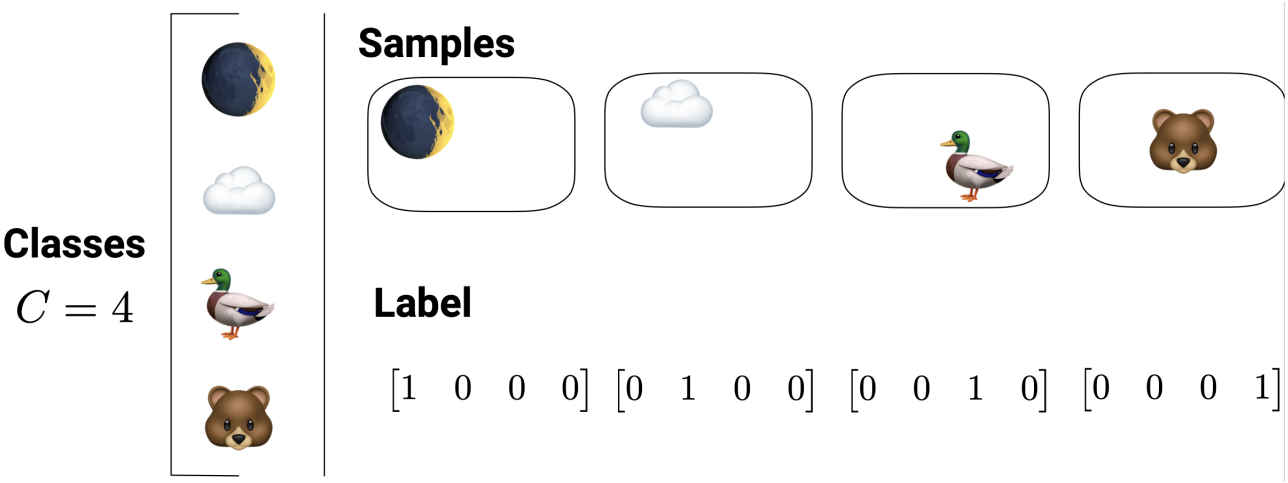


Figure 11: Multi-Class Classification

Definition 13. *Multi-Label Classification or annotation allows for each sample to have 1 or more classes assigned to it. In this case, classes are mutually non-exclusive (one common element).*

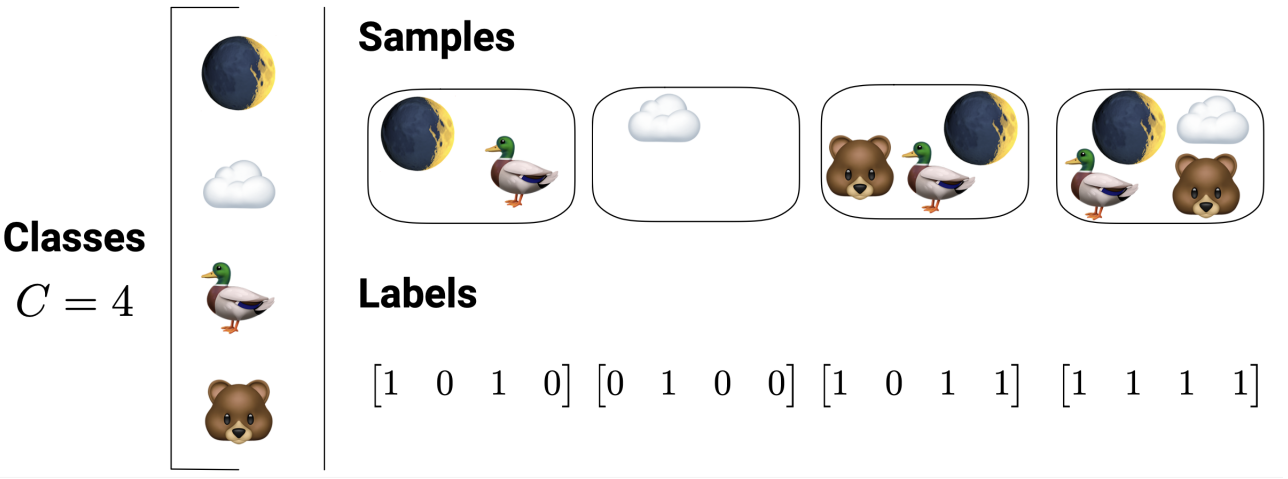


Figure 12: Multi-Label Classification

We can show some examples of activation layer and loss choice in different probelms.

	Output Activation	Loss Function
Binary Classification	Sigmoid	Binary Cross-Entropy
Multi-Class Classification	Softmax	Categorical Cross-Entropy
Multi-Label Classification	Sigmoid	Binary Cross-Entropy
Linear Regression	Identity	Mean Squared Error
Regression to values in [0, 1]	Sigmoid	Mean Squared Error or Binary Cross-Entropy

Figure 13: Examples of Activation Layer and Loss Choice

7.6.2 One-hot Encoding

Definition 14. *One-hot encoding is the process of assigning a single location within a vector to represent a given category.*

Strength
"High"
"Medium"
"Low"
...
"Med"

→

High	Medium	Low
1	0	0
0	1	0
0	0	1
...	...	...
0	1	0

Figure 14: One-hot encoding

Examples:

1. $\mathbf{z} = [4]_{1\times1} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}_{1\times5}$
2. $\mathbf{y} = \begin{bmatrix} 3 & 0 & 2 \end{bmatrix}_{1\times3} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}_{3\times4}$

3. $\mathbf{v} = \begin{bmatrix} 5 & 0 & 4 & 4 & 3 \end{bmatrix}_{1\times 5} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}_{5\times 6}$

Usefulness of Encodings

1. Close to a traditional design matrix for linear regression that uses a codified dummy variable structure. e.g. FALSE (0) or TRUE (1)
2. Reduce the size of data stored by using numbers instead of strings.
3. Poor if there are too many unique values (e.g. text messages on a phone.)

7.6.3 Softmax function

Definition 15. *Softmax function or normalized exponential function converts between real valued numbers to values between 0 and 1*

Softmax: $S_j(\vec{x}) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}$,

$$S(\vec{x}) = \left[\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, \cdots, \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right]^T$$

We can show the softmax function has following properties:

(1) First-derivative:

$$\frac{\partial S_j(\vec{x})}{\partial x_j} = S_j(\vec{x})[1 - S_j(\vec{x})]$$

(2) Stabilizing softmax:

$$S_j(\vec{x} + c) = \frac{e^{x_j + c}}{\sum_{i=1}^{n} e^{x_i + c}} = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}} = S_j(\vec{x})$$

7.6.4 Categorical Cross-entropy Loss Function

Definition 16. *Categorical Cross-entropy Loss is a way to quantify the difference between a "true" values $\{y_c\}_{c\in C}$ and an estimated $\{\hat{y}_c\}_{c\in C}$ across C categories.*

Note: y needs one-hot encoding firstly,

$$L(y, \hat{y}) = -\sum_{c\in C} (y_c \cdot \log(\hat{y}_c))$$

Definition 17. *Categorical Cross-entropy Cost is a way of quantifying the cost over multiple points from different categories.*

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{y}_i) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c\in C}(y_{i,c} \cdot \log(\hat{y}_{i,c}))$$

7.7 Deep Feedforward Networks

7.7.1 Definition

In any neural network, a dense layer is a layer that is deeply connected with its preceding layer which means the neurons of the layer are connected to *every neuron* of its preceding layer.

Definition 18. *Deep feedforward networks, feedforward neural networks, multilayer perceptrons (MLPs), or dense neural networks form the foundations of deep learning models. Learning occurs in only one direction:* ***forward****. There are* ***no feedback*** *connections in whichoutputs of the model are fed back into itself. There are no cycles or loops present. Information must flow from the input layer, through one or more hidden layers, before reaching the output.*

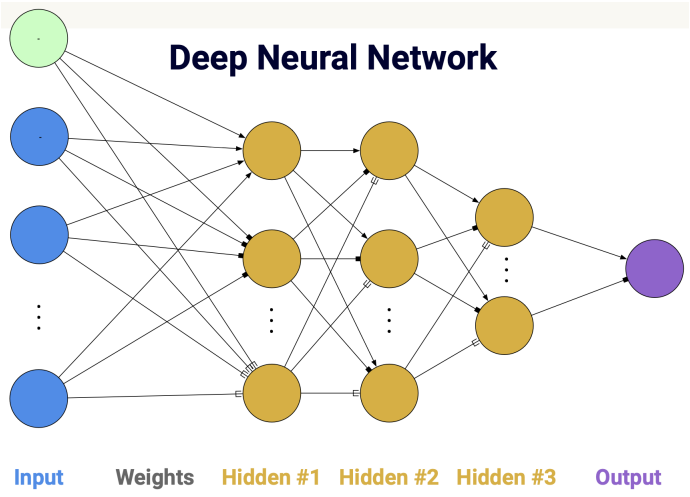


Figure 15: Deep Neural Network

A deep neural network contains *Input Layer*, *Hidden Layer*, and *Output Layer*.

7.7.2 Universal Approximation Theorem

Universal Approximation Theorem, in its lose form, states that a **feed-forward network** with a **single hidden layer** containing a finite number of neurons **can approximate any continuous function**. (Which is also equivalent to having a nonpolynomial activation function)

Theorem 2 (Universal approximation theorem)**.** *Fix a continuous function* $\sigma : \mathbb{R} \to \mathbb{R}$ *(activation function) and positive integers* $d, D \in \mathbb{Z}^+$*. The function* σ *is not a polynomial* $\Leftrightarrow$ *for every continuous function* $f : \mathbb{R}^d \to \mathbb{R}^D$ *(target function), every compact subset* K *of* $\mathbb{R}^d$*, and every* $\epsilon > 0$ *there exists a continuous function* $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ *(the layer output) with representation*

$$f_\epsilon = W_2 \circ \sigma \circ W_1$$

where W_2, W_1 *are composable affine maps and* o *denotes component-wise composition, such that the approximation bound*

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

holds for any ϵ *arbitrarily small (distance from* f *to* f_ϵ *can be infinitely small).*

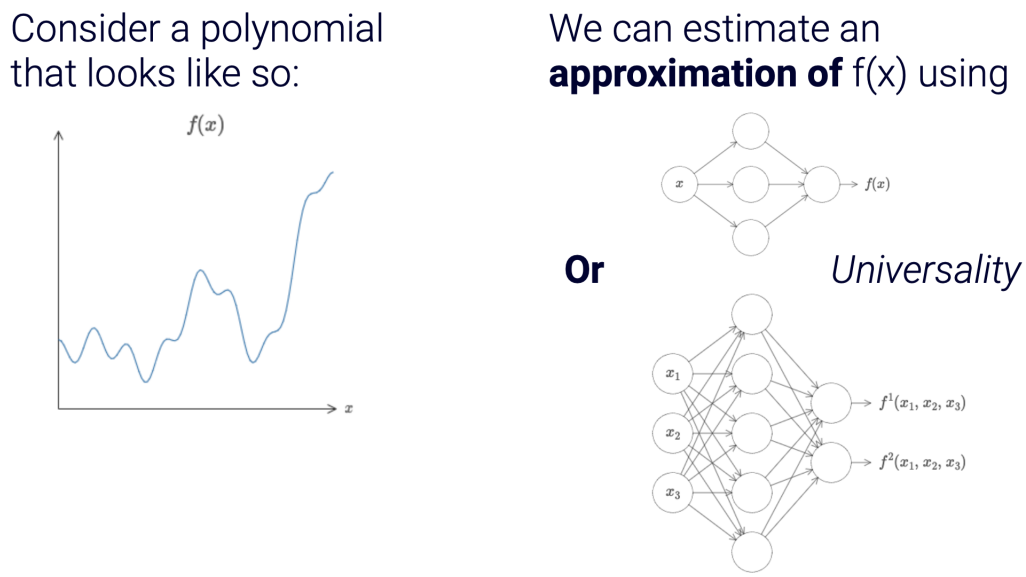


Figure 16: Universal Approximation Theorem

7.8 Mini-batch Optimization

7.8.1 Stochastic Gradient Descent (SGD) and Batch Gradient Descent (BGD)

Stochastic Gradient Descent (SGD)

1. Start with a random guess.
2. For n epochs:
   1) Reorder data
   2) Retrieve an observation $i = 1, 2, ...$ one by one in reordered data:
      (1) Compute gradient on single data point i: $\frac{\partial J_i(W)}{\partial W}$
      (2) Update parameters: $W = W - \alpha \frac{\partial J_i(W)}{\partial W}$
3. Output parameters

Note: "On-line"/"Stochastic" **Single** Observation Updates

Batch Gradient Descent (BGD)

1. Start with a random guess.
2. For n epochs:
   1) Compute gradients on **all the data**: $\frac{\partial J(W)}{\partial W}$
   2) Update parameters: $W = W - \alpha \frac{\partial J(W)}{\partial W}$
3. Output parameters

Note: All data used in update

7.8.2 Mini-Batch Gradient Descent (MBGD)

We want a middle ground between SGD and BGD.

Mini-Batch Gradient Descent (MBGD)

1. Start with a random guess.
2. For n epochs:

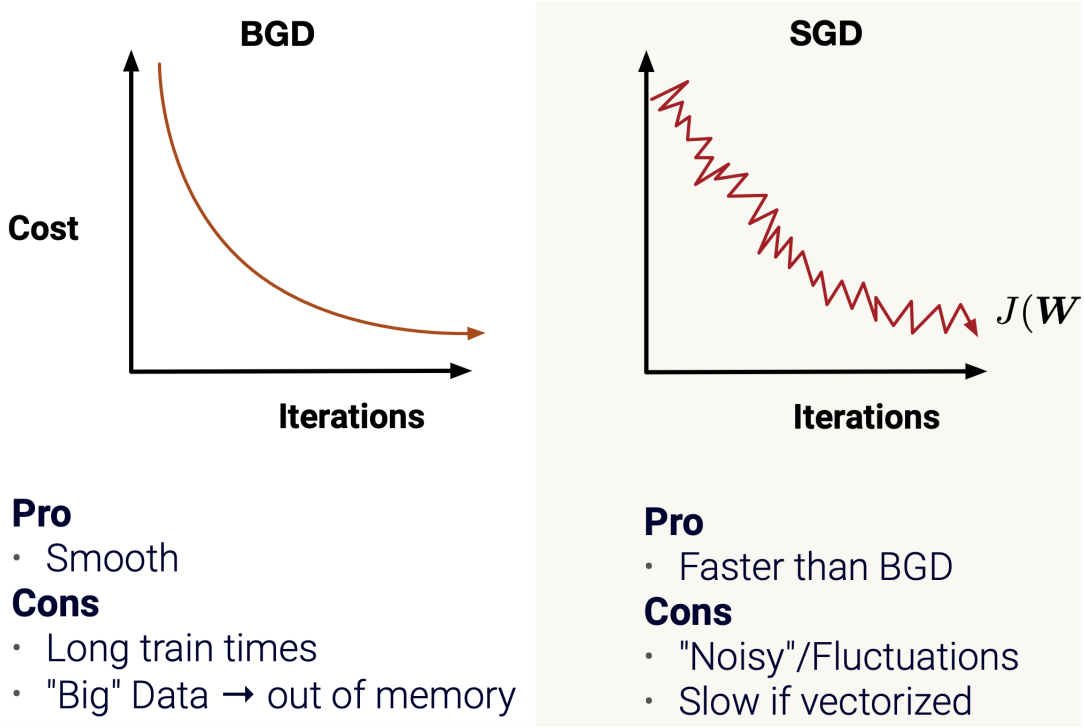


Figure 17: BGD and SGD

1) Reorder data and retrieve a subet of reordered data with size b (batch size)

2) Retrieve an observation $i = 1, 2, ...b$ one by one in the subet:

   (1) Compute gradient on single data point i: $\frac{\partial J_i(W)}{\partial W}$

   (2) Update parameters: $W = W - \alpha \frac{\partial J_i(W)}{\partial W}$

3. Output parameters

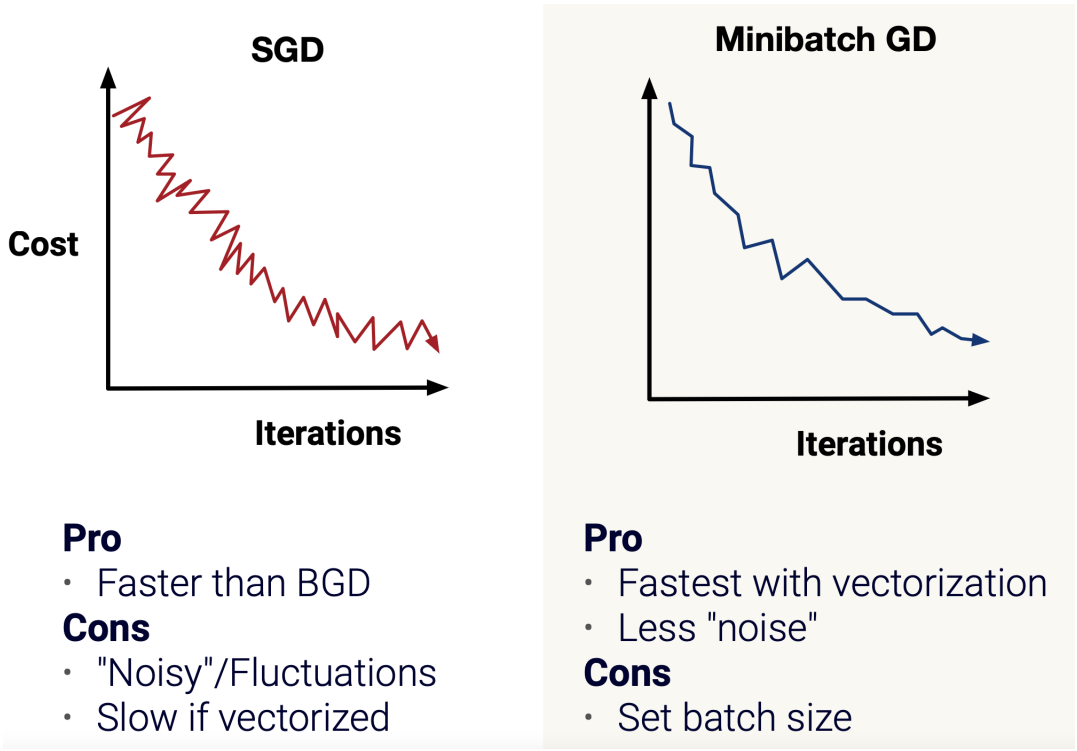


Figure 18: SGD and MBGD

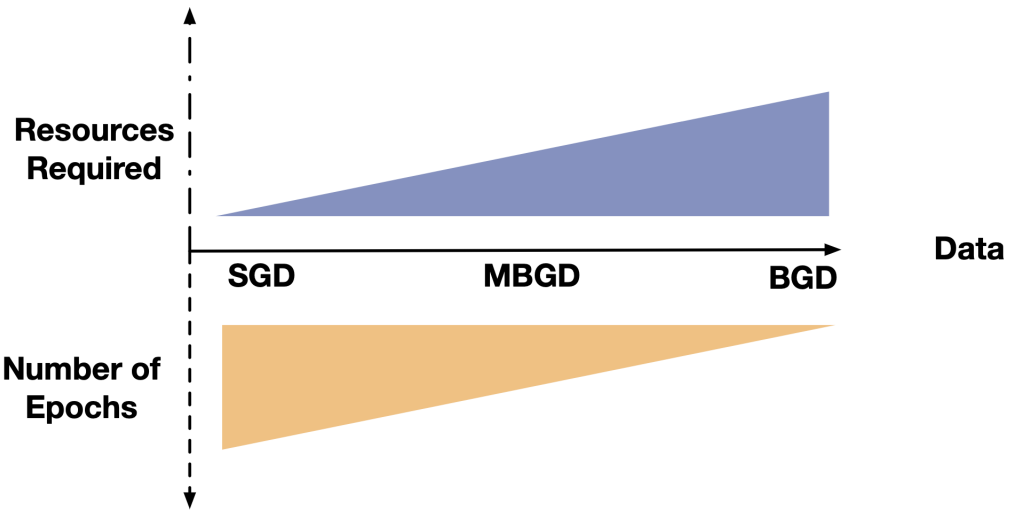


Figure 19: Comparison of Approaches