Mathematical Background

Vahid Tarokh
ECE685D
Fall 2025

Quick Review of Linear Algebra

Introduction

- In order to design and implement deep networks we need to know
  - Basics of Linear Algebra
  - Basics of Multivariable Calculus
  - Basics of Probability
- For writing research papers, you may need to know more.
- Here, we will include some of the background for completeness, but will not teach them all.
- Source: **Dive into Deep Learning**
  - **Professor Smola's Slides**
  - **Professor David Carlson's Slides**
  - **Professor Lawrence Caron's slides**
  - **Professor Ruslan Salakhutdinov's slides (available online)**

Scalars

- **Simple operations**

$$c = a + b$$
$$c = a \cdot b$$
$$c = \sin a$$

- **Length**

$$|a| = \begin{cases} a & \text{if} a > 0 \\ -a & \text{otherwise} \end{cases}$$
$$|a + b| \leq |a| + |b|$$
$$|a \cdot b| = |a| \cdot |b|$$

Vectors

- **Simple operations**

$$c = a + b \quad \text{where } c_i = a_i + b_i$$
$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$
$$c = \sin a \quad \text{where } c_i = \sin a_i$$

Definition of a vector space

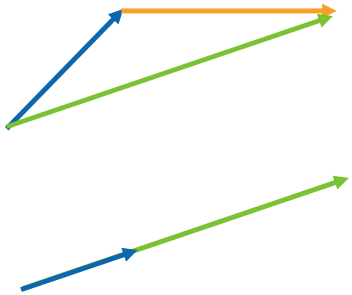
- **Length**

$$\|a\|_2 = \left[\sum_{i=1}^{m} a_i^2\right]^{\frac{1}{2}}$$

Definition of a norm

$$\|a\| \geq 0 \text{ for all } a$$
$$\|a + b\| \leq \|a\| + \|b\|$$
$$\|a \cdot b\| = |a| \cdot \|b\|$$

Vectors



$$c = a + b$$

$$c = \alpha \cdot b$$

Mathematician's `'parallel for all do'`

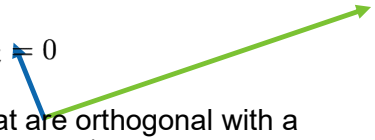
Vectors

- **Dot product**

$$a^\top b = \sum_i a_i b_i$$

- **Orthogonality**

$$a^\top b = \sum_i a_i b_i = 0$$



(e.g. if we have two vectors that are orthogonal with a third, their linear combination is it, too)

Matrices

- **Simple operations**

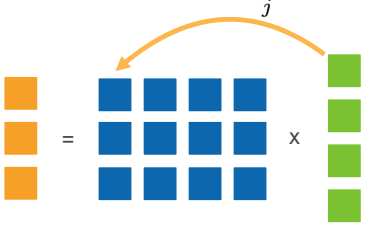
$$C = A + B \quad \text{where } C_{ij} = A_{ij} + B_{ij}$$
$$C = \alpha \cdot B \quad \text{where } C_{ij} = \alpha B_{ij}$$
$$C = \sin A \quad \text{where } C_{ij} = \sin A_{ij}$$

- **Functional Analysis 101**
  vector = function, matrix = linear operator
  most theorems work sort-of in infinite dimensional spaces

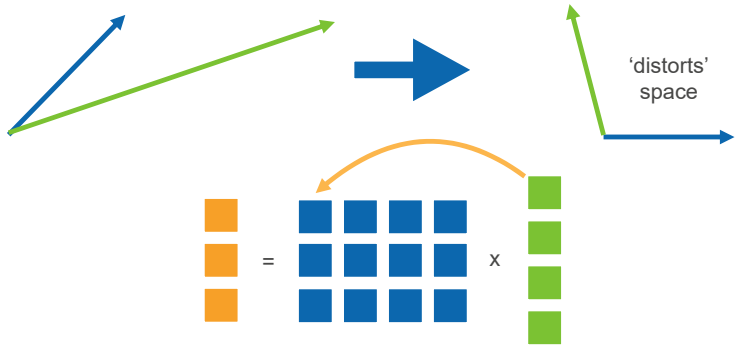
Matrices

- **Multiplications (matrix vector)**

$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$



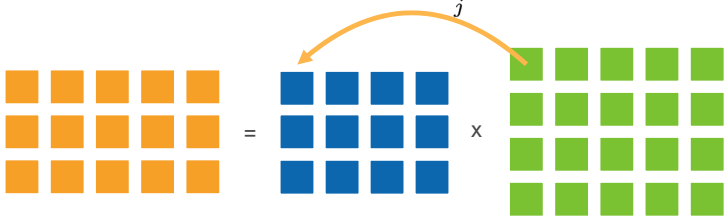
Matrices



Matrices

- **Multiplications (matrix matrix)**

$$C = AB \text{ where } C_{ik} = \sum_j A_{ij} B_{jk}$$



Matrices

- **Norms**

$$c = A \cdot b \text{ hence } \|c\| \leq \|A\| \cdot \|b\|$$

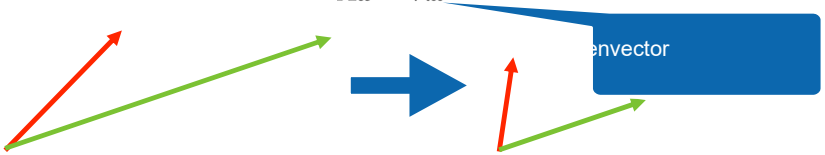
  - Choices depending on how to measure length of b and c
- **A Popular norm**
  - Frobenius norm

$$\|A\|_{\text{Frob}} = \left[\sum_{ij} A_{ij}^2 \right]^{\frac{1}{2}}$$

Matrices

- **Eigenvectors and eigenvalue**
  - Vectors that aren't changed by the matrix

$$Ax = \lambda x$$



  - For symmetric matrices we can always find this

Special Matrices

- **Symmetric, antisymmetric** $A_{ij} = A_{ji}$ and $A_{ij} = -A_{ji}$

- **Non-negative definite**

$$\|x\|^2 = x^\top x \geq 0 \text{ generalizes to } x^\top A x \geq 0$$

(all non-negative eigenvalues)

Special Matrices

- **Orthogonal Matrices**
  - All rows of the matrix are orthogonal to each other
  - All rows of the matrix have unit length

$$U \text{ with } \sum_j U_{ij} U_{kj} = \delta_{ik}$$

Show that $U^\top U = \mathbf{1}$

  - Rewrite in matrix form

$$UU^\top = \mathbf{1}$$

- **Permutation Matrices**

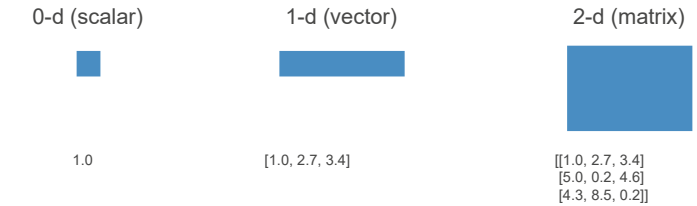
$$P \text{ where } P_{ij} = 1 \text{ if and only if } j = \pi(i)$$

Show that P is orthogonal

Multidimensional Arrays

N-dimensional Array Examples

N-dimensional array, short for ndarray, is the main data structure for machine learning and neural networks

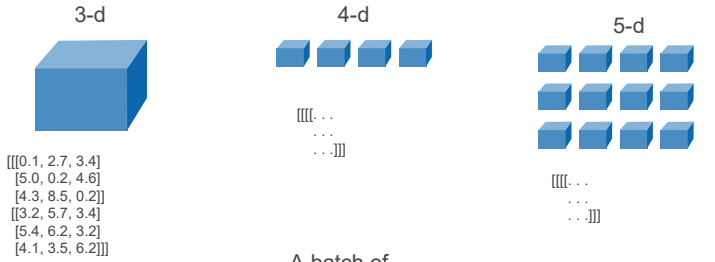


A class label

A feature vector

A example-by-feature matrix

ND Array Examples, cont

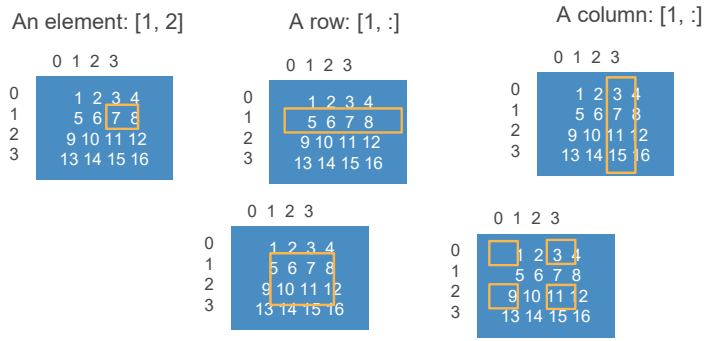


A RGB image (width x height x channels)

A batch of RGB images (batch-size x width x height x channels)

A batch of videos (batch-size x time x width x height x channels)

Access Elements



Review of Basic Probability

Probability

Space of events X

- server working; slow response; server broken
- income of the user (e.g. $95,000)
- query text for search (e.g. “statistics tutorial”)

Probability axioms

$$\Pr(X) \in [0,1],\ \Pr(\mathcal{X}) = 1$$
$$\Pr(\cup_i X_i) = \textstyle\sum_i \Pr(X_i) \text{ if } X_i \cap X_j = \emptyset$$

Example queries

- P(server working) = 0.999 — discrete
- P(90,000 < income < 100,000) = 0.1 — continuous

What you must know

- Definitions of random variable and random vector
- Conditional probability, Independence, and dependence
- Law of Total Probability
- Definition of PMF, PDF, and CDF
- Generation of an arbitrary random variable with a given pdf from the uniform random variable
- PMF and PDF of transforms of random vectors
- Mathematical Expectation
- Variance, Covariance, correlation, etc.
- Multivariable Calculus and Lagrange’s multiplier method

(In)dependence

Independence

- Login behavior of two users (approximately)
- Disk crash in different colos (approximately)

$$\Pr(x, y) = \Pr(x) \cdot \Pr(y)$$

independent events

- Emails
- Queries
- News stream / Buzz / Tweets
- IM communication
- Russian Roulette

Everywhere

$$\Pr(x, y) \neq \Pr(x) \cdot \Pr(y)$$

Weak Low of Large Numbers

Let $X_1, X_2, \cdots X_n$ be i.i.d. random variable with mean μ. Then for any $\epsilon > 0$

$$P[\ |(X_1 + X_2 + \cdots + X_n)/n \ - \mu| > \epsilon\ ] \to 0$$

as $n \to \infty$.

Order Statistics

For $X_1, X_2, \ldots, X_n$ iid random variables X_k is the kth smallest X, usually called the kth order statistic.

$X_{(1)}$ is therefore the smallest X and

$$X_{(1)} = \min(X_1, \ldots, X_n)$$

Similarly, $X_{(n)}$ is the largest X and

$$X_{(n)} = \max(X_1, \ldots, X_n)$$

Order Statistics (density of maximum)

For $X_1, X_2, \ldots, X_n$ iid continuous random variables with pdf f and cdf F the density of the maximum is

$$\begin{aligned}
P(X_{(n)} \in [x, x+\epsilon]) &= P(\text{one of the } X\text{'s} \in [x, x+\epsilon] \text{ and all others} < x) \\
&= \sum_{i=1}^{n} P(X_i \in [x, x+\epsilon] \text{ and all others} < x) \\
&= nP(X_1 \in [x, x+\epsilon] \text{ and all others} < x) \\
&= nP(X_1 \in [x, x+\epsilon])P(\text{all others} < x) \\
&= nP(X_1 \in [x, x+\epsilon])P(X_2 < x) \cdots P(X_n < x) \\
&= nf(x)\epsilon F(x)^{n-1}
\end{aligned}$$

$$f_{(n)}(x) = nf(x)F(x)^{n-1}$$

Order Statistics (density of minimum)

For $X_1, X_2, \ldots, X_n$ iid continuous random variables with pdf f and cdf F the density of the minimum is

$$\begin{aligned}
P(X_{(1)} \in [x, x+\epsilon]) &= P(\text{one of the } X\text{'s} \in [x, x+\epsilon] \text{ and all others} > x) \\
&= \sum_{i=1}^{n} P(X_i \in [x, x+\epsilon] \text{ and all others} > x) \\
&= nP(X_1 \in [x, x+\epsilon] \text{ and all others} > x) \\
&= nP(X_1 \in [x, x+\epsilon])P(\text{all others} > x) \\
&= nP(X_1 \in [x, x+\epsilon])P(X_2 > x) \cdots P(X_n > x) \\
&= nf(x)\epsilon(1 - F(x))^{n-1}
\end{aligned}$$

$$f_{(1)}(x) = nf(x)(1 - F(x))^{n-1}$$

Order Statistics (density of k-th)

For $X_1, X_2, \ldots, X_n$ iid continuous random variables with pdf f and cdf F the density of the kth order statistic is

$$\begin{aligned}
P(X_{(k)} \in [x, x+\epsilon]) &= P(\text{one of the } X\text{'s} \in [x, x+\epsilon] \text{ and exactly } k-1 \text{ of the others} < x) \\
&= \sum_{i=1}^{n} P(X_i \in [x, x+\epsilon] \text{ and exactly } k-1 \text{ of the others} < x) \\
&= nP(X_1 \in [x, x+\epsilon] \text{ and exactly } k-1 \text{ of the others} < x) \\
&= nP(X_1 \in [x, x+\epsilon])P(\text{exactly } k-1 \text{ of the others} < x) \\
&= nP(X_1 \in [x, x+\epsilon]) \left(\binom{n-1}{k-1} P(X < x)^{k-1} P(X > x)^{n-k} \right)
\end{aligned}$$

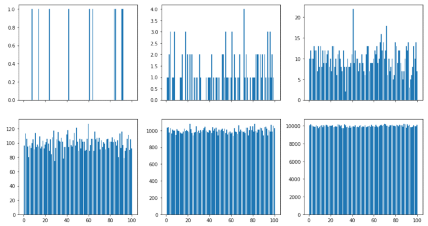
$$f_{(k)}(x) = nf(x) \binom{n-1}{k-1} F(x)^{k-1}(1 - F(x))^{n-k}$$

Uniform Distribution

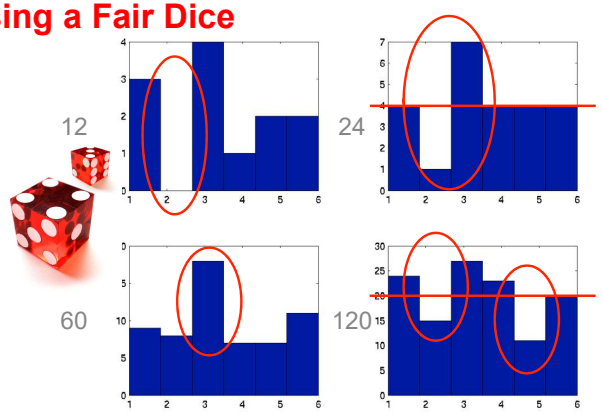
- Constant within an interval, zero outside

$$p(x) = \frac{1}{U - L} \text{ if } L \leq x \leq U$$

- Useful for initializing parameters or for load distribution



Tossing a Fair Dice



Euler's Gamma Function

For $x > 0$ The Euler's gamma function is defined as:

$$\Gamma(x) \equiv \int_0^\infty u^{x-1} e^{-u} du.$$

- The value $\Gamma(n) = (n-1)!$ When n > 0 is a positive integer.

- Show that $\Gamma(x) = (x-1)\Gamma(x-1)$ for $x > 1$.
- Calculate $\Gamma(3/2)$
- Calculate $\Gamma(1/2)$

Beta Distribution

• We define a distribution for a parameter $\mu \in [0, 1]$

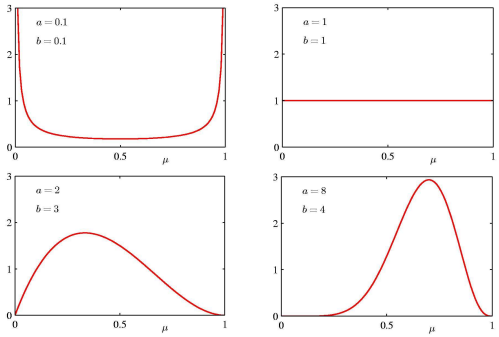
$$\begin{aligned} \text{Beta}(\mu|a,b) &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \mu^{a-1}(1-\mu)^{b-1} \\ \mathbb{E}[\mu] &= \frac{a}{a+b} \\ \text{var}[\mu] &= \frac{ab}{(a+b)^2(a+b+1)} \end{aligned}$$

where the Euler's gamma function is defined as:

$$\Gamma(x) \equiv \int_0^\infty u^{x-1} e^{-u} du.$$

and ensures that the Beta distribution is normalized.

Beta Distribution



Relationship Between Beta and Uniform

Let $X_1, X_2, \ldots, X_n \overset{iid}{\sim} \text{Unif}(0,1)$ then the density of $X_{(n)}$ is given by

$$f_{(k)}(x) = nf(x)\binom{n-1}{k-1}F(x)^{k-1}(1-F(x))^{n-k}$$
$$= \begin{cases} n\binom{n-1}{k-1}x^{k-1}(1-x)^{n-k} & \text{if } 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

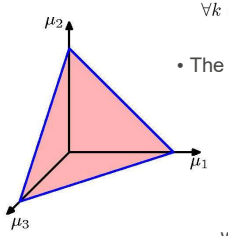
This is an example of the Beta distribution where $r = k$ and $s = n - k + 1$.

$$X_{(k)} \sim \text{Beta}(k, n-k+1)$$

Dirichlet Distribution

• Consider a distribution over the K-dimensional simplex, subject to constraints:

$$\forall k : \mu_k \geqslant 0 \quad \text{and} \quad \sum_{k=1}^{K} \mu_k = 1$$



• The Dirichlet distribution is defined as:

$$\text{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)}\prod_{k=1}^{K}\mu_k^{\alpha_k-1}$$

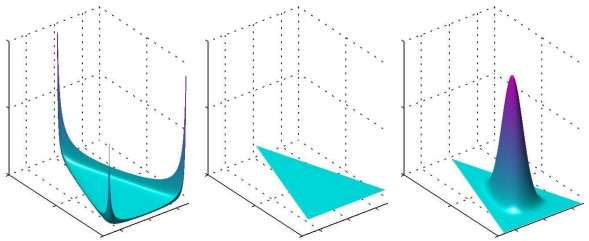
$$\alpha_0 = \sum_{k=1}^{K}\alpha_k$$

where $\alpha_1, \ldots, \alpha_k$ are the parameters of the distribution, and Γ(x) is the gamma function.

• The Dirichlet distribution is confined to a simplex as a consequence of the constraints.

Dirichlet Distribution

• Plots of the Dirichlet distribution over three variables.



$\alpha_k = 10^{-1}$ $\alpha_k = 10^{0}$ $\alpha_k = 10^{1}$

Generation of Dirichlet from Beta

- Consider a Stick of length 1.

Simulate a random variate $X_j \sim Beta(\alpha_j, \sum_{i=j+1}^{k} \alpha_i)$, where $j = 1, \ldots, k-1$. When $j = 1$, we have $X_1 \sim Beta(\alpha_1, \sum_{i=2}^{k} \alpha_i)$. The first piece of the stick has length $1 \cdot X_1$, such that the length of the remaining stick is $1 - X_1$. Also, set $Y_1 = X_1$.

Generation of Dirichlet from Beta

When $j = 2$, we have $X_2 \sim Beta(\alpha_2, \sum_{i=3}^{k} \alpha_i)$. The second piece of the stick has length $(1-X_1)X_2$, such that the length of the remaining stick is $(1-X_1)-(1-X_1)X_2 = (1-X_1)(1-X_2)$. Also, set $Y_2 = (1-X_1)X_2$.

$\vdots$

Continue in this way.

Generation of Dirichlet from Beta

When $j = k-1$, we have $X_{k-1} \sim Beta(\alpha_{k-1}, \alpha_k)$. The $(k-1)^{th}$ piece of the stick has length $X_{k-1} \prod_{j=1}^{k-2}(1-X_j)$, such that the length of the remaining stick is $\prod_{j=1}^{k-1}(1-X_j)$. Also, set $Y_{k-1} = X_{k-1} \prod_{j=1}^{k-2}(1-X_j)$. Note that the k^{th} piece of the stick has length $\prod_{j=1}^{k-1}(1-X_j)$ and set $Y_k = \prod_{j=1}^{k-1}(1-X_j)$. We can conclude that $(Y_1, \ldots, Y_k) \sim Dir(\alpha_1, \ldots, \alpha_k)$.

Source: Bela A Frigyik, Amol Kapila, and Maya R Gupta. Introduction to the Dirichlet distribution and related processes. Technical report, UWEETR-2010-0006, 2010

Entropy

The entropy of a d-dimensional random vector $\mathbf{X} := \begin{bmatrix} X_1 & \cdots & X_d \end{bmatrix}^T$ is defined by the expectation of the self information

$$H(\mathbf{X}) := \mathbb{E}_{\mathbf{X}}\left[\log \tfrac{1}{p(\mathbf{X})}\right] = \sum_{\mathbf{x} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_d} p(\mathbf{x}) \log \tfrac{1}{p(\mathbf{x})} = H(X_1, \ldots, X_d)\,.$$

The conditional entropy of X given Y is defined by

$$H(X|Y) := \sum_{y \in \mathcal{Y}} p(y) H(X|Y=y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x,y) \log \tfrac{1}{p_{X|Y}(x|y)}.$$

Kullback-Leibler Divergence

Let $p(\cdot)$ and $q(\cdot)$ are two p.m.f.'s of a random variable X. The relative entropy between p and q is $D(p||q) := \mathbb{E}_p\left[\log \frac{p(X)}{q(X)}\right]$

(the subscript "p" denotes that the expectation is taken over the distribution p.)

Please note that KL divergence is NOT symmetric: D (p || q) ≠ D (q || p).

Important Results:

$D(p||q) \geq 0$, *with equality iff* $p(x) = q(x)$ *for all* $x \in \mathcal{X}$.

Maximum Likelihood Estimator (MLE)

- Let $p_*(\cdot)$ be the true data generating distribution
- $E_*(\cdot)$ be expectation w.r.t. $p_*(\cdot)$
  - Suppose that iid samples (observations}

$$y_1, y_2, \cdots, y_n$$

are given.

  - Let $p \equiv p_\theta$ for $\theta \in \Theta$ denote our guesses for $p_*(\cdot)$
  - MLE: Choose the value of $\theta \in \Theta$ that achieves the maximum of

$$\frac{\Sigma_1^n \log p(y_i)}{n}$$

Maximum Likelihood Estimator (MLE)

- Why is it so popular?
  - It is an elementary function of the probability density function
  - Intimate relation with KL-divergence
  - Notice that

$$-\frac{\Sigma_1^n \log p(y_i)}{n} \rightarrow E_{p^*}[-\log\ p(y)]$$

  - Minimizing

$$-\frac{\Sigma_1^n \log p(y_i)}{n}$$

is asymptotically equivalent to minimizing

$$E_*\{-\log p(y)\} = D_{KL}(p_*||p) + \mathrm{H}(p_*)$$

or equivalently

$$D_{KL}(p_*||p).$$

Bernoulli Distribution

- Consider a single binary random variable $x \in \{0, 1\}$.
- For example, x can describe the outcome of flipping a coin:

Coin flipping: heads = 1, tails = 0.

- The probability of x=1 will be denoted by the parameter ¹, so that:

$$p(x = 1|\mu) = \mu \qquad 0 \leq \mu \leq 1.$$

- The probability distribution, known as Bernoulli distribution, can be written as:

$$\begin{aligned} \mathrm{Bern}(x|\mu) &= \mu^x(1-\mu)^{1-x} \\ \mathbb{E}[x] &= \mu \\ \mathrm{var}[x] &= \mu(1-\mu) \end{aligned}$$

Parameter Estimation

• Suppose we observed a data $\mathcal{D} = \{x_1, ..., x_N\}$

• We can construct the likelihood function, which is a function of μ.

$$p(\mathcal{D}|\mu) = \prod_{n=1}^{N} p(x_n|\mu) = \prod_{n=1}^{N} \mu^{x_n}(1-\mu)^{1-x_n}$$

• Equivalently, we can maximize the log of the likelihood function:

$$\ln p(\mathcal{D}|\mu) = \sum_{n=1}^{N} \ln p(x_n|\mu) = \sum_{n=1}^{N} \{x_n \ln \mu + (1-x_n)\ln(1-\mu)\}$$

• Note that the likelihood function depends on the N observations x_n only through the sum

$$\sum_n x_n$$

Sufficient Statistic

Parameter Estimation

• Suppose we observed a data $\mathcal{D} = \{x_1, ..., x_N\}$

$$\ln p(\mathcal{D}|\mu) = \sum_{n=1}^{N} \ln p(x_n|\mu) = \sum_{n=1}^{N} \{x_n \ln \mu + (1-x_n)\ln(1-\mu)\}$$

• Setting the derivative of the log-likelihood function w.r.t μ to zero, we obtain:

$$\mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} x_n = \frac{m}{N}$$

where m is the number of heads.

Binomial Distribution

• We can also work out the distribution of the number m of observations of x=1 (e.g. the number of heads).

• The probability of observing m heads given N coin flips and a parameter μ is given by:

$$p(m \text{ heads}|N,\mu) =$$

$$\mathrm{Bin}(m|N,\mu) = \binom{N}{m}\mu^m(1-\mu)^{N-m}$$

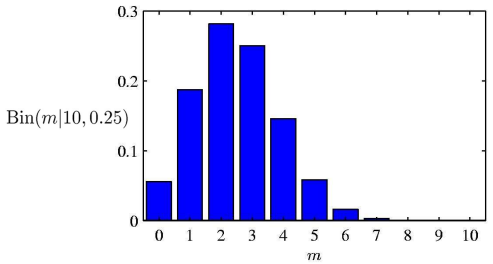
• The mean and variance can be easily derived as:

$$\mathbb{E}[m] \equiv \sum_{m=0}^{N} m\mathrm{Bin}(m|N,\mu) = N\mu$$

$$\mathrm{var}[m] \equiv \sum_{m=0}^{N} (m-\mathbb{E}[m])^2 \mathrm{Bin}(m|N,\mu) = N\mu(1-\mu)$$

Example

• Histogram plot of the Binomial distribution as a function of m for N=10 and μ = 0.25.



Multinomial Variables

• Consider a random variable that can take on one of K possible mutually exclusive states (e.g. roll of a dice).

• We will use so-called 1-of-K encoding scheme.

• If a random variable can take on K=6 states, and a particular observation of the variable corresponds to the state x_3=1, then **x** will be resented as:

1-of-K coding scheme: $\mathbf{x} = (0, 0, 1, 0, 0, 0)^{\mathrm{T}}$

• If we denote the probability of x_k=1 by the parameter μ_k, then the distribution over **x** is defined as:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k} \qquad \forall k : \mu_k \geqslant 0 \quad \text{and} \quad \sum_{k=1}^{K} \mu_k = 1$$

Multinomial Variables

• Multinomial distribution can be viewed as a generalization of Bernoulli distribution to more than two outcomes.

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k}$$

• It is easy to see that the distribution is normalized:

$$\sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\mu}) = \sum_{k=1}^{K} \mu_k = 1$$

and

$$\mathbb{E}[\mathbf{x}|\boldsymbol{\mu}] = \sum_{\mathbf{x}} p(\mathbf{x}|\boldsymbol{\mu})\mathbf{x} = (\mu_1, \ldots, \mu_K)^{\mathrm{T}} = \boldsymbol{\mu}$$

Maximum Likelihood Estimation

• Suppose we observed a data $\mathcal{D} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$

• We can construct the likelihood function, which is a function of ¹.

$$p(\mathcal{D}|\boldsymbol{\mu}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mu_k^{x_{nk}} = \prod_{k=1}^{K} \mu_k^{\left(\sum_n x_{nk}\right)} = \prod_{k=1}^{K} \mu_k^{m_k}$$

• Note that the likelihood function depends on the N data points only though the following K quantities:

$$m_k = \sum_n x_{nk}, \quad k = 1, ..., K.$$

which represents the number of observations of x_k=1.

• These are called the sufficient statistics for this distribution.

Maximum Likelihood Estimation

$$p(\mathcal{D}|\boldsymbol{\mu}) = \prod_{n=1}^{N} \prod_{k=1}^{K} \mu_k^{x_{nk}} = \prod_{k=1}^{K} \mu_k^{\left(\sum_n x_{nk}\right)} = \prod_{k=1}^{K} \mu_k^{m_k}$$

• To find a maximum likelihood solution for ¹, we need to maximize the log-likelihood taking into account the constraint that $\sum_k \mu_k = 1$

• Forming the Lagrangian:

$$\sum_{k=1}^{K} m_k \ln \mu_k + \lambda \left(\sum_{k=1}^{K} \mu_k - 1 \right)$$

$$\mu_k = -m_k/\lambda \qquad \mu_k^{\mathrm{ML}} = \frac{m_k}{N} \qquad \lambda = -N$$

which is the fraction of observations for which x_k=1.

Multinomial Distribution

• We can construct the joint distribution of the quantities $\{m_1, m_2, \ldots, m_k\}$ given the parameters [1] and the total number N of observations:

$$\begin{aligned} \mathrm{Mult}(m_1, m_2, \ldots, m_K|\boldsymbol{\mu}, N) &= \binom{N}{m_1 m_2 \ldots m_K} \prod_{k=1}^{K} \mu_k^{m_k} \\ \mathbb{E}[m_k] &= N\mu_k \\ \mathrm{var}[m_k] &= N\mu_k(1-\mu_k) \\ \mathrm{cov}[m_j m_k] &= -N\mu_j\mu_k \end{aligned}$$

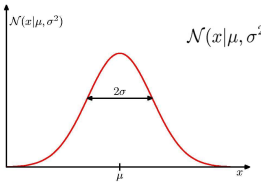
• The normalization coefficient is the number of ways of partitioning N objects into K groups of size $m_1, m_2, \ldots, m_K$.

• Note that

$$\sum_k m_k = N.$$

Gaussian Univariate Distribution

• In the case of a single variable x, the Gaussian distribution takes form:



$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

which is governed by two parameters:

- μ (mean)
- σ^2 (variance)

• The Gaussian distribution satisfies:

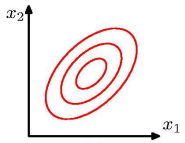
$$\mathcal{N}(x|\mu,\sigma^2) > 0$$

$$\int_{-\infty}^{\infty} \mathcal{N}\left(x|\mu,\sigma^2\right)\,\mathrm{d}x = 1$$

Multivariate Gaussian Distribution

• For a D-dimensional vector **x**, the Gaussian distribution takes form:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$



which is governed by two parameters:

- μ is a D-dimensional mean vector.
- Σ is a D by D covariance matrix.

and $|\Sigma|$ denotes the determinant of Σ.

• Note that the covariance matrix is a symmetric positive definite matrix.

Central Limit Theorem

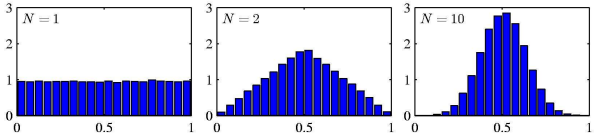
• The distribution of the sum of N i.i.d. random variables becomes increasingly Gaussian as N grows.

• Consider N variables, each of which has a uniform distribution over the interval [0,1].

• Let us look at the distribution over the mean:

$$\frac{x_1 + x_2 + \ldots. + x_N}{N}.$$

• As N increases, the distribution tends towards a Gaussian distribution.



Moments of the Gaussian Distribution

• The expectation of **x** under the Gaussian distribution:

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\int \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}\mathbf{x}\,\mathrm{d}\mathbf{x} \\ &= \frac{1}{(2\pi)^{D/2}}\frac{1}{|\boldsymbol{\Sigma}|^{1/2}}\underbrace{\int \exp\left\{-\frac{1}{2}\mathbf{z}^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}\mathbf{z}\right\}(\mathbf{z}+\boldsymbol{\mu})\,\mathrm{d}\mathbf{z}}\end{aligned}$$

The term in z in the factor (z^{+1}) will vanish by symmetry.

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}$$

Moments of the Gaussian Distribution

• The second order moments of the Gaussian distribution:

$$\mathbb{E}[\mathbf{x}\mathbf{x}^{\mathrm{T}}] = \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}} + \boldsymbol{\Sigma}$$

• The covariance is given by:

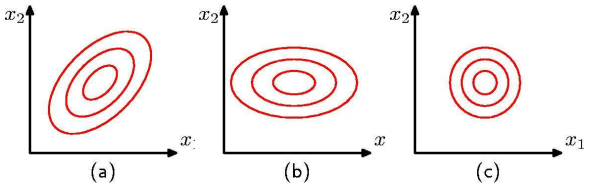
$$\mathrm{cov}[\mathbf{x}] = \mathbb{E}\left[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^{\mathrm{T}}\right] = \boldsymbol{\Sigma}$$

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu}$$

• Because the parameter matrix § governs the covariance of x under the Gaussian distribution, it is called the covariance matrix.

Moments of the Gaussian Distribution

• Contours of constant probability density:



Covariance matrix is of general form.

Diagonal, axis-aligned covariance matrix.

Spherical (proportional to identity) covariance matrix.

Partitioned Gaussian Distribution

• Consider a D-dimensional Gaussian distribution: $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$

• Let us partition **x** into two disjoint subsets x_a and x_b:

$$\mathbf{x} = \begin{pmatrix}\mathbf{x}_a \\ \mathbf{x}_b\end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix}\boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b\end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix}\boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb}\end{pmatrix}$$

• In many situations, it will be more convenient to work with the precision matrix (inverse of the covariance matrix):

$$\boldsymbol{\Lambda} \equiv \boldsymbol{\Sigma}^{-1} \qquad \boldsymbol{\Lambda} = \begin{pmatrix}\boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb}\end{pmatrix}$$

• Note that Λ_{aa} is not given by the inverse of Σ_{aa}.

Marginal Distribution

• It turns out that the marginal distribution is also a Gaussian distribution:

$$\begin{aligned} p(\mathbf{x}_a) &= \int p(\mathbf{x}_a, \mathbf{x}_b)\, \mathrm{d}\mathbf{x}_b \\ &= \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_a, \boldsymbol{\Sigma}_{aa}) \end{aligned}$$

• For a marginal distribution, the mean and covariance are most simply expressed in terms of partitioned covariance matrix.

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}$$

Conditional Distribution

• It turns out that the conditional distribution is also a Gaussian distribution:

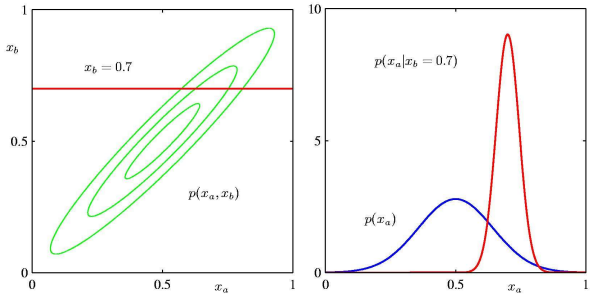
$$p(\mathbf{x}_a | \mathbf{x}_b) = \mathcal{N}(\mathbf{x}_a | \boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b})$$

Covariance does not depend on x_b.

$$\begin{aligned} \boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Lambda}_{aa}^{-1} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba} \\ \boldsymbol{\mu}_{a|b} &= \boldsymbol{\Sigma}_{a|b}\left\{\boldsymbol{\Lambda}_{aa}\boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b)\right\} \\ &= \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\mathbf{x}_b - \boldsymbol{\mu}_b) \\ &= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b) \end{aligned}$$

Linear function of x_b.

Conditional and Marginal Distributions



Maximum Likelihood Estimation

• To find a maximum likelihood estimate of the mean, we set the derivative of the log-likelihood function to zero:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \ln p(\mathbf{X} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}) = 0$$

and solve to obtain:

$$\boldsymbol{\mu}_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n.$$

• Similarly, we can find the ML estimate of Σ :

$$\boldsymbol{\Sigma}_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})(\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})^{\mathrm{T}}.$$

Maximum Likelihood Estimation

• Evaluating the expectation of the ML estimates under the true distribution, we obtain:

Unbiased estimate

$$\mathbb{E}[\boldsymbol{\mu}_{\mathrm{ML}}] = \boldsymbol{\mu}$$

$$\mathbb{E}[\boldsymbol{\Sigma}_{\mathrm{ML}}] = \frac{N-1}{N}\boldsymbol{\Sigma}.$$

Biased estimate

• Note that the maximum likelihood estimate of § is biased.

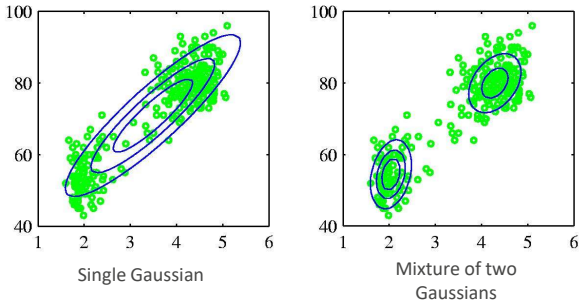
• We can correct the bias by defining a different estimator:

$$\widetilde{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})(\mathbf{x}_n - \boldsymbol{\mu}_{\mathrm{ML}})^{\mathrm{T}}.$$

Mixture of Gaussians

• When modeling real-world data, Gaussian assumption may not be appropriate.

• Consider the following example: Old Faithful Dataset



Mixture of Gaussians

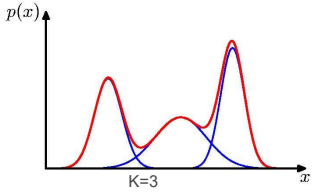
• We can combine simple models into a complex model by defining a superposition of K Gaussian densities of the form:

$$p(\mathbf{x}) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Component

Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K}\pi_k = 1$$

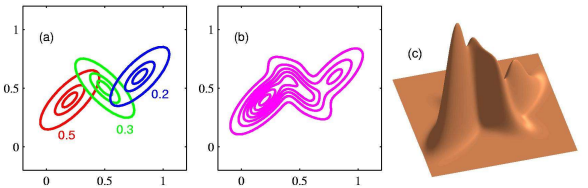


• Note that each Gaussian component has its own mean μ_k and covariance Σ_k. The parameters π_k are called mixing coefficients.

• More generally, mixture models can comprise linear combinations of other distributions.

Mixture of Gaussians

• Illustration of a mixture of 3 Gaussians in a 2-dimensional space:



(a) Contours of constant density of each of the mixture components, along with the mixing coefficients

(b) Contours of marginal probability density $p(\mathbf{x}) = \sum_{k=1}^{K}\pi_k\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$

(c) A surface plot of the distribution p(x).

Maximum Likelihood Estimation

• Given a dataset D, we can determine model parameters by maximizing the log-likelihood function:

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Log of a sum: no closed form solution

• **Solution**: use standard, iterative, numeric optimization methods or the Expectation Maximization algorithm.

The Exponential Distribution

The family of exponential distribution provides probability models that are very widely used.

Definition

X is said to have an **exponential distribution** with parameter $\lambda > 0$ if the pdf of *X* is

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The Gamma Distribution

Definition

A continuous random variable *X* is said to have a **gamma distribution** if the pdf of *X* is

$$f(x; \alpha, \beta) = \begin{cases} \frac{1}{\beta^{\alpha}\Gamma(\alpha)} x^{\alpha-1} e^{-x/\beta} & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where the parameters α and βsatisfy $\alpha > 0$, $\beta > 0$. The **standard gamma distribution** has $\beta = 1$.

We may be lazy and write Gam(x| α , β) for the above pdf instead.

Generation of Dirichlet from Gamma

- Take $Y_1 \sim \text{Gam}(\text{x}|\ \alpha_1\ , \beta)$, $Y_2 \sim \text{Gam}(\text{x}|\ \alpha_2\ , \beta)$, $\cdots\ Y_n \sim \text{Gam}(\text{x}|\ \alpha_n\ , \beta)$,
- Let $V = \sum_1^n Y_i$.
- Let

$$X_i = \frac{Y_i}{V}$$

- Then $(X_1, \cdots, X_n)$ are distributed according to Dirichlet with parameters $\alpha_1, \cdots\ \alpha_n$.

Student's t-Distribution

• Consider Student's t-Distribution

$$
\begin{aligned}
p(x|\mu,a,b) &= \int_0^\infty \mathcal{N}(x|\mu,\tau^{-1})\mathrm{Gam}(\tau|a,b)\,\mathrm{d}\tau \\
&= \int_0^\infty \mathcal{N}\left(x|\mu,(\eta\lambda)^{-1}\right)\mathrm{Gam}(\eta|\nu/2,\nu/2)\,\mathrm{d}\eta \\
&= \frac{\Gamma(\nu/2+1/2)}{\Gamma(\nu/2)}\left(\frac{\lambda}{\pi\nu}\right)^{1/2}\left[1+\frac{\lambda(x-\mu)^2}{\nu}\right]^{-\nu/2-1/2} \\
&= \mathrm{St}(x|\mu,\lambda,\nu)
\end{aligned}
$$

Infinite mixture of Gaussians

where

$$\lambda = a/b \qquad \eta = \tau b/a \qquad \nu = 2a.$$

Sometimes called the precision parameter.

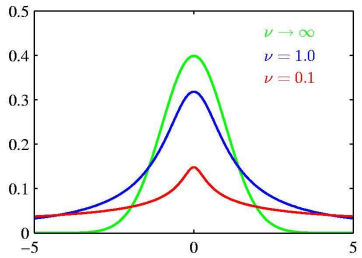
Degrees of freedom

Student's t-Distribution

• Setting $\nu = 1$ recovers Cauchy distribution

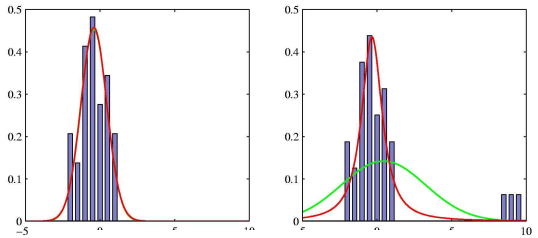
• The limit $\nu \to \infty$ corresponds to a Gaussian distribution.

	$\nu = 1$	$\nu \to \infty$
$\mathrm{St}(x\|\mu,\lambda,\nu)$	Cauchy	$\mathcal{N}(x\|\mu,\lambda^{-1})$



Student's t-Distribution

• Robustness to outliners: Gaussian vs. t-Distribution.



Student's t-Distribution

• The multivariate extension of the t-Distribution of dimension D:

$$
\begin{aligned}
\mathrm{St}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Lambda},\nu) &= \int_0^\infty \mathcal{N}(\mathbf{x}|\boldsymbol{\mu},(\eta\boldsymbol{\Lambda})^{-1})\mathrm{Gam}(\eta|\nu/2,\nu/2)\,\mathrm{d}\eta \\
&= \frac{\Gamma(D/2+\nu/2)}{\Gamma(\nu/2)}\frac{|\boldsymbol{\Lambda}|^{1/2}}{(\pi\nu)^{D/2}}\left[1+\frac{\Delta^2}{\nu}\right]^{-D/2-\nu/2}
\end{aligned}
$$

where $\Delta^2 = (\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Lambda}(\mathbf{x}-\boldsymbol{\mu})$

• Properties:

$$
\begin{aligned}
\mathbb{E}[\mathbf{x}] &= \boldsymbol{\mu}, && \text{if } \nu > 1 \\
\mathrm{cov}[\mathbf{x}] &= \frac{\nu}{(\nu-2)}\boldsymbol{\Lambda}^{-1}, && \text{if } \nu > 2 \\
\mathrm{mode}[\mathbf{x}] &= \boldsymbol{\mu}
\end{aligned}
$$

The Exponential Family

• The exponential family of distributions over **x** is defined to be a set of distributions of the form:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

where

- $\boldsymbol{\eta}$ is the vector of natural parameters
- $\mathbf{u}(\mathbf{x})$ is the vector of sufficient statistics

• The function g($\boldsymbol{\eta}$) can be iinterpreted as the coefficient that ensures that the distribution p($\mathbf{x}|\boldsymbol{\eta}$) is normalized:

$$g(\boldsymbol{\eta})\int h(\mathbf{x})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}\,\mathrm{d}\mathbf{x} = 1$$

Bernoulli Distribution

• The Bernoulli distribution is a member of the exponential family:

$$\begin{aligned} p(x|\mu) &= \mathrm{Bern}(x|\mu) = \mu^x(1-\mu)^{1-x} \\ &= \exp\{x\ln\mu + (1-x)\ln(1-\mu)\} \\ &= (1-\mu)\exp\left\{\ln\left(\frac{\mu}{1-\mu}\right)x\right\} \end{aligned}$$

• Comparing with the general form of the exponential family:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

we see that

$$\eta = \ln\left(\frac{\mu}{1-\mu}\right) \qquad \text{and so} \qquad \mu = \underbrace{\sigma(\eta) = \frac{1}{1+\exp(-\eta)}}_{\text{Logistic sigmoid}}.$$

Bernoulli Distribution

• The Bernoulli distribution is a member of the exponential family:

$$\begin{aligned} p(x|\mu) &= \mathrm{Bern}(x|\mu) = \mu^x(1-\mu)^{1-x} \\ &= \exp\{x\ln\mu + (1-x)\ln(1-\mu)\} \\ &= (1-\mu)\exp\left\{\ln\left(\frac{\mu}{1-\mu}\right)x\right\} \end{aligned}$$

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

• The Bernoulli distribution can therefore be written as:

$$p(x|\eta) = \sigma(-\eta)\exp(\eta x)$$

where

$$\begin{aligned} u(x) &= x \\ h(x) &= 1 \\ g(\eta) &= 1 - \sigma(\eta) = \sigma(-\eta). \end{aligned}$$

Multinomial Distribution

• The Multinomial distribution is a member of the exponential family:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{M}\mu_k^{x_k} = \exp\left\{\sum_{k=1}^{M}x_k\ln\mu_k\right\} = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left(\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right)$$

where $\mathbf{x} = (x_1,\ldots,x_M)^{\mathrm{T}}$ $\boldsymbol{\eta} = (\eta_1,\ldots,\eta_M)^{\mathrm{T}}$

and

$$\begin{aligned} \eta_k &= \ln\mu_k \\ \mathbf{u}(\mathbf{x}) &= \mathbf{x} \\ h(\mathbf{x}) &= 1 \\ g(\boldsymbol{\eta}) &= 1. \end{aligned}$$

NOTE: The parameters μ_k are not independent since the corresponding μ_k must satisfy

$$\sum_{k=1}^{M}\mu_k = 1.$$

• In some cases it will be convenient to remove the constraint by expressing the distribution over the M-1 parameters.

Multinomial Distribution

• The Multinomial distribution is a member of the exponential family:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{M} \mu_k^{x_k} = \exp\left\{\sum_{k=1}^{M} x_k \ln \mu_k\right\} = h(\mathbf{x}) g(\boldsymbol{\eta}) \exp\left(\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right)$$

• Let $\mu_M = 1 - \sum_{k=1}^{M-1} \mu_k$

• This leads to:

$$\eta_k = \ln\left(\frac{\mu_k}{1 - \sum_{j=1}^{M-1} \mu_j}\right) \quad \text{and} \quad \mu_k = \underbrace{\frac{\exp(\eta_k)}{1 + \sum_{j=1}^{M-1} \exp(\eta_j)}}_{\text{Softmax function}}.$$

• Here the parameters η_k are independent.

• Note that:

$$0 \leqslant \mu_k \leqslant 1 \quad \text{and} \quad \sum_{k=1}^{M-1} \mu_k \leqslant 1.$$

Multinomial Distribution

• The Multinomial distribution is a member of the exponential family:

$$p(\mathbf{x}|\boldsymbol{\mu}) = \prod_{k=1}^{M} \mu_k^{x_k} = \exp\left\{\sum_{k=1}^{M} x_k \ln \mu_k\right\} = h(\mathbf{x}) g(\boldsymbol{\eta}) \exp\left(\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right)$$

• The Multinomial distribution can therefore be written as:

$$p(\mathbf{x}|\boldsymbol{\mu}) = h(\mathbf{x}) g(\boldsymbol{\eta}) \exp\left(\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right)$$

where

$$\begin{aligned} \boldsymbol{\eta} &= (\eta_1, \ldots, \eta_{M-1}, 0)^{\mathrm{T}} \\ \mathbf{u}(\mathbf{x}) &= \mathbf{x} \\ h(\mathbf{x}) &= 1 \\ g(\boldsymbol{\eta}) &= \left(1 + \sum_{k=1}^{M-1} \exp(\eta_k)\right)^{-1}. \end{aligned}$$

Gaussian Distribution

• The Gaussian distribution can be written as:

$$\begin{aligned} p(x|\mu, \sigma^2) &= \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \\ &= \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}x^2 + \frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}\mu^2\right\} \\ &= h(x) g(\boldsymbol{\eta}) \exp\left\{\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(x)\right\} \end{aligned}$$

where

$$\boldsymbol{\eta} = \begin{pmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{pmatrix} \qquad h(\mathbf{x}) = (2\pi)^{-1/2}$$

$$\mathbf{u}(x) = \begin{pmatrix} x \\ x^2 \end{pmatrix} \qquad g(\boldsymbol{\eta}) = (-2\eta_2)^{1/2} \exp\left(\frac{\eta_1^2}{4\eta_2}\right).$$

ML for the Exponential Family

• Recall the Exponential Family:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x}) g(\boldsymbol{\eta}) \exp\left\{\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right\}$$

• From the definition of the normalizer g(´):

$$g(\boldsymbol{\eta}) \int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right\} \mathrm{d}\mathbf{x} = 1$$

• We can take a derivative w.r.t ´:

$$\nabla g(\boldsymbol{\eta}) \underbrace{\int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right\} \mathrm{d}\mathbf{x}}_{1/g(\boldsymbol{\eta})} + g(\boldsymbol{\eta}) \underbrace{\int h(\mathbf{x}) \exp\left\{\boldsymbol{\eta}^{\mathrm{T}} \mathbf{u}(\mathbf{x})\right\} \mathbf{u}(\mathbf{x}) \mathrm{d}\mathbf{x}}_{\mathbb{E}[\mathbf{u}(\mathbf{x})]} = 0$$

• Thus

$$-\nabla \ln g(\boldsymbol{\eta}) = \mathbb{E}[\mathbf{u}(\mathbf{x})]$$

ML for the Exponential Family

• Recall the Exponential Family:

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

• We can take a derivative w.r.t ´:

$$\nabla g(\boldsymbol{\eta}) \underbrace{\int h(\mathbf{x})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}\,\mathrm{d}\mathbf{x}}_{1/g(\boldsymbol{\eta})} + g(\boldsymbol{\eta}) \underbrace{\int h(\mathbf{x})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}\mathbf{u}(\mathbf{x})\,\mathrm{d}\mathbf{x}}_{\mathbb{E}[\mathbf{u}(\mathbf{x})]} = 0$$

• Thus

$$-\nabla \ln g(\boldsymbol{\eta}) = \mathbb{E}[\mathbf{u}(\mathbf{x})]$$

• Note that the covariance of **u**(**x**) can be expressed in terms of the second derivative of g(´), and similarly for the higher moments.

ML for the Exponential Family

• Suppose we observed i.i.d data $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$.

• We can construct the log-likelihood function, which is a function of the natural parameter ´.

$$p(\mathbf{x}|\boldsymbol{\eta}) = h(\mathbf{x})g(\boldsymbol{\eta})\exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\mathbf{u}(\mathbf{x})\right\}$$

$$p(\mathbf{X}|\boldsymbol{\eta}) = \left(\prod_{n=1}^{N} h(\mathbf{x}_n)\right) g(\boldsymbol{\eta})^N \exp\left\{\boldsymbol{\eta}^{\mathrm{T}}\sum_{n=1}^{N}\mathbf{u}(\mathbf{x}_n)\right\}.$$

• Therefore we have

$$-\nabla \ln g(\boldsymbol{\eta}_{\mathrm{ML}}) = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\mathbf{u}(\mathbf{x}_n)}_{\text{Sufficient Statistic}}$$

Review of Multivariable Calculus

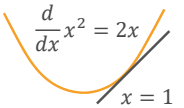
Review Scalar Derivative

y	a	x^n	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	nx^{n-1}	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

a is not a function of x

y	$u+v$	uv	$y = f(u), u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx}+\frac{dv}{dx}$	$\frac{du}{dx}v+\frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$

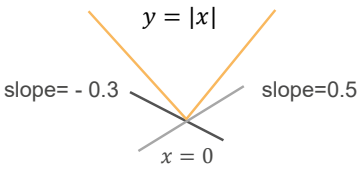
Derivative is the slope of the tangent line



The slope of the tangent line is 2

Subderivative

Extend derivative to non-differentiable cases



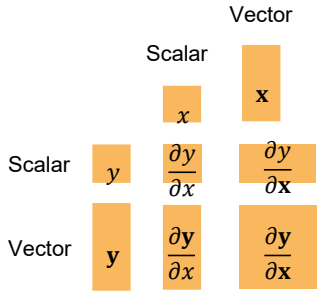
$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & if\ x > 0 \\ -1 & if\ x < 0 \\ a & if\ x = 0, a \in [-1,1] \end{cases}$$

Another example:

$$\frac{\partial}{\partial x} max(x, 0) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x < 0 \\ a & if\ x = 0, a \in [0,1] \end{cases}$$

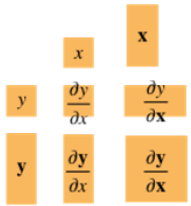
Gradients

Generalize derivatives into vectors



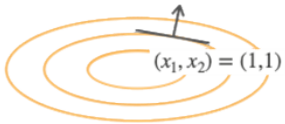
$\partial y / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \frac{\partial y}{\partial \mathbf{x}} = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_n}\right]$$



$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2]$$

Direction (2, 4), perpendicular to the contour lines



Examples

y	a	au	$sum(\mathbf{x})$	$\| \mathbf{x} \|^2$
$\frac{\partial y}{\partial \mathbf{x}}$	$\mathbf{0}^T$	$a\frac{\partial u}{\partial \mathbf{x}}$	$\mathbf{1}^T$	$2\mathbf{x}^T$

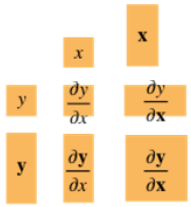
a is not a function of $\mathbf{x}$

$\mathbf{0}$ and $\mathbf{1}$ are vectors

y	$u + v$	uv	$\langle \mathbf{u}, \mathbf{v} \rangle$
$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}}$	$\frac{\partial u}{\partial \mathbf{x}} v + \frac{\partial v}{\partial \mathbf{x}} u$	$\mathbf{u}^T \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

$\partial \mathbf{y}/\partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$



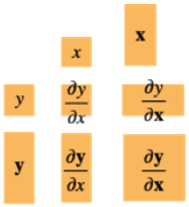
$\partial y/\partial \mathbf{x}$ is a row vector, while $\partial \mathbf{y}/\partial x$ is a column vector

It is called numerator-layout notation. The reversed version is called denominator-layout notation

$\partial \mathbf{y}/\partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \ldots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \ldots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \ldots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$



Examples

$\mathbf{y}$	$\mathbf{a}$	$\mathbf{x}$	$\mathbf{Ax}$	$\mathbf{x}^T\mathbf{A}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{A}$	$\mathbf{A}^T$

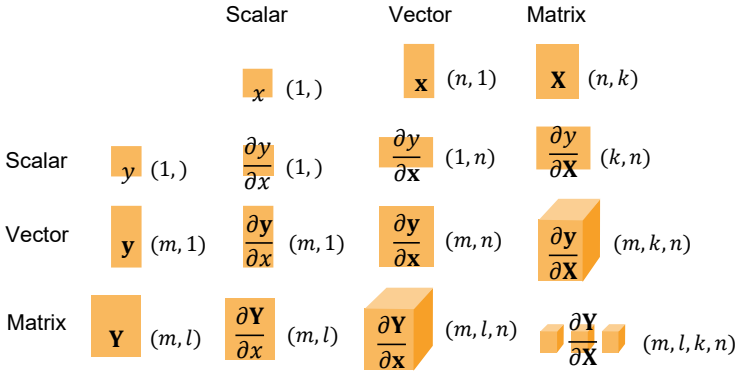
$\mathbf{y}$	$a\mathbf{u}$	$\mathbf{Au}$	$\mathbf{u}+\mathbf{v}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$a\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\mathbf{A}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}+\frac{\partial \mathbf{v}}{\partial \mathbf{x}}$

$\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^m, \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$

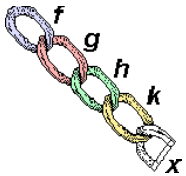
$a, \mathbf{a}$ and $\mathbf{A}$ are not functions of $\mathbf{x}$

$\mathbf{0}$ and $\mathbf{I}$ are matrices

Generalize to Matrices



Chain Rule



aws

Generalize to Vectors

Chain rule for scalars:

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Generalize to vectors straightforwardly

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial \mathbf{x}} \qquad \frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

$$(1,n)\ \ (1,)(1,n) \qquad (1,n)\ \ (1,k)\ (k,n) \qquad (m,n)\ (m,k)(k,n)$$

Example 1

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

Assume $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \quad y \in \mathbb{R}$

$$z = \left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)^2$$

Compute $\dfrac{\partial z}{\partial \mathbf{w}}$

$$\frac{\partial z}{\partial \mathbf{w}} = \frac{\partial z}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial \mathbf{w}}$$
$$= \frac{\partial b^2}{\partial b}\frac{\partial a - y}{\partial a}\frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}}$$
$$= 2b \cdot 1 \cdot \mathbf{x}^T$$
$$= 2\left(\langle \mathbf{x}, \mathbf{w} \rangle - y\right)\mathbf{x}^T$$

Decompose

$$a = \langle \mathbf{x}, \mathbf{w} \rangle$$
$$b = a - y$$
$$z = b^2$$

Example 2

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

Assume $\mathbf{X} \in \mathbb{R}^{m \times n}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m$

$$z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

Compute $\dfrac{\partial z}{\partial \mathbf{w}}$

$$\frac{\partial z}{\partial \mathbf{w}} = \frac{\partial z}{\partial \mathbf{b}}\frac{\partial \mathbf{b}}{\partial \mathbf{a}}\frac{\partial \mathbf{a}}{\partial \mathbf{w}}$$
$$= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}}\frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}}\frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}}$$
$$= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X}$$
$$= 2\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \mathbf{X}$$

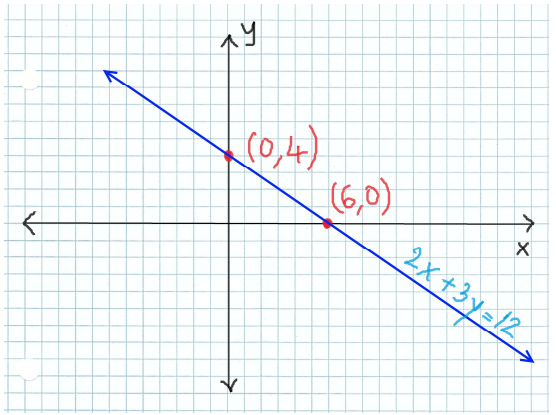
Decompose

$$\mathbf{a} = \mathbf{X}\mathbf{w}$$
$$\mathbf{b} = \mathbf{a} - \mathbf{y}$$
$$z = \|\mathbf{b}\|^2$$

Linear and Logistic Regression, Classification

Vahid Tarokh
ECE685D, Fall 2025

Linear Methods



aws

A Simplified Model

Assumption 1
The key factors impacting y are denoted by x_1, x_2, x_3

Assumption 2
The value of y is a weighted sum over the key factors

$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_0$$

Weights and bias are determined later.

Linear Least Squares

Given a vector of d-dimensional inputs $\mathbf{x} = (x_1, x_2, ..., x_d)^T$,we want to predict the target (response) using the linear model:

$$y(x, \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + ... + w_dx_d = w_0 + \sum_{j=1}^{d} w_jx_j.$$

The term w_0 is the intercept, or often called bias term. It will be convenient to include the constant variable 1 in **x** and write:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T\mathbf{w}.$$

Observe a training set consisting of N observations

$$\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)^T,$$

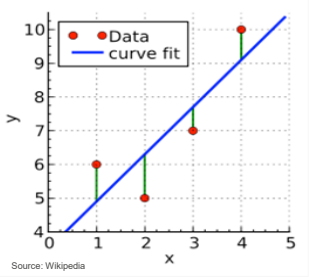
together with the corresponding target values

$$\mathbf{t} = (t_1, t_2, ..., t_N)^T.$$

Note that **X** is an $N \times (d + 1)$ matrix.

Linear Least Squares

One option is to minimize **the sum of the squares of the errors** between the predictions $y(\mathbf{x}_n, \mathbf{w})$ for each data point x_n and the corresponding real-valued targets t_n.



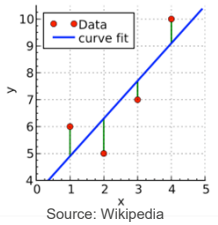
Loss function: sum-of-squared error function:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(\mathbf{x}_n^T\mathbf{w} - t_n)^2$$

$$= \frac{1}{2}(\mathbf{Xw} - \mathbf{t})^{\mathbf{T}}(\mathbf{Xw} - \mathbf{t}).$$

Linear Least Squares

If $\mathbf{X^T X}$ is nonsingular, then the unique solution is given by:



optimal weights

vector of target values

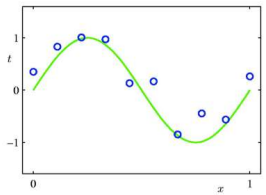
$$\mathbf{w}^* = (\mathbf{X^T X})^{-1}\mathbf{X^T t}$$

the design matrix has one input vector per row

- At an arbitrary input $\mathbf{x}_0$, the prediction is $y(\mathbf{x}_0, \mathbf{w}) = \mathbf{x}_0^T\mathbf{w}^*$.
- The entire model is characterized by d+1 parameters $\mathbf{w}^*$.

Example: Polynomial Curve Fitting

Consider observing a training set consisting of N 1-dimensional observations: $\mathbf{x} = (x_1, x_2, ..., x_N)^T$, together with corresponding real-valued targets: $\mathbf{t} = (t_1, t_2, ..., t_N)^T$.



- The green plot is the true function
- The training data was generated by $\sin(2\pi x)$. taking x_n spaced uniformly between [0 1].
- The target set (blue circles) was obtained by first computing the corresponding values of the sin function, and then adding a small Gaussian noise.

Goal: Fit the data using a polynomial function of the form:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

Note: the polynomial function is a nonlinear function of x, but it is a linear function of the coefficients **w** ! **Linear Models**.

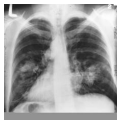
Classification

Classification

• The goal of classification is to assign an input **x** into one of K discrete classes C_k, where k=1,..,K.

• Typically, each input is assigned only to one class.

• Example: The input vector **x** is the set of pixel intensities, and the output variable t will represent the presence of cancer, class C_1, or absence of cancer, class C_2.



C_1: Cancer present

C_2: Cancer absent

x -- set of pixel intensities

Linear Classification

• The goal of classification is to assign an input **x** into one of K discrete classes C_k, where k=1,..,K.

• The input space is divided into decision regions whose boundaries are called decision boundaries or decision surfaces.

• We will consider linear models for classification. Remember, in the simplest linear regression case, the model is linear in parameters:

$$y(\mathbf{x}, \mathbf{w}) = \mathbf{x}^T\mathbf{w} + w_0.$$

adaptive parameters

$$y(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}^T\mathbf{w} + w_0).$$

fixed nonlinear function: activation function

• For classification, we need to predict discrete class labels, or posterior probabilities that lie in the range of (0,1), so we use a nonlinear function.

Linear Classification

$$y(\mathbf{x}, \mathbf{w}) = f(\mathbf{x}^T\mathbf{w} + w_0).$$

• The decision surfaces correspond to $y(\mathbf{x}, \mathbf{w}) = \text{const},$ so that $\mathbf{x}^T\mathbf{w} + w_0 = \text{const},$ and hence the decision surfaces are linear functions of **x**, even if the activation function is nonlinear.

• This class of models is called generalized linear models.

• Note that these models are no longer linear in parameters, due to the presence of nonlinear activation function.

• This leads to more complex analytical and computational properties, compared to linear regression.

• Note that we can make a fixed nonlinear transformation of the input variables using a vector of basis functions $\boldsymbol{\phi}(\mathbf{x}),$ as we did for regression models.

Notation

• In the case of two-class problems, we can use the binary representation for the target value $t \in \{0,1\}$ such that t=1 represents the positive class and t=0 represents the negative class.

- We can interpret the value of t as the probability of the positive class, and the output of the model can be represented as the probability that the model assigns to the positive class.

• If there are K classes, we use a 1-of-K encoding scheme, in which **t** is a vector of length K containing a single 1 for the correct class and 0 elsewhere.

• For example, if we have K=5 classes, then an input that belongs to class 2 would be given a target vector:

$$t = (0, 1, 0, 0, 0)^T.$$

- We can interpret a vector **t** as a vector of class probabilities.

Three Approaches to Classification

• First approach: Construct a discriminant function that directly maps each input vector to a specific class.

• Second approach: Model the decision regions and then use this to make optimal decisions.

• There are two alternative approaches:

- Discriminative Approach: Model $p(\mathcal{C}_k|\mathbf{x})$, directly, for example by representing them as parametric models, and optimize for parameters using the training set (e.g. logistic regression).

- Generative Approach: Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ together with the prior probabilities $p(\mathcal{C}_k)$ for the classes. Infer posterior probability using Bayes' rule:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

• For example, we could fit multivariate Gaussians to the input vectors of each class. Given a test vector, we see under which Gaussian the test vector is most probable.

Discriminant Functions

• Consider: $y(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + w_0$.

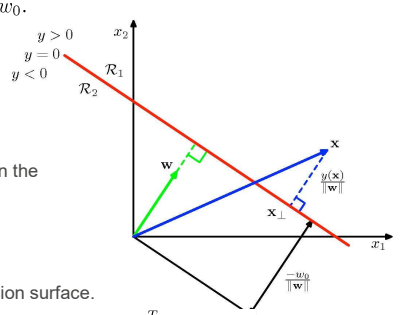
• Assign **x** to C_1 if $y(\mathbf{x}) \geq 0$, and class C_2 otherwise.

• Decision boundary:

$$y(\mathbf{x}) = 0.$$

• If two points $\mathbf{x_A}$ and $\mathbf{x_B}$ lie on the decision surface, then:

$$y(\mathbf{x}_A) = y(\mathbf{x}_B) = 0,$$
$$\mathbf{w}^T(\mathbf{x}_A - \mathbf{x}_B) = 0.$$



• **w** is orthogonal to the decision surface.

• If **x** is a point on the decision surface, then: $\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||}$.

• Hence w_0 determines the location of the decision surface.

Multiple Classes

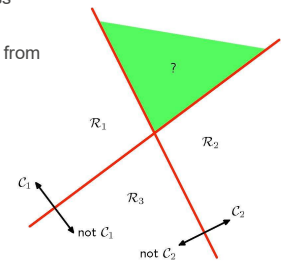
• Consider the extension of linear discriminants to K>2 classes.

• One option is to use K-1 classifiers, each of which solves a two class problem:

- Separate points in class C_k from points not in that class.

• There are regions in input space that are ambiguously classified.

One-versus-the-rest



Multiple Classes

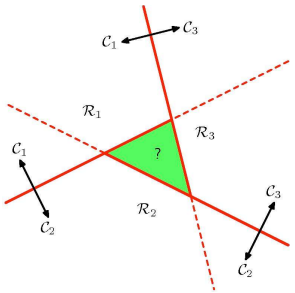
• Consider the extension of linear discriminants to K>2 classes.

• An alternative is to use K(K-1)/2 binary discriminant functions.

- Each function discriminates between two particular classes.

• Similar problem of ambiguous regions.

One-versus-one



Simple Solution

• Use K linear discriminant functions of the form:

$$y_k(\mathbf{x}) = \mathbf{x}^T\mathbf{w}_k + w_{k0}, \text{ where } k = 1, ..., K.$$

• Assign **x** to class C_k, if $y_k(\mathbf{x}) > y_j(\mathbf{x})\ \forall j \neq k$ (pick the max).

• This is guaranteed to give decision boundaries that are singly connected and convex.

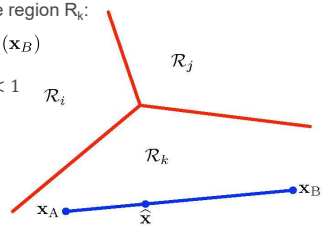
• For any two points that lie inside the region R_k:

$$y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A) \text{ and } y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$$

implies that for any positive $0 < \alpha < 1$

$$y_k(\alpha\mathbf{x}_A + (1-\alpha)\mathbf{x}_B) > y_j(\alpha\mathbf{x}_A + (1-\alpha)\mathbf{x}_B)$$

due to linearity of the discriminant functions.



The Perceptron Algorithm

• We now consider another example of a linear discriminant model.

• Consider the following generalized linear model of the form

$$y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x}))$$

where nonlinear activation function f(.) is given by a step function:

$$f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

and **x** is transformed using a fixed nonlinear transformation $\phi(\mathbf{x})$.

• Hence we have a two-class model.

The Perceptron Algorithm

• A natural choice of error function would be the total number of misclassified examples (but hard to optimize, discontinuous).

• We will consider an alternative error function.

• First, note that:

- Patterns x_n in Class C_1 should satisfy:

$$\mathbf{w}^T\phi(\mathbf{x}_n) > 0$$

- Patterns x_n in Class C_2 should satisfy:

$$\mathbf{w}^T\phi(\mathbf{x}_n) < 0$$

• Using the target coding t ∈ {-1, +1}, we see that we would like all patterns to satisfy:

$$\mathbf{w}^T\phi(\mathbf{x}_n)t_n > 0$$

Error Function

• Using the target coding t ∈ { -1, +1}, we see that we would like all patterns to satisfy:

$$\mathbf{w}^T\phi(\mathbf{x}_n)t_n > 0$$

• The error function is therefore given by:

$$E_P(\mathbf{w}) = -\sum_{n\in M} \mathbf{w}^T\phi(\mathbf{x}_n)t_n$$

M denotes the set of all misclassified patterns

• The error function is linear in **w** in regions of **w** space where the example is misclassified.

• The error function is piece-wise linear (show this).

Error Function

• We can use gradient descent. Given a misclassified example, the change in weight is given by:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \bigtriangledown E_p(\mathbf{w}) = \mathbf{w}^t + \eta\phi(\mathbf{x}_n)t_n,$$

where η is the learning rate.

• Since the perceptron function $y(\mathbf{x}) = f(\mathbf{w}^T\phi(\mathbf{x}))$ is unchanged if we multiple **w** by a constant, we set η = 1.

• Note that the contribution to the error from a misclassified example will be reduced:

$$\begin{aligned} -\mathbf{w}^{(t+t)T}\phi(\mathbf{x}_n)t_n &= -\mathbf{w}^{(t)T}\phi(\mathbf{x}_n)t_n - (\phi(\mathbf{x}_n)t_n)^T(\phi)(\mathbf{x}_n)t_n) \\ &< -\mathbf{w}^{(t)T}\phi(\mathbf{x}_n)t_n \end{aligned}$$

Always positive

Error Function

• Note that the contribution to the error from a misclassified example will be reduced:

$$\begin{aligned} -\mathbf{w}^{(t+t)T}\phi(\mathbf{x}_n)t_n &= -\mathbf{w}^{(t)T}\phi(\mathbf{x}_n)t_n - (\phi(\mathbf{x}_n)t_n)^T(\phi)(\mathbf{x}_n)t_n) \\ &< -\mathbf{w}^{(t)T}\phi(\mathbf{x}_n)t_n \end{aligned}$$

Always positive

• However, the change in **w** may cause some previously correctly classified points to be misclassified.

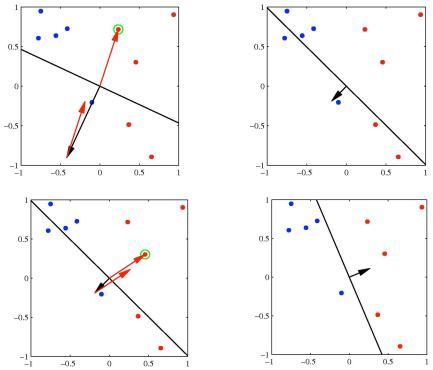
• No convergence guarantees in general.

• If there exists an exact solution (if the training set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in finite number of steps.

• The perceptron does not provide probabilistic outputs, nor does it generalize readily to K>2 classes.

Illustration of Convergence

• Convergence of the perceptron learning algorithm



Three Approaches to Classification

• Construct a discriminant function that directly maps each input vector to a specific class.

• Model the conditional probability distribution $p(\mathcal{C}_k|\mathbf{x})$, and then use this distribution to make optimal decisions.

• There are two alternative approaches:

- Discriminative Approach: Model $p(\mathcal{C}_k|\mathbf{x})$, directly, for example by representing them as parametric models, and optimize for parameters using the training set (e.g. logistic regression).

- Generative Approach: Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ together with the prior probabilities $p(\mathcal{C}_k)$ for the classes. Infer posterior probability using Bayes' rule:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

We will consider next.

Probabilistic Generative Models

• Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ separately for each class, as well as the class priors $p(\mathcal{C}_k)$.

• Consider the case of two classes. The posterior probability of class C_1 is given by:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$
$$= \frac{1}{1+\exp(-a)} = \sigma(a),$$

Logistic sigmoid function

where we defined:

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \ln \frac{p(\mathcal{C}_1|\mathbf{x})}{1-p(\mathcal{C}_1|\mathbf{x})},$$

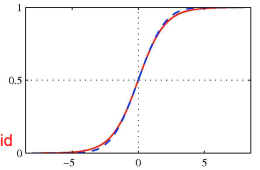
which is known as the logit function. It represents the log of the ratio of probabilities of two classes, also known as the log-odds.

Sigmoid Function

• The posterior probability of class C_1 is given by:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$
$$= \frac{1}{1+\exp(-a)} = \sigma(a),$$

Logistic sigmoid function



• The term sigmoid means S-shaped: it maps the whole real axis into (0 1).

• It satisfies:

$$\sigma(-a) = 1 - \sigma(a), \quad \frac{\mathrm{d}}{\mathrm{d}a}\sigma(a) = \sigma(a)(1-\sigma(a)).$$

Softmax Function

• For case of K>2 classes, we have the following multi-class generalization:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} = \frac{\exp(a_k)}{\sum_j \exp(a_j)}, \; a_k = \ln[p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)].$$

• This normalized exponential is also known as the softmax function, as it represents a smoothed version of the "max" function:

$$\text{if } a_k \gg a_j, \; \forall j \neq k, \; \text{then } p(\mathcal{C}_k|\mathbf{x}) \approx 1, \; p(\mathcal{C}_j|\mathbf{x}) \approx 0.$$

• We now look at some specific forms of class conditional distributions.

Example of Continuous Inputs

• Assume that the input vectors for each class are from a Gaussian distribution, and all classes share the same covariance matrix:

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right).$$

• For the case of two classes, the posterior is logistic function:

$$p(\mathcal{C}_k|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0),$$

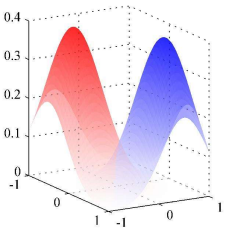
where we have defined:

$$\mathbf{w} = \mathbf{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2),$$
$$w_0 = -\frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}.$$

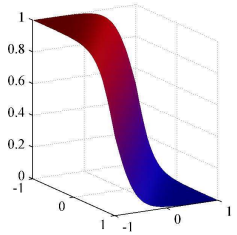
• The quadratic terms in **x** cancel (due to the assumption of common covariance matrices).

• This leads to a linear function of **x** in the argument of logistic sigmoid. Hence the decision boundaries are linear in input space.

Example of Two Gaussian Models



Class-conditional densities for two classes



The corresponding posterior probability $p(\mathcal{C}_1|\mathbf{x})$, given by the sigmoid function of a linear function of $\mathbf{x}$.

Case of K Classes

• For the case of K classes, the posterior is a softmax function:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}|\mathcal{C}_j)p(\mathcal{C}_j)} = \frac{\exp(a_k)}{\sum_j \exp(a_j)},$$

$$a_k = \mathbf{w}_k^T \mathbf{x} + w_{k0},$$

where, similar to the 2-class case, we have defined:

$$\mathbf{w}_k = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k,$$
$$w_{k0} = -\frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln p(\mathcal{C}_k).$$

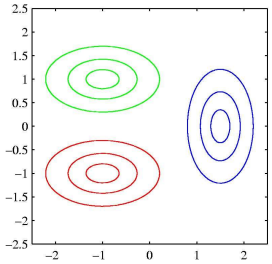
• Again, the decision boundaries are linear in input space.

• If we allow each class-conditional density to have its own covariance, we will obtain quadratic functions of $\mathbf{x}$.

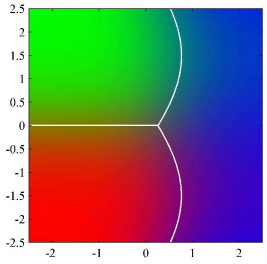
• This leads to a quadratic discriminant.

Quadratic Discriminant

The decision boundary is linear when the covariance matrices are the same and quadratic when they are not.



Class-conditional densities for three classes



The corresponding posterior probabilities for three classes.

Maximum Likelihood Solution

• Consider the case of two classes, each having a Gaussian class-conditional density with shared covariance matrix.

• We observe a dataset $\{\mathbf{x}_n, t_n\}$, $n = 1, .., N$.

- Here t_n=1 denotes class C_1, and t_n=0 denotes class C_2.
- Also denote $p(\mathcal{C}_1) = \pi$, $p(\mathcal{C}_2) = 1 - \pi$.

• The likelihood function takes form:

$$p(\mathbf{t}, \mathbf{X}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \left[\pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right]^{t_n} \left[(1-\pi)\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right]^{1-t_n}.$$

Data points from class C_1.

Data points from class C_2.

• As usual, we will maximize the log of the likelihood function.

Maximum Likelihood Solution

$$p(\mathbf{t}, \mathbf{X}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \left[\pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right]^{t_n} \left[(1-\pi)\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right]^{1-t_n}.$$

• Maximizing the respect to π, we look at the terms of the log-likelihood functions that depend on π :

$$\sum_n \left[t_n \ln \pi + (1-t_n)\ln(1-\pi)\right] + \text{const.}$$

Differentiating, we get:

$$\pi = \frac{1}{N}\sum_{n=1}^{N} t_n = \frac{N_1}{N_1 + N_2}.$$

• Maximizing the respect to μ_1, we look at the terms of the log-likelihood functions that depend on μ_1 :

$$\sum_n t_n \ln \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = -\frac{1}{2}\sum_n t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_1) + \text{const.}$$

Differentiating, we get:

$$\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n \mathbf{x}_n.$$

And similarly:

$$\boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{n=1}^{N} (1-t_n)\mathbf{x}_n.$$

Maximum Likelihood Solution

$$p(\mathbf{t}, \mathbf{X}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \left[\pi \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})\right]^{t_n} \left[(1-\pi)\mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right]^{1-t_n}.$$

• Maximizing the respect to Σ:

$$\begin{aligned} &-\frac{1}{2}\sum_n t_n \ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_n t_n (\mathbf{x}_n - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_1) \\ &-\frac{1}{2}\sum_n (1-t_n)\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_n (1-t_n)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_2) \\ &= -\frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{N}{2}\text{Tr}(\boldsymbol{\Sigma}^{-1}\mathbf{S}). \end{aligned}$$

• Here we defined:

$$\mathbf{S} = \frac{N_1}{N}\mathbf{S}_1 + \frac{N_2}{N}\mathbf{S}_2,$$

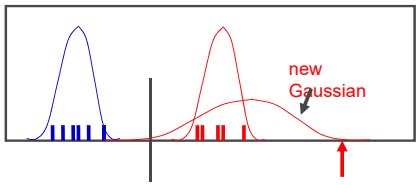
$$\mathbf{S}_1 = \frac{1}{N_1}\sum_{n\in\mathcal{C}_1} (\mathbf{x}_n - \boldsymbol{\mu}_1)(\mathbf{x}_n - \boldsymbol{\mu}_1)^T,$$

$$\mathbf{S}_2 = \frac{1}{N_2}\sum_{n\in\mathcal{C}_2} (\mathbf{x}_n - \boldsymbol{\mu}_2)(\mathbf{x}_n - \boldsymbol{\mu}_2)^T.$$

• Using standard results (see HW) for a Gaussian distribution we have: $\boldsymbol{\Sigma} = \mathbf{S}$.

• Maximum likelihood solution represents a weighted average of the covariance matrices associated with each of the two classes.

Example



• For generative fitting, the red mean moves rightwards but the decision boundary moves leftwards! If you believe the data is Gaussian, this is reasonable.

Three Approaches to Classification

• Construct a discriminant function that directly maps each input vector to a specific class.

• Model the conditional probability distribution $p(\mathcal{C}_k|\mathbf{x})$, and then use this distribution to make optimal decisions.

• There are two approaches:

- Discriminative Approach: Model $p(\mathcal{C}_k|\mathbf{x})$, directly, for example by representing them as parametric models, and optimize for parameters using the training set (e.g. logistic regression).
- Generative Approach: Model class conditional densities $p(\mathbf{x}|\mathcal{C}_k)$ together with the prior probabilities $p(\mathcal{C}_k)$ for the classes. Infer posterior probability using Bayes' rule:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})}.$$

We will consider next.

Fixed Basis Functions

• So far, we have considered classification models that work directly in the input space.

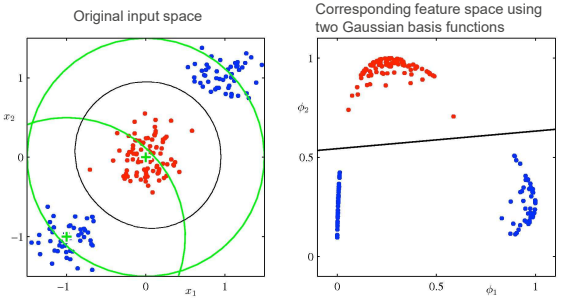
• All considered algorithms are equally applicable if we first make a fixed nonlinear transformation of the input space using vector of basis functions $\boldsymbol{\phi}(\mathbf{x})$.

• Decision boundaries will be linear in the feature space $\boldsymbol{\phi}$, but would correspond to nonlinear boundaries in the original input space $\mathbf{x}$.

• Classes that are linearly separable in the feature space $\boldsymbol{\phi}(\mathbf{x})$ need not be linearly separable in the original input space.

Logistic Regression

Linear Basis Function Models



• We define two Gaussian basis functions with centers shown by green the crosses, and with contours shown by the green circles.

• Linear decision boundary (right) is obtained using logistic regression, and corresponds to nonlinear decision boundary in the input space (left, black curve).

Logistic Regression

• Let us look at the two-class classification problem.

• We have seen that the posterior probability of class C_1 can be written as a sigmoid function:

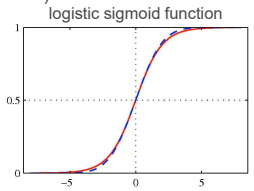
$$p(\mathcal{C}_1|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})} = \sigma(\mathbf{w}^T\mathbf{x}),$$

where $p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$, and we omit the bias term for clarity.

• This model is known as logistic regression (although this is a model for classification rather than regression).

Note that for generative models, we would first determine the class conditional densities and class-specific priors, and then use Bayes' rule to obtain the posterior probabilities.

Here we model $p(\mathcal{C}_k|\mathbf{x})$ directly.



Logistic Regression

• We observed a training dataset $\{\mathbf{x}_n, t_n\},\ n = 1, .., N;\ t_n \in \{0, 1\}$.

• Maximize the probability of getting the label right, so the likelihood function takes form:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} \left[y_n^{t_n} (1 - y_n)^{1-t_n} \right], \quad y_n = \sigma(\mathbf{w}^T \mathbf{x}_n).$$

• Taking the negative log of the likelihood, we can define the cross-entropy error function (that we want to minimize):

$$E(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} \left[t_n \ln y_n + (1 - t_n) \ln(1 - y_n) \right] = \sum_{n=1}^{N} E_n.$$

• Differentiating and using the chain rule:

$$\frac{\mathrm{d}}{\mathrm{d}y_n} E_n = \frac{y_n - t_n}{y_n(1 - y_n)}, \quad \frac{\mathrm{d}}{\mathrm{d}\mathbf{w}} y_n = y_n(1 - y_n)\mathbf{x}_n, \quad \boxed{\frac{\mathrm{d}}{\mathrm{d}a}\sigma(a) = \sigma(a)(1 - \sigma(a)).}$$

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{w}} E_n = \frac{\mathrm{d}E_n}{\mathrm{d}y_n} \frac{\mathrm{d}y_n}{\mathrm{d}\mathbf{w}} = (y_n - t_n)\mathbf{x}_n.$$

• Note that the factor involving the derivative of the logistic function cancelled.

ML for Logistic Regression

• We therefore obtain:

$$\nabla E(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\mathbf{x}_n.$$

prediction target

• This takes exactly the same form as the gradient of the sum-of-squares error function for the linear regression model.

• Unlike in linear regression, there is no closed form solution, due to nonlinearity of the logistic sigmoid function.

• The error function can be optimized using standard gradient-based (or more advanced) optimization techniques.

• Easy to adapt to the online learning setting.

Multiclass Logistic Regression

• For the multiclass case, we represent posterior probabilities by a softmax transformation of linear functions of input variables:

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x})}.$$

• Unlike in generative models, here we will use maximum likelihood to determine parameters of this discriminative model directly.

• As usual, we observed a dataset $\{\mathbf{x}_n, t_n\},\ n = 1, .., N,$ where we use 1-of-K encoding for the target vector $\mathbf{t}_n$.

• So if $\mathbf{x}_n$ belongs to class C_k, then $\mathbf{t}$ is a binary vector of length K containing a single 1 for element k (the correct class) and 0 elsewhere.

• For example, if we have K=5 classes, then an input that belongs to class 2 would be given a target vector:

$$t = (0, 1, 0, 0, 0)^T.$$

Multiclass Logistic Regression

• We can write down the likelihood function:

$$p(\mathbf{T}|\mathbf{X}, \mathbf{w}_1, ..., \mathbf{w}_K) = \prod_{n=1}^{N} \left[\prod_{k=1}^{K} p(\mathcal{C}_k|\mathbf{x}_n)^{t_{nk}} \right] = \prod_{n=1}^{N} \left[\prod_{k=1}^{K} y_{nk}^{t_{nk}} \right]$$

N × K binary matrix of target variables.

Only one term corresponding to correct class contributes.

where $y_{nk} = p(\mathcal{C}_k|\mathbf{x}_n) = \frac{\exp(\mathbf{w}_k^T \mathbf{x}_n)}{\sum_j \exp(\mathbf{w}_j^T \mathbf{x}_n)}$.

• Taking the negative logarithm gives the cross-entropy entropy function for multi-class classification problem:

$$E(\mathbf{w}_1, ..., \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{X}, \mathbf{w}_1, ..., \mathbf{w}_K) = -\sum_{n=1}^{N} \left[\sum_{k=1}^{K} t_{nk} \ln y_{nk} \right].$$

• Taking the gradient:

$$\nabla E_{\mathbf{w}_j}(\mathbf{w}_1, ...\mathbf{w}_K) = \sum_{n=1}^{N} (y_{nj} - t_{nj})\mathbf{x}_n.$$

Special Case of Softmax

• If we consider a softmax function for two classes:

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{\exp(a_1)}{\exp(a_1) + \exp(a_2)} = \frac{1}{1 + \exp(-(a_1 - a_2))} = \sigma(a_1 - a_2).$$

• So the logistic sigmoid is just a special case of the softmax function that avoids using redundant parameters:
- Adding the same constant to both a_1 and a_2 has no effect.
- The over-parameterization of the softmax is because probabilities must add up to one.

From Logistic Regression to Feed-Forward Neural Networks

Vahid Tarokh
ECE685D, Fall 2025

Introduction

- **We will next discuss logistic regression and the construction of neural Neural Networks.**
- **Important Note: Source of some of my slides (with great appreciation and acknowledgements)**
  - **Professor David Carlson Slides**
  - **Professor Alex Smola's slides (available online)**
  - **Professor Ruslan Salakhutdinov's slides (available online)**
  - **Professor Hugo Larochelle's class on Neural Networks**

Logistic Regression

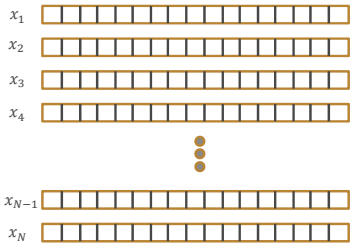
Learning a Predictive Model Based on Labeled Data

x, data/features for a subject

y, associated label 0/1

End goal: *predict* y from x

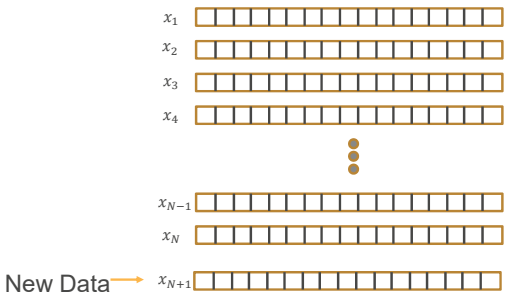
Training Set (Historical Data)



Start by limiting y to a binary outcome:

- False/True
- 0/1

Making Predictions

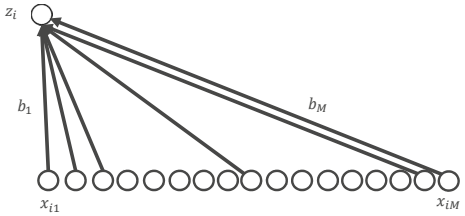


Start by limiting y to a binary outcome:

- False/True
- 0/1

Want to learn how to *predict* outcome

Linear Predictive Model



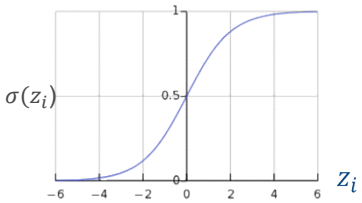
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

Convert to a Probability

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Extra Constant

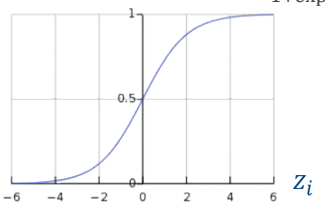
$$p(y_i = 1|x_i) = \sigma(z_i)$$



Convert to a Probability

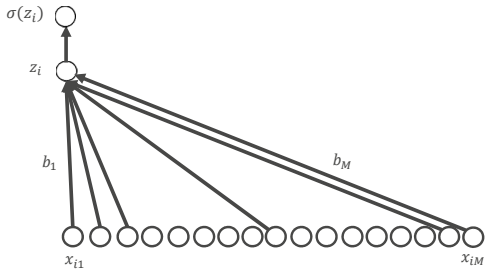
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

$$p(y_i = 1|x_i) = \sigma(z_i) = \frac{\exp(z_i)}{1+\exp(z_i)} = \frac{1}{1+\exp(-z_i)}$$

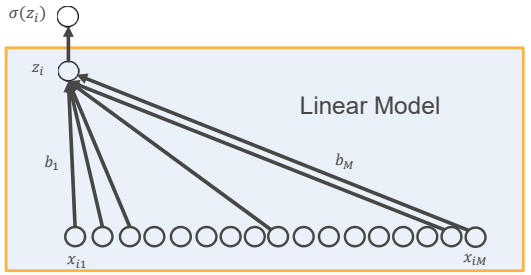


- Large and positive z_i indicates that event $y_i = 1$ is likely
- Large and negative z_i indicates that event $y_i = 0$ is likely

Logistic Regression

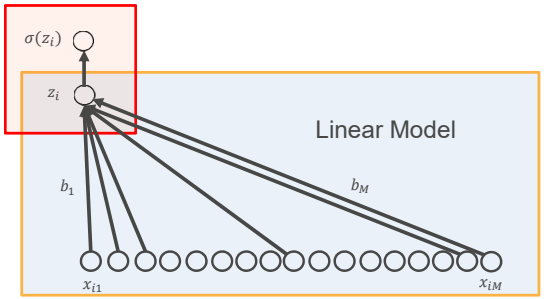


Logistic Regression



Logistic Regression

Convert to Probability



What do the parameters and model mean?

AN EXAMPLE

Example

Outcome:

- $y_i = 1$, it rains on day i;
- $y_i = 0$, it does not rain on day i

Features:

- On day i what is the $\{cloud\ cover, humidity, temperature, air\ pressure, \dots\}$

x_i, features for day i

y_i, did it rain on day i

Example

Outcome: $y_i = 1$, it rains on day i; $y_i = 0$, it does not rain on day i

Features: On day i what is the $\{1: cloud\ cover, 2: humidity, 3: temperature, \dots\}$

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Cloud Cover Humidity

- If cloud cover is positively related to rainfall, b_1 should be positive

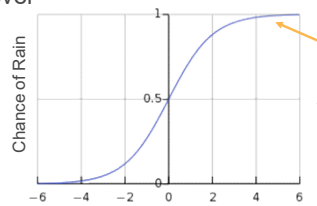
Impact on the Sigmoid Function

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Cloud Cover

Humidity

$$p(y_i = 1|x_i) = \sigma(z_i)$$



As the value z_i increases, the chance of rain increases

z_i

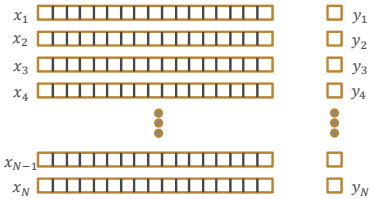
Building the Training Set

Need to learn the parameters

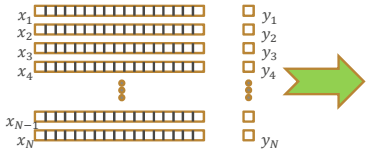
Requires *training data*

Record data from N days

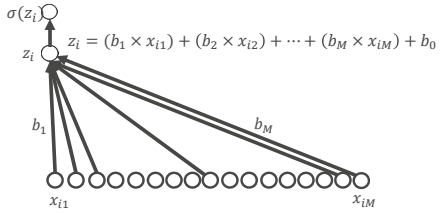
- Capture features: $\{cloud\ cover, humidity, temperature, \ldots\}$
- Did it rain?



Learning Model Parameters



Training Set



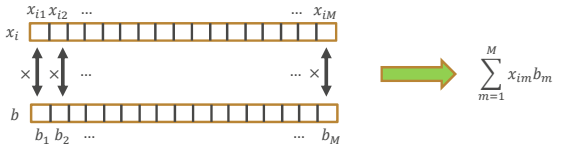
Logistic Regression Model (or "Network")

Learned Parameters

$(b_0, b_1, \ldots b_N)$

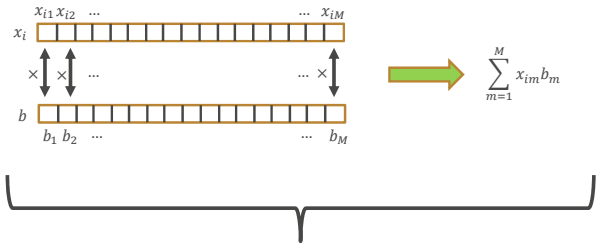
Interpretation of Logistic Regression

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$



Interpretation of Logistic Regression

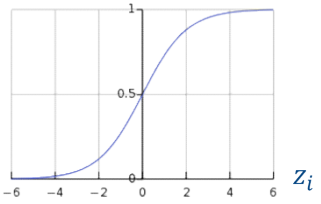
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$



Interpretation of Logistic Regression

$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

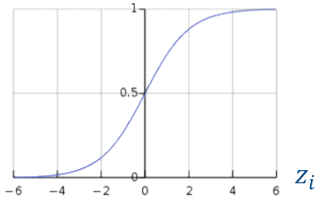
$$p(y_i = 1|x_i) = \sigma(z_i)$$



Interpretation of Logistic Regression

$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

$$p(y_i = 1|x_i) = \sigma(z_i)$$



- May think of vector b as a template or filter (will visualize to make clear)
- If x_i is aligned/matched with b, then the sum will be larger
- The parameter b_0 is a bias to correct for class prevalences

Artificial Neurons

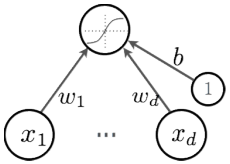
Artificial Neuron

• Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

• Neuron output activation:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



where

$\mathbf{w}$ are the weights (parameters)

b is the bias term

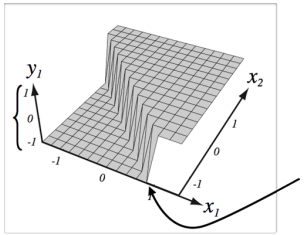
$g(\cdot)$ is called the activation function

Artificial Neuron

• Output activation of the neuron:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is determined by $g(\cdot)$



Bias only changes the position of the riff

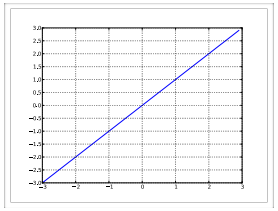
(from Pascal Vincent's slides)

Activation Function

• Linear activation function:

$$g(a) = a$$

- No nonlinear transformation
- No input squashing

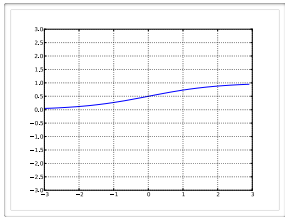


Activation Function

• Sigmoid activation function:

$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

- Squashes the neuron's output between 0 and 1
- Always positive
- Bounded
- Strictly Increasing



Does this ring a bell?

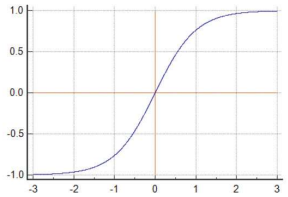
Activation Function

• Hyperbolic tangent (“tanh”) activation function:

- Squashes the neuron’s activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

(wrong plot)

$$g(a) = \tanh(a) = \\ = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

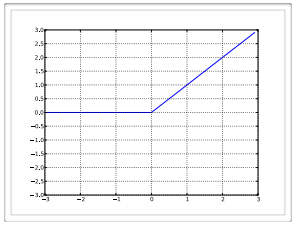


Activation Function

• Rectified linear (ReLU) activation function:

- Bounded below by 0 (always non-negative)
- Tends to produce units with sparse activities
- Not upper bounded
- Strictly increasing

$$g(a) = \text{reclin}(a) = \max(0, a)$$

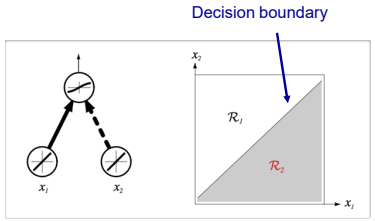


Decision Boundary of a Neuron

• Binary classification:

- With sigmoid, one can interpret neuron as estimating $p(y = 1|\mathbf{x})$
- Interpret as a logistic classifier
- If activation is greater than 0.5, predict 1
- Otherwise predict 0

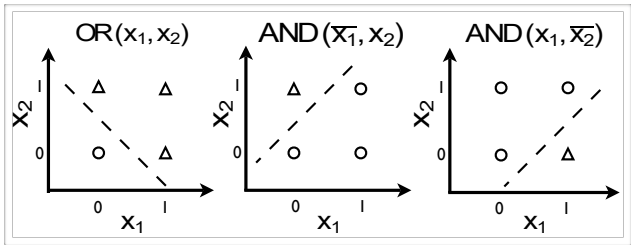
Same idea can be applied to a tanh( ) activation



(from Pascal Vincent’s slides)

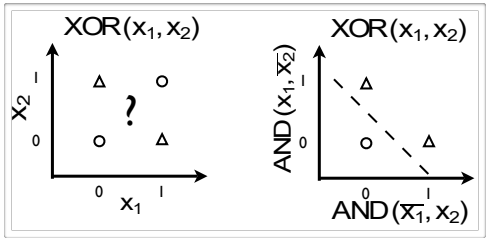
Capacity of a Single Neuron

• Can solve linearly separable problems.



Capacity of a Single Neuron

• Can not solve non-linearly separable problems.

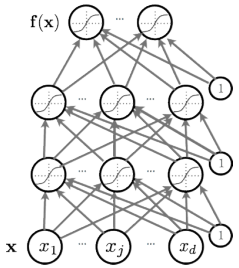


• Need to transform the input into a better representation.
• Remember basis functions!

Feed-Forward Neural Nets

Feedforward Neural Networks

- How neural networks predict f(x) given an input x:
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Back-propagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization
  - Unsupervised Pre-training



Single Hidden Layer Neural Net

• Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

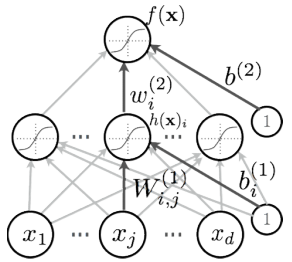
• Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

• Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

Output activation function



Softmax Activation Function

- Remember multi-way classification:
  - We need multiple outputs (1 output per class)
  - We need to estimate conditional probability: $p(y = c|\mathbf{x})$
  - Discriminative Learning

- Softmax activation function at the output

$$\mathbf{o}(\mathbf{a}) = \text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

  - strictly positive
  - sums to one

- Predict class with the highest estimated class conditional probability.

Multilayer Neural Net

• Consider a network with L hidden layers.

- layer pre-activation for k>0

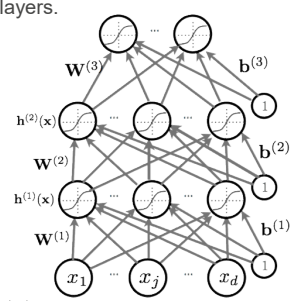
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- output layer activation (k=L+1):

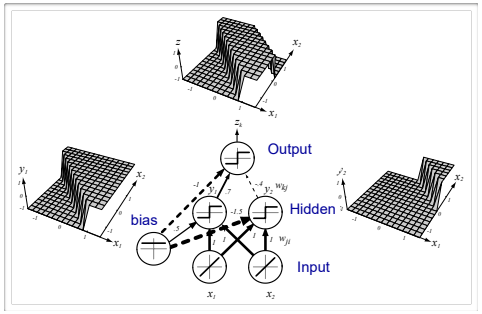
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$

Capacity of Neural Nets

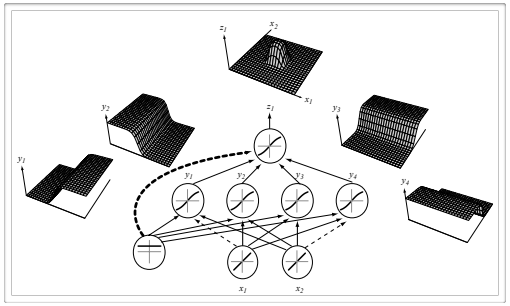
• Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

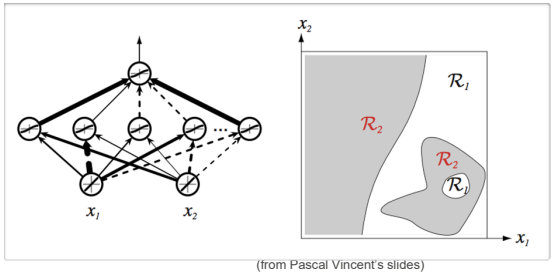
• Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

- Consider a single layer neural network



(from Pascal Vincent's slides)

Universal Approximation

The key result is due to great mathematicians Kolmogorov and Arnold (very difficult to prove) established in 1956.

Any continuous function of m inputs can be represented **exactly** by a small (polynomial sized) two-layer network.

$$f(x_1, \ldots, x_m) = \sum_{i=1}^{2m+1} g_i \left(\sum_{j=1}^{m} h_{i,j}(x_j) \right)$$

Where g_i and $h_{i,j}$ are continuous scalar-to-scalar functions.

Universal Approximation

A much more trivial result to prove is:

For any (possibly discontinuous) $f : [0,1]^m \to \mathbb{R}$ we have

$$f(x_1, \ldots, x_m) = g \left(\sum_i h_i(x_i) \right)$$

for (discontinuous) scalar-to-scalar functions g and h_i.

Proof: Any single real number contains an infinite amount of information.

Select h_i to spread out the digits of its argument so that $\sum_i h_i(x_i)$ contains all the digits of all the x_i.

Universal Approximation

Another relatively straightforward result is due to Cybenko (1989):

Any continuous function can be approximated arbitrarily well by a two layer perceptron.

For any continuous $f : [0,1]^m \to \mathbb{R}$ and any $\varepsilon > 0$, there exists

$$F(x) = \alpha \cdot \sigma(Wx + \beta)$$

$$= \sum_i \alpha_i \sigma \left(\sum_j W_{i,j}\, x_j + \beta_i \right)$$

such that for all x in $[0,1]^m$ we have $|F(x) - f(x)| < \varepsilon$.

Universal Approximation

• Universal Approximation Theorem (Hornik, 1991):

- "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units''

• This applies for sigmoid, tanh and many other activation functions.

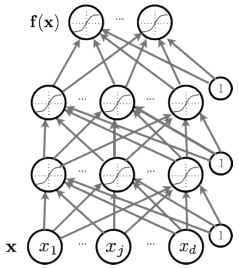
• However, this does not mean that there is learning algorithm that can find the necessary parameter values.

How to Train Neural Networks

Vahid Tarokh
ECE 685D, Fall 2025

Feedforward Neural Networks

- How neural networks predict f(x) given an input x:
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Back-propagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization
  - Unsupervised Pre-training



Training

• Empirical Risk Minimization:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{T} \sum_t \underbrace{l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})}_{\text{Loss function}} + \underbrace{\lambda \Omega(\boldsymbol{\theta})}_{\text{Regularizer}}$$

• Learning is cast as optimization.

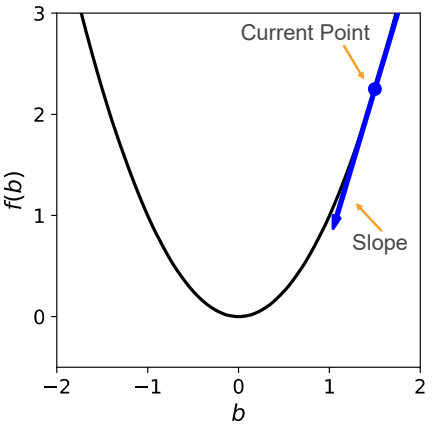
➢ For classification problems, we would like to minimize classification error.

➢ Loss function can sometimes be viewed as a surrogate for what we want to optimize (e.g. upper bound)

GRADIENT DESCENT

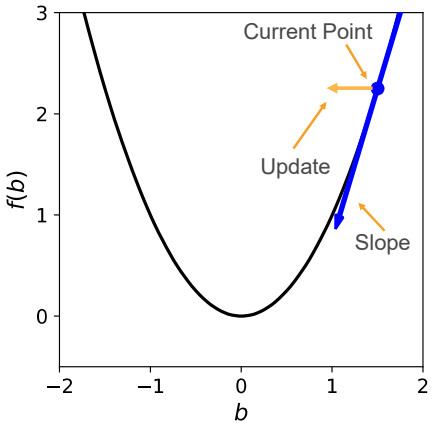
Visualization of Optimization Method

- We want to minimize a mathematical function (i.e. our average loss function)
- One approach is to:
  1. Find the direction pointing "down the hill" (towards a smaller value)
  2. Move a bit in that direction
  3. Repeat 1-2 until satisfied



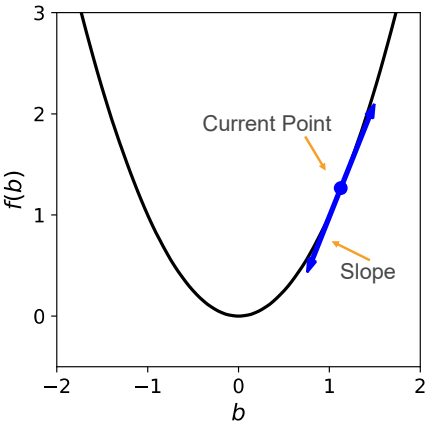
Visualization of Optimization Method

- We want to minimize a mathematical function (i.e. our average loss function)
- One approach is to:
  1. Find the direction pointing "down the hill" (towards a smaller value)
  2. Move a bit in that direction
  3. Repeat 1-2 until satisfied



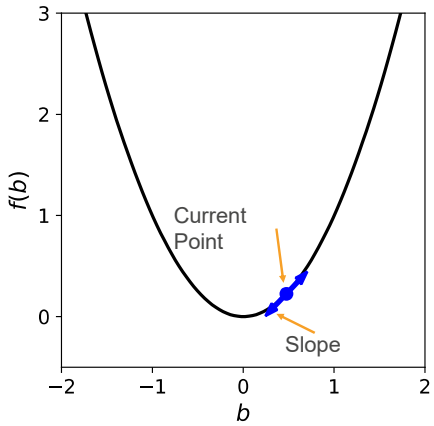
Visualization of Optimization Method

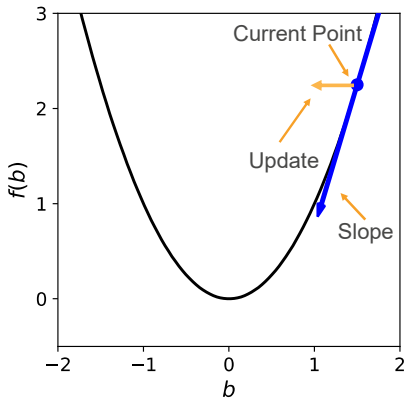
- We want to minimize a mathematical function (i.e. our average loss function)
- One approach is to:
  1. Find the direction pointing "down the hill" (towards a smaller value)
  2. Move a bit in that direction
  3. Repeat 1-2 until satisfied
- This shows the **first** update



Visualization of Optimization Method

- We want to minimize a mathematical function (i.e. our average loss function)
- One approach is to:
  1. Find the direction pointing "down the hill" (towards a smaller value)
  2. Move a bit in that direction
  3. Repeat 1-2 until satisfied
- This shows the **fourth** update





Mathematical Description of Gradient Descent

- We want to minimize a function
  $$b^* = \arg\min_b f(b)$$
- Start at an initial value b^0
- We will run a series of updates to move from b^k to b^{k+1} (i.e. from b^0 to b^1)
- Iteratively run the procedure:
  1. Calculate the slope at the current point (For one parameter, this is the derivative. For multiple parameters, this is the *gradient*.):
     $\nabla f(b^k)$,
     ∇ means gradient or multidimensional slope
  2. Move in the direction of the negative gradient with *step size* α^k:
     $b^{k+1} = b^k - \alpha^k \nabla f(b^k)$
  3. Repeat 1-2 until converged

STOCHASTICS GRADIENT DESCENT

Stochastic Gradient Descent

• Perform updates after seeing each example:

- Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
- For t=1:T
  - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$$

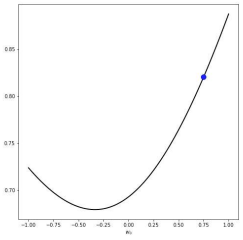
Training epoch = Iteration of all examples

• To train a neural net, we need:

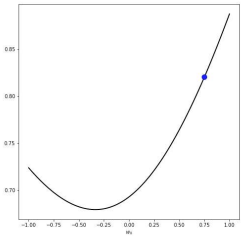
- Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

Videos for Visualization

Gradient Descent



Stochastic Gradient Descent



Comments on SGD

Stochastic Gradient Descent can update *many more times* than Gradient Descent

Gets ***near*** the solution very quickly

Allows scaling to *big data* (update time doesn't increase with the data size)

In practice, we often use a minibatch, which uses a few data examples to estimate the gradient

Loss Function

• Let us start by considering a classification problem with a softmax output layer.

• We need to estimate: $f(\mathbf{x})_c = p(y = c|\mathbf{x})$

- We can maximize the log-probability of the correct class given an input: $\log p(y^{(t)} = c|x^{(t)})$

• Alternatively, we can minimize the negative log-likelihood:

$$l(\mathbf{f}(\mathbf{x}), y) = -\sum_c 1_{(y=c)} \log f(\mathbf{x})_c = -\log f(\mathbf{x})_y$$

• This is also known as a cross-entropy entropy function for multi-class classification problem (will be discussed more later on).

Stochastic Gradient Descent

• Perform updates after seeing each example:

- Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$

- For t=1:T

  - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$

Training epoch = Iteration of all examples

• To train a neural net, we need:

- Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

Multilayer Neural Net: Reminder

• Consider a network with L hidden layers.

- layer pre-activation for k>0

$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

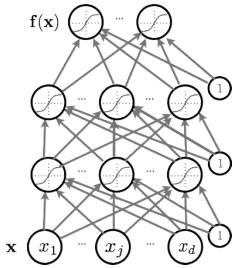
- hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- output layer activation (k=L+1):

$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$

Softmax activation function



Gradient Computation

• Loss gradient at output

- Partial derivative:

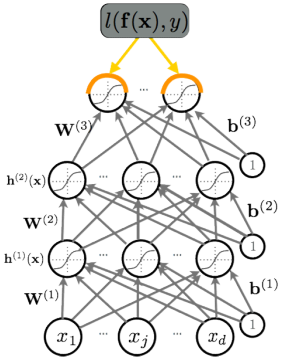
$$\frac{\partial}{\partial f(\mathbf{x})_c} - \log f(\mathbf{x})_y = \frac{-1_{(y=c)}}{f(\mathbf{x})_y}$$

- Gradient:

$$\nabla_{\mathbf{f}(\mathbf{x})} - \log f(\mathbf{x})_y = \frac{-1}{f(\mathbf{x})_y}\begin{bmatrix} 1_{(y=0)} \\ \vdots \\ 1_{(y=C-1)} \end{bmatrix} = \frac{-\mathbf{e}(y)}{f(\mathbf{x})_y}$$

Indicator function

Remember: $f(\mathbf{x})_c = p(y = c|\mathbf{x})$



Gradient Computation

• Loss gradient at output pre-activation

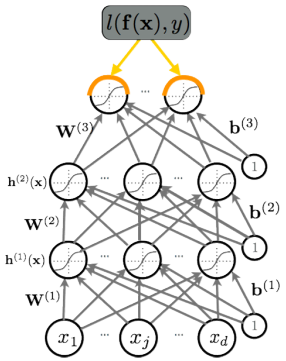
- Partial derivative:

$$\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y = -\left(1_{(y=c)} - f(\mathbf{x})_c\right)$$

- Gradient:

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y = -\left(\mathbf{e}(y) - \mathbf{f}(\mathbf{x})\right)$$

Indicator function



Derivation

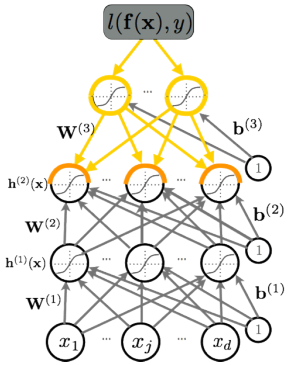
$$\begin{aligned}
&\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} - \log f(\mathbf{x})_y \\
&= \frac{-1}{f(\mathbf{x})_y}\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} f(\mathbf{x})_y \\
&= \frac{-1}{f(\mathbf{x})_y}\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c} \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y \\
&= \frac{-1}{f(\mathbf{x})_y}\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} \\
&= \frac{-1}{f(\mathbf{x})_y}\left(\frac{\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)\left(\frac{\partial}{\partial a^{(L+1)}(\mathbf{x})_c}\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)}{\left(\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})\right)^2}\right) \\
&= \frac{-1}{f(\mathbf{x})_y}\left(\frac{1_{(y=c)}\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})} - \frac{\exp(a^{(L+1)}(\mathbf{x})_y)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\frac{\exp(a^{(L+1)}(\mathbf{x})_c)}{\sum_{c'}\exp(a^{(L+1)}(\mathbf{x})_{c'})}\right) \\
&= \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y - \mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_y\,\mathrm{softmax}(\mathbf{a}^{(L+1)}(\mathbf{x}))_c\right) \\
&= \frac{-1}{f(\mathbf{x})_y}\left(1_{(y=c)}f(\mathbf{x})_y - f(\mathbf{x})_y\,f(\mathbf{x})_c\right) \\
&= -\left(1_{(y=c)} - f(\mathbf{x})_c\right)
\end{aligned}$$

$$\frac{\partial \frac{g(x)}{h(x)}}{\partial x} = \frac{\partial g(x)}{\partial x}\frac{1}{h(x)} - \frac{g(x)}{h(x)^2}\frac{\partial h(x)}{\partial x}$$

Gradient Computation

• Loss gradient for hidden layers

- This is getting complicated!



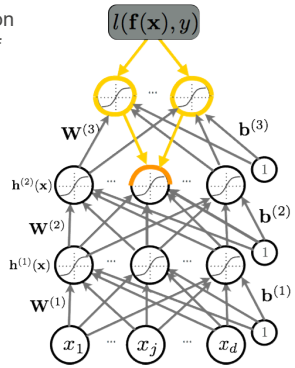
Gradient Computation

• Chain Rule: Assume that a function $p(a)$ can be written as a function of intermediate results $q_i(a)$, then:

$$\frac{\partial p(a)}{\partial a} = \sum_i \frac{\partial p(a)}{\partial q_i(a)} \frac{\partial q_i(a)}{\partial a}$$

• We can invoke it by setting:

- a be a hidden unit
- $q_i(a)$ be a pre-activation in the layer above
- $p(a)$ be the loss function



Gradient Computation

• Loss gradient at hidden layers

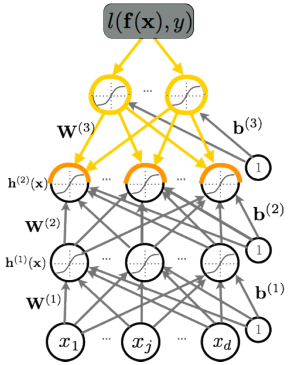
- Partial derivative:

$$\frac{\partial}{\partial h^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y$$

$$= \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} \frac{\partial a^{(k+1)}(\mathbf{x})_i}{\partial h^{(k)}(\mathbf{x})_j}$$

$$= \sum_i \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k+1)}(\mathbf{x})_i} W_{i,j}^{(k+1)}$$

Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$



Gradient Computation

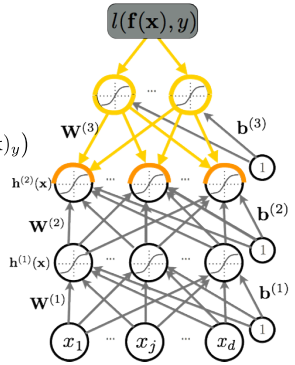
• Loss gradient at hidden layers

- Gradient

$$\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

$$= \mathbf{W}^{(k+1)\top} \underbrace{\left(\nabla_{\mathbf{a}^{(k+1)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)}_{\text{We already know how to compute that}}$$

Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$



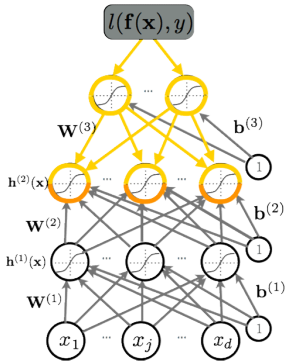
Gradient Computation

• Loss gradient at hidden layers (pre-activation)

- Partial derivative:

$$\frac{\partial}{\partial a^{(k)}(\mathbf{x})_j} - \log f(\mathbf{x})_y$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} \frac{\partial h^{(k)}(\mathbf{x})_j}{\partial a^{(k)}(\mathbf{x})_j}$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial h^{(k)}(\mathbf{x})_j} g'(a^{(k)}(\mathbf{x})_j)$$

Remember:
$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$



Gradient Computation

• Loss gradient at hidden layers (pre-activation)

Let's look at the gradients of activation functions.

- Gradient:

$$\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$
$$= \left(\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)^\top \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} \mathbf{h}^{(k)}(\mathbf{x})$$
$$= \left(\nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \odot [\ldots, g'(a^{(k)}(\mathbf{x})_j), \ldots]$$

Gradient of the activation function

Remember:
$h^{(k)}(\mathbf{x})_j = g(a^{(k)}(\mathbf{x})_j)$

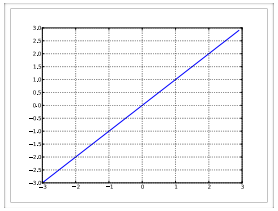
Linear Activation Function Gradient

• Linear activation function:

$$g(a) = a$$

- Partial derivative

$$g'(a) = 1$$



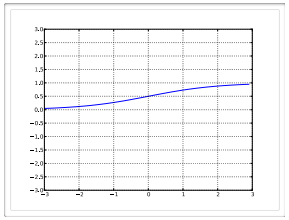
Sigmoid Activation Function Gradient

• Sigmoid activation function:

$$g(a) = \text{sigm}(a) = \frac{1}{1+\exp(-a)}$$

- Partial derivative

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



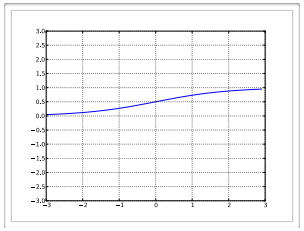
Tanh Activation Function Gradient

• Hyperbolic tangent (“tanh”) activation function:

$$g(a) = \tanh(a) = \\ = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{\exp(2a)-1}{\exp(2a)+1}$$

- Partial derivative

$$g'(a) = 1 - g(a)^2$$



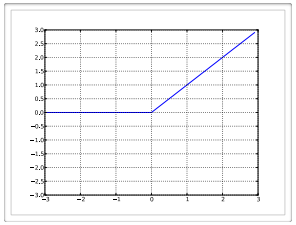
Tanh Activation Function Gradient

• Rectified linear (ReLU) activation function:

$$g(a) = \text{reclin}(a) = \max(0, a)$$

- Partial derivative

$$g'(a) = 1_{a>0}$$



Stochastic Gradient Descent

• Perform updates after seeing each example:

- Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$
- For t=1:T
  - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \Delta$$

Training epoch = Iteration of all examples

• To train a neural net, we need:

- Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$
- Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$,$\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

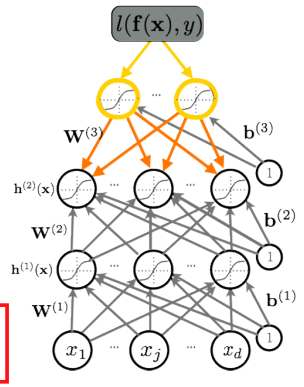
Gradient Computation

• Loss gradient of parameters

- Partial derivative (weights):

$$\frac{\partial}{\partial W_{i,j}^{(k)}} - \log f(\mathbf{x})_y$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial W_{i,j}^{(k)}}$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} h_j^{(k-1)}(\mathbf{x})$$

Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$

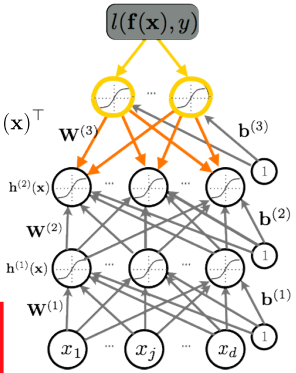


Gradient Computation

• Loss gradient of parameters

- Gradient (weights):

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y$$
$$= \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$



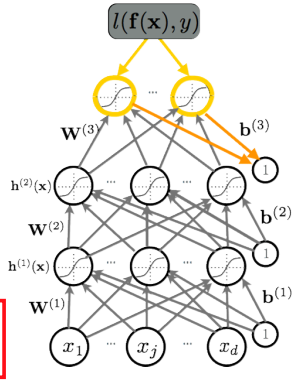
Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$

Gradient Computation

• Loss gradient of parameters

- Partial derivative (biases):

$$\frac{\partial}{\partial b_i^{(k)}} - \log f(\mathbf{x})_y$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i} \frac{\partial a^{(k)}(\mathbf{x})_i}{\partial b_i^{(k)}}$$
$$= \frac{\partial - \log f(\mathbf{x})_y}{\partial a^{(k)}(\mathbf{x})_i}$$



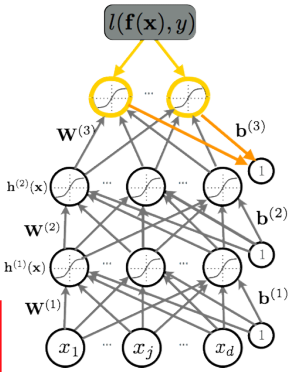
Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$

Gradient Computation

• Loss gradient of parameters

- Gradient (biases):

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y$$
$$= \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$



Remember:
$a^{(k)}(\mathbf{x})_i = b_i^{(k)} + \sum_j W_{i,j}^{(k)} h^{(k-1)}(\mathbf{x})_j$

Backpropagation Algorithm

• Perform forward propagation

• Compute output gradient (before activation):

$$\nabla_{\mathbf{a}^{(L+1)}(\mathbf{x})} - \log f(\mathbf{x})_y \Longleftarrow -(\mathbf{e}(y) - \mathbf{f}(\mathbf{x}))$$

• For k=L+1 to 1

- Compute gradients w.r.t. the hidden layer parameters:

$$\nabla_{\mathbf{W}^{(k)}} - \log f(\mathbf{x})_y \Longleftarrow \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \mathbf{h}^{(k-1)}(\mathbf{x})^\top$$

$$\nabla_{\mathbf{b}^{(k)}} - \log f(\mathbf{x})_y \Longleftarrow \nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y$$

- Compute gradients w.r.t. the hidden layer below:

$$\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \Longleftarrow \mathbf{W}^{(k)\top} \left(\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \log f(\mathbf{x})_y\right)$$

- Compute gradients w.r.t. the hidden layer below (before activation):

$$\nabla_{\mathbf{a}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y \Longleftarrow \left(\nabla_{\mathbf{h}^{(k-1)}(\mathbf{x})} - \log f(\mathbf{x})_y\right) \odot [\ldots, g'(a^{(k-1)}(\mathbf{x})_j), \ldots]$$

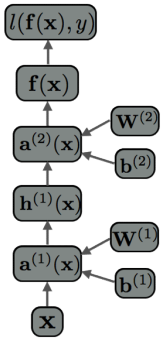
Computational Flow Graph

• Forward propagation can be represented as an acyclic flow graph

• Forward propagation can be implemented in a modular way:

➢ Each box can be an object with an fprop method, that computes the value of the box given its children

➢ Calling the fprop method of each box in the right order yields forward propagation



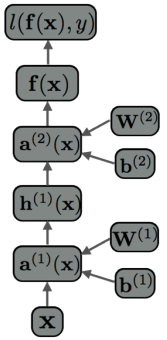
Computational Flow Graph

• Each object also has a bprop method

- it computes the gradient of the loss with respect to each child box.

- fprop depends on the fprop output of box's children, while bprop depends on the bprop of box's parents

• By calling bprop in the reverse order, we obtain backpropagation



Stochastic Gradient Descent

• Perform updates after seeing each example:

- Initialize: $\boldsymbol{\theta} \equiv \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$

- For t=1:T

  - for each training example $(\mathbf{x}^{(t)}, y^{(t)})$

$$\Delta = -\nabla_{\boldsymbol{\theta}} l(f(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) - \lambda \nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$$

Training epoch = Iteration of all examples

• To train a neural net, we need:

➢ Loss function: $l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

➢ A procedure to compute gradients: $\nabla_{\boldsymbol{\theta}} l(\mathbf{f}(\mathbf{x}^{(t)}; \boldsymbol{\theta}), y^{(t)})$

➢ Regularizer and its gradient: $\Omega(\boldsymbol{\theta})$,$\nabla_{\boldsymbol{\theta}} \Omega(\boldsymbol{\theta})$

Weight Decay

• L^2 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j \left(W_{i,j}^{(k)}\right)^2 = \sum_k ||\mathbf{W}^{(k)}||_F^2$$

• Gradient:

$$\nabla_{\mathbf{W}^{(k)}} \Omega(\boldsymbol{\theta}) = 2\mathbf{W}^{(k)}$$

- Only applies to weights, not biases (weigh decay)

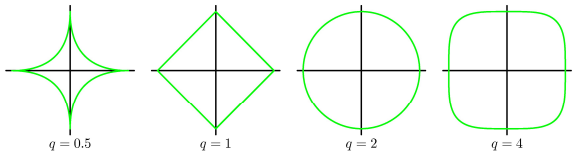
- Can be interpreted as having a Gaussian prior over the weights, while performing MAP estimation.

- We will later look at Bayesian methods.

Other Regularizers

• Using a more general regularizer, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|w_j|^q$$



Lasso Quadratic

L^1 Regularization

• L^1 regularization:

$$\Omega(\boldsymbol{\theta}) = \sum_k \sum_i \sum_j |W_{i,j}^{(k)}|$$

• Gradient:

$$\nabla_{\mathbf{W}^{(k)}}\Omega(\boldsymbol{\theta}) = \mathrm{sign}(\mathbf{W}^{(k)})$$

$$\mathrm{sign}(\mathbf{W}^{(k)})_{i,j} = 1_{\mathbf{W}_{i,j}^{(k)}>0} - 1_{\mathbf{W}_{i,j}^{(k)}<0}$$

- Only applies to weights, not biases (weigh decay)
- Can be interpreted as having a Laplace prior over the weights, while performing MAP estimation.
- Unlike L2, L1 will push some weights to be exactly 0.

Initialization

• Initialize biases to 0

• For weights

- Can not initialize weights to 0 with tanh activation
  - All gradients would be zero (saddle point)
- Can not initialize all weights to the same value
  - All hidden units in a layer will always behave the same
  - Need to break symmetry
- Sample $\mathbf{W}_{i,j}^{(k)}$ from $U[-b, b]$, where

$$b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}}$$

Sample around 0 and break symmetry

Size of $\mathbf{h}^{(k)}(\mathbf{x})$

Model Selection

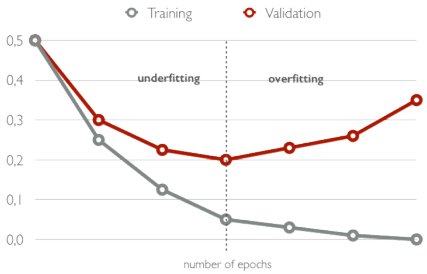
• Training Protocol:

- Train your model on the Training Set $\mathcal{D}^{\mathrm{train}}$
- For model selection, use Validation Set $\mathcal{D}^{\mathrm{valid}}$
  - Hyper-parameter search: hidden layer size, learning rate, number of iterations/epochs, etc.
- Estimate generalization performance using the Test Set $\mathcal{D}^{\mathrm{test}}$

• Remember: Generalization is the behavior of the model on **unseen examples**.

Early Stopping

• To select the number of epochs, stop training when validation set error increases (with some look ahead).



Tricks of the Trade:

• Normalizing your (real-valued) data:

- for each dimension x_i subtract its training set mean
- divide each dimension x_i by its training set standard deviation
- this can speed up training

• Decreasing the learning rate: As we get closer to the optimum, take smaller update steps:

i. start with large learning rate (e.g. 0.1)
ii. maintain until validation error stops improving
iii. divide learning rate by 2 and go back to (ii)

Gradient Checking

• To debug your implementation of fprop/bprop, you can compare with a finite-difference approximation of the gradient:

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$$

- $f(x)$ would be the loss
- x would be a parameter
- $f(x+\epsilon)$ would be the loss if you add ϵ to the parameter
- $f(x-\epsilon)$ would be the loss if you subtract ϵ to the parameter

Debugging on Small Dataset

• If not, investigate the following situations:

- Are some of the units saturated, even before the first update?
  - scale down the initialization of your parameters for these units
  - properly normalize the inputs
- Is the training error bouncing up and down?
  - decrease the learning rate

• This does not mean that you have computed gradients correctly:

- You could still overfit with some of the gradients being wrong