Chapter 1

Introduction to data-driven modeling, motivating examples and course outline

(tentative) Notation:

- *Lower case letters: event realizations, column vectors (sometime) may use an underscore (e.g., w, x, y, z, f, p). Row vectors will be denoted as w^T, functions, probability distribution.*
- *Lower case letters: random variables, matrices, some functions (cdf's) e.g., X, Y, Z, F.*
- *Caligraphic upper case: sets, operators (e.g., $\mathcal{F}$, $\mathcal{L}$, $\mathcal{J}$)*

Note on expectations: $\mathbb{E}[f(x)] := \mathbb{E}_{x\sim p(x)}[f(x)]$.

1.1 Probability primer

- frequentist (UQ based on long run frequencies).
- Bayesian (pure UQ based on probabilities).

Probability space $(\Omega, \mathcal{F}, p)$ where Ω is the sample space, $\mathcal{F}$ is the set of events from σ algebra and p is the probability measure. Measure theory underpins the theoretical foundations and developments of probability theory.

1.2 Discrete random variables (events that take discrete values)

$P(A)$ denotes the probability that the event A is true. By definition $0 \leq P(A) \leq 1$, i.e. $P(A) = 0$ implies the event will definitely not happen, $P(A) = 1$ implies the event will definitely happen.

$$P : \mathcal{F} \to [0, 1] \tag{1.1}$$
$$X : \Omega \to \mathbb{R} \tag{1.2}$$

$P(\bar{A})$ denotes probability of the event not A. consequently, $P(\bar{A}) = 1 - P(A)$.
Discrete random variable X is an event taking values from a finite or countably infinite discrete set $\mathcal{X}$.
We denote the probability of $X = x$ as $P(X = x)$, or simply $p(x)$ (more formally $p(\{w : X(w) = k\})$).
This satisfies $0 \leq P(A) \leq 1$, and $\sum_{x\in\mathcal{X}} p(x) = 1$, and $p(\{\emptyset\}) = 0$.
Here $p(\cdot)$ is called a probability mass function.

Fundamental rules of probability

- Union: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ (A or B) implies $P(A \cup B) \leq P(A) + P(B)$.

- Joint: $P(A,B) = P(A \cap B) = P(A|B)P(B)$, this is also called the product rule, and it directly follows from the definition of conditional probability.
- Conditional probability: $P(A|B) = \frac{P(A \cap B)}{P(B)}$, $P(B) > 0$.

Given a joint distribution $P(A,B)$, we can define the marginal distribution as:

$$P(A) = \sum_{b \in B} p(A,B) = \sum_{b \in B} P(A|B)P(B) = \sum_{b} P(A|B=b)P(B=b) \tag{1.3}$$

This is called the sum rule, or the rule of total probability.

Bayes rule

It is a product of combining the definition of conditional probability with the product and sum rules:

$$P(X=x|Y=y) = \frac{P(X=x \cap Y=y)}{P(Y=y)} \tag{1.4}$$

$$= \frac{P(X=x|Y=y)p(Y=y)}{\sum_{x'} p(X=x')P(Y=y|X=x')} \tag{1.5}$$

Law of total probability

If $A_1, ..., A_k$ are a set of disjoint events (i.e., $A_i \cap A_j = \emptyset, \forall i \neq j$) such that $\cup_{i=1}^{K} A_i = \Omega$ then $\sum_{i=1}^{K} P(A_i) = 1$.

Independence

- X, Y are (unconditionally) independent if: $X \perp Y$ implies $P(X,Y) = P(X)P(Y)$.
- X, Y are conditionally independent if: $X \perp Y|Z$ implies $P(X,Y|Z) = P(X|Z)P(Y|Z)$.

1.3 Continuous Random Variables (events that take continuous values)

Define the events $A = (X \leq a)$, $B(X \leq b)$, and $W = (a < X \leq b)$.
Then observe that $B = A \cup W$, $A \cap W = \{\emptyset\}$, therefore:

$$P(B) = P(A) + P(W) - P(A \cap W) = P(A) + P(W) \tag{1.6}$$

so that

$$P(W) = P(B) - P(A) \tag{1.7}$$

Now let us define the function $F(q) := P(X \leq q)$, this is called the cumulative distribution, or cdf of X.
It is a monotonically non-decreasing function. Then $P(W) = P(a < X \leq b) = F(b) - F(a)$.

Assuming that the cdf is differentiable, we can define $f(x) = \frac{d}{dx}F(x)$. This is called the probability density function, or pdf of X.
By definition, $P(a < X \leq b) = \int_a^b f(x)dx$.
As the size of the interval decreases, we can write $P(x \leq X < x + dx) = P(x)dx$.
We require $P(x) > 0$ but it is possible that $P(x) > 1$, for any given x, as long as the density integrates to 1, i.e., $\int_{-\infty}^{\infty} f(x)dx = 1$.

- $f(x) \leq 0$
- $\int_{-\infty}^{\infty} f(x)dx = 1$
- $\int_{x \in A} f(x)dx = P(x \in A)$

examples: Uniform distribution, Gaussian distribution.

Quantiles

If the cdf is an monotonically increasing function, it has an inverse F^{-1}. Given, if F is the cdf of X, then $F^{-1}(\alpha)$ is the value of x_α such that $P(X \leq x_\alpha) = \alpha$. This probability is called the $\alpha-$quantiles of X.
The value $F^{-1}(0.5)$ is the median of the distribution, with half of the probability mass on the left and half on the right.
We can also use the inverse cdf to compute tail area probabilities.
examples: the Gaussian distribution.

Mean and Variance (quantifying the properties of a probability distribution or a random variable)

Mean, or expected value: (1st-order moment)

- Discrete random variables: $\mu = \mathbb{E}[x] := \sum_{x \in \mathcal{X}} xP(x)$.
- Continuous random variables: $\mu = \mathbb{E}[x] := \int_x xP(x)dx$.

with respect to the probability measure of:

- $\mathbb{E}[\alpha] = \alpha$ for any constant $\alpha \in \mathbb{R}$.
- $\mathbb{E}[\alpha f(x)] = \alpha \mathbb{E}[f(x)]$.
- $\mathbb{E}[f(x) + g(x)] = \mathbb{E}[f(x)] + \mathbb{E}[g(x)]$.

Variance: measures the spread of the distribution, denoted by σ^2. (2nd-order moment)

$$\sigma^2 = Var[x] := \mathbb{E}[(x-\mu)^2] = \int_{\mathcal{X}} (x-\mu)^2 p(x)dx \tag{1.8}$$

$$= \int_{\mathcal{X}} x^2 p(x)dx + \mu^2 \int_{\mathcal{X}} p(x)dx - 2\mu \int_{\mathcal{X}} xp(x)dx \tag{1.9}$$

$$= \mathbb{E}[x^2] - \mu^2 \tag{1.10}$$

Therefore: $\mathbb{E}[x^2] = \mu^2 + \sigma^2$. We can also define the standard deviation as: $std[x] := \sqrt{Var[x]} = \sqrt{\sigma^2}$.

Some common discrete distributions

Binomial

$X \sim Bin(n, \theta)$, e.g., toss a coin n times. If the probability of "heads" is θ then is a binomial random variable. This generalizes to the multi-nomial distribution for cases with non-binary outcomes, e.g., toss of a die.
pmf: $Bin(k|n, \theta) = binom(n, k)\theta^k(1-\theta)^{n-k}$ where $binom(n, k) = \frac{n!}{(n-k)!k!}$.
$\mathbb{E}[x] = n\theta$ and $Var[x] = n\theta(1-\theta)$.

Bernoulli

Suppose we only toss the coin once. Then $X \in \{0, 1\}$ and if the probability of success if θ, the X is a Bernoulli random variable:

$$X \sim Ber(\theta) \tag{1.11}$$

pmf: $Ber(x|\theta) = \theta^{\mathbf{1}_{x=1}}(1-\theta)^{\mathbf{1}_{x=0}}$, where $\mathbb{1}$ is the indicator function, e.g.,

$$\mathbf{1}_{(x=1)} = \begin{cases} 1, & x = 1 \\ 0, & otherwise \end{cases} \tag{1.12}$$

In other words:

$$Ber(x|\theta) = \begin{cases} \theta, & if\ x = 1 \\ 1-\theta, & if\ x = 0 \end{cases} \tag{1.13}$$

This is of course a special case of the Binomial with $n = 1$. Generalizes to the multi-noulli distribution.

Poisson

$X \in \{0, 1, 2, ...\}$ has a Poisson distribution with parameter $\lambda > 0$: $X \sim Poi(\lambda)$.
pmf: $Poi(x|\lambda) = e^{-\lambda}\frac{\lambda^x}{x!}$ where $e^{-\lambda}$ serves as normalization.
This is used to model counts of rare events. (e.g., birth, defects, mutations, car accidents, etc)

Empirical distribution/ Measure

Given a set of data, $\mathcal{D} = \{x_1, ..., x_N\}$, we define the empirical measure as:

$$P_e(A) = \frac{1}{N}\sum_{i=1}^{N}\delta_{x_i}(A), \tag{1.14}$$

where $\delta_x(A)$ is the Dirac measure:

$$\delta_x(A) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \tag{1.15}$$

We can also associate weights with each sample:

$$P(x) = \sum_{i=1}^{N} w_i\delta_{x_i}(x), \quad 0 \leq w_i \leq 1, \quad \sum_{i=1}^{N} w_i = 1 \tag{1.16}$$

We can think of this as a histogram with "spikes" at the data points x_i, where the height y each spike is determined by w_i.

Some common continuous distributions

Gaussian/normal distribution

$X \sim (N)(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ where $\frac{1}{\sqrt{2\pi\sigma^2}}$ serves as normalizing constant. And $\mu = \mathbb{E}[x]$ is the mean (and mode), $\sigma^2 = Var[x]$ is the variance. We also note that μ, σ^2 are sufficient statistics.
Notation: $X \sim \mathcal{N}(\mu, \sigma^2) \Rightarrow p(X = x) = \mathcal{N}(x|\mu, \sigma^2)$.
The special case of $\mu = 0, \sigma^2 = 1$ is referred to as the standard normal. The inverse variance is often called the precision: $\lambda = \frac{1}{\sigma^2}$.
The cdf is defined as: $\phi(x; \mu, \sigma^2) = \int_{-\infty}^{x}\mathcal{N}(z|\mu, \sigma^2)dz$ and can be computed in terms of the error function: $\phi(x; \mu, \sigma) = \frac{1}{2}[1 + erf(\frac{z}{\sqrt{2}})]$ where $z = \frac{x-\mu}{\sigma}$ and $erf(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$.

Why is the Gaussian popular?

- Easy to compute with and interpret (its two parameters are the mean, and variance!)
- The central limit theorem states that sums of independent random variables have an approximately Gaussian distribution, hence making it a good model for residual errors /noise.
- Makes very few assumptions (= has maximum entropy).

- Easy to implement.
- Degenerate pdf: $\lim_{\sigma^2 \to 0} \mathcal{N}(x|\mu, \sigma^2) = \delta(x - \mu)$, where $\delta(\cdot)$ is the Dirac delta function:

$$\delta_x(x) = \begin{cases} \infty, & if\ x = 0 \\ 0, & if\ x \neq 0 \end{cases} \tag{1.17}$$

and $\int_{-\infty}^{\infty} \delta(x)dx = 1$.

Shifting property: $\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(\mu)$, since the integrand is only non-zero.

Student-t distribution

$$\mathcal{T}(x|\mu, \sigma^2, \upsilon) \propto [1 + \frac{1}{\upsilon}(\frac{x-\mu}{\sigma})^2]^{\frac{\upsilon+1}{2}} \tag{1.18}$$

mean $= \mu$, mode $= \mu$, var = $\frac{\upsilon\sigma^2}{\upsilon-2}$ $\rightarrow$ only defined if $\upsilon > 2$.

Heavy tails, robustness to outliers for small υ compared to the Gaussian.
For $\upsilon = 1$, it is often called the Cauchy distribution. A common choice is $\upsilon = 4$. For $\upsilon \gg 5$ it converges to the Gaussian.

Laplace distribution

$$Lap(x|\mu, b) = \frac{1}{2b} \exp(-\frac{|x-\mu|}{b}) \tag{1.19}$$

μ is a location parameter and b is a scale parameter.
mean $= \mu$, mode $= \mu$, var = $2b^2$.
Heavy tails, robustness, puts more mass at zero compared to the Gaussian/student-t.
Last property is useful in cases where one wants to encourage sparsity in their model!
More cases: Gamma, Beta, Chi-squared, Pareto.

1.4 Covariance and correlation

Joint distributions $p(x_1, ..., x_D)$

The covariance between two random variables X and Y measures the degree to which X and Y are linearly related:

$$cov[x, y] = \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] = \mathbb{E}[xy] - \mathbb{E}[x]\mathbb{E}[y] \tag{1.20}$$

If $\underline{x}$ is a d-dimensional random vector, then its covariance matrix is a symmetric, positive definite matrix definted as:

$$cov[\underline{x}] = cov[\underline{x}, \underline{x}] = \mathbb{E}[(\underline{x} - \mathbb{E}[\underline{x}])(\underline{x} - \mathbb{E}[\underline{x}])^T] = \begin{bmatrix} var[x_1] & cov[x_1, x_2] & \dots & cov[x_1, x_d] \\ cov[x_2, x_1] & var[x_2] & \dots & cov[x_2, x_d] \\ \vdots & \vdots & & \vdots \\ cov[x_d, x_1] & cov[x_d, x_2] & \dots & var[x_d] \end{bmatrix} = \Sigma = \Lambda^{-1} \tag{1.21}$$

where Σ is the covariance matrix and Λ is the precision matrix. Covariances can take values between zero and infinity.
Sometimes it is more preferable to work with a normalized measure with a finite upper bound.

Pearson correlation coefficient

$$corr[x,y] = \frac{cov[x,y]}{\sqrt{var[x]var[y]}}, \qquad -1 \leq corr[x,y] \leq 1. \tag{1.22}$$

Specifically, one can show that $corr[x,y] = 1$ iff $Y = aX + b$ for some a, b.
If X and Y are independent: $corr[x,y] = 0$.

The multi-variate Gaussian

The most widely used joint probability density function for continuous random variables.
pdf in D dimensions:

$$\underset{\sim}{x} = (x_1, ..., x_D) \sim \mathcal{N}(\underset{\sim}{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp[-\frac{1}{2}(\underset{\sim}{x} - \mu)^T \Sigma (\underset{\sim}{x} - \mu)] \tag{1.23}$$

where $\frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}}$ serves as normalizing constant and $\mu = \mathbb{E}[\underset{\sim}{x}] \in \mathbb{R}^D$, $\Sigma = cov[\underset{\sim}{x}, \underset{\sim}{x}]$ is a $D \times D$ covariance matrix. $\Lambda = \Sigma^{-1}$ is the precision matrix.

1.5 Transformations of random variables

$x \sim p(\cdot)$, $y = f(x)$, $p(y)$?

Linear transformations

$$y = f(x) = Ax + b \Rightarrow \mathbb{E}[y] = \mathbb{E}[Ax + b] = A\mu + b, \tag{1.24}$$

where $\mu = \mathbb{E}[x]$ due to the linearity of expectation.
If $y = a^T \underset{\sim}{x} + b$, then $\mathbb{E}[y] = a^T\mu + b$. Also, $cov[y] = cov[Ax + b] = A^T \Sigma A$, where $\Sigma = cov[x]$.
If $f(\cdot)$ is scalar valued, this becomes: $cov[y] = var[a^T \underset{\sim}{x} + b] = a^T \Sigma a$.

General transformations

- Discrete random variables: If X is a discrete random variable, then we can derive the pmf of y. $p_y(y) = \sum_{x:f(x)=y} p_x(x)$
- Continuous random variables: We can no longer use the above since we sum up densities. In stead, we work with cdf's

$$F_y(y) = p(Y \leq y) = p(f(x) \leq y) = p(x \in \{x|f(x) \leq y\}) \tag{1.25}$$

Then we can obtain the pdf (of y) by differentiating the cdf:

$$F_y(y) = p(f(x) \leq y) = p(x \leq f^{-1}(y)) = F_x(f^{-1}(y)) \tag{1.26}$$

Taking derivatives this gives:

$$p_y(y) = \frac{d}{dy}F_y(y) = \frac{d}{dy}F_x(f^{-1}(y)) = \frac{dx}{dy}\frac{d}{dx}F_x(x) = \frac{dx}{dy}p_x(x), \tag{1.27}$$

where $x = f^{-1}(y)$.
In general: $p_y(y) = p_x(x)|\frac{dx}{dy}|$.

This can be extended to the multi-variate case: $p_y(\underset{\sim}{y}) = p_x(\underset{\sim}{x})|det\frac{\partial \underset{\sim}{x}}{\partial \underset{\sim}{y}}|$ Jacobian of the inverse mapping.

1.6 Central limit theorem

Consider N random variables with pdfs $p(x_i)$ (not necessarily Gaussian!), each having mean μ and variance σ^2. We assume that each variable is independent and identically distributed, i.e., iid. Let $S_N = \sum_{i=1}^N x_i$ be the sum of the random variables. Then,

$$p(S_N = s) = \frac{1}{\sqrt{2\pi N\sigma^2}} \exp(-\frac{(s-N\mu)^2}{2N\sigma^2}) \quad as \quad N \to \infty. \tag{1.28}$$

Hence, the distribution of $Z_N := \frac{S_N - N\mu}{\sigma\sqrt{N}} = \frac{\bar{x}-\mu}{\sigma/\sqrt{N}}$ converges to a standard normal. here $\bar{x} := \frac{1}{N}\sum_{i=1}^N x_i$ is the sample mean.

1.7 Monte Carlo approximation

In general, computing the distribution of a function of random variables can be difficult using the change of variables formula. A powerful alternative is sampling.
We first generate S samples $x_1, ..., x_s$ out of $p_x(\cdot)$ (either iid, or correlated e.g., mcmc). Then we can approximate $p_y(\cdot)$ using the empirical distribution of $\{f(x_s)\}_{s=1}^{s^*}$.

Computing expectations using the Monte Carlo approximation

$$\mathbb{E}_{x\sim p(x)}[f(x)] = \mathbb{E}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{S}\sum_{s=1}^{S} f(x_s) \tag{1.29}$$

By changing the function $f(\cdot)$ we can compute different quantities of interest:

- $\bar{x} = \frac{1}{S}\sum_{s=1}^S x_s \approx \mathbb{E}[x]$.
- $\frac{1}{S}\sum_{s=1}^S (x_s - \bar{x})^2 \approx Var[x]$.
- $\frac{1}{S}|\{x_s \le c\}| \to p(x \le c)$, median $\{x_1, ..., x_s\} \to median(x)$
- Entropy and relative entropy: $x \sim p(\cdot)$, $y \sim q(\cdot)$
  $\mathbb{H}[X] = \int p(x)\log p(x)dx$, $\mathbb{H}[X|Y] = -\int_{-\infty}^{\infty} p(x)\log\frac{p(x)}{q(x)}dx = \mathbb{KL}[p||q]$ is the Kullback-Leibler divergence.

Mutual information: $I(X,Y) = \int_{\mathcal{X}}\int_{\mathcal{Y}} p_{(X,Y)}(x,y)\log\frac{p_{(X,Y)}(x,y)}{p_X(x)p_Y(y)}dxdy$ (= 0 if X and Y are independent).

Maximum Likelihood Estimation (MLE)

Important technique for estimating parameters of a distribution / model.

Setup

Given some data $\mathcal{D} := \{x_1, ..., x_N\}, x_i \in \mathbb{R}^d$.
Assume a family of distributions $p_\theta(x), \theta \in \Theta$.
Assume a probabilistic model for the data, i.e.,

$$x \overset{\text{i.i.d}}{\sim} p_\theta(x)$$

for some θ.

Goal

Estimating the true value of θ that best explains the observed data.

Definition

θ_{MLE} is a maximum likelihood estimate if $\theta_{MLE} = \arg\max_{\theta\in\Theta} p(\mathcal{D}|\theta)$, where $p(\mathcal{D}|\theta)$ is the likelihood. More precisely, $p(\mathcal{D}|\theta_{MLE}) = \max_{\theta\in\Theta} p(\mathcal{D}|\theta)$.

$$p(\mathcal{D}|\theta) = p(x_1, ..., x_N|\theta) = \prod_{i=1}^{N} p(x_i|\theta) = \prod_{i=1}^{N} p(X = x_i|\theta)$$

Remarks

- 1. An MLE might not be unique.
- 2. The MLE may fail to exist (the maximum may not be achieved for a $\theta \in \Theta$).

Pros

- Usually it is easy to compute and often is interpretable (e.g. the mean of an random variable is the sample mean).
- Nice asymptotic properties:
  - consistent (as $N \to \infty$ converges to the true θ in probability).
  - asymptotically normal (as $N \to \infty$ its distribution converges to a normal).
  - Efficient (i.e. it has the lowest asymptotic variance).
  - Invariant under re-parametrization, i.e. $\forall g : g(\theta_{MLE})$ is an MLE for $g(\theta)$.

Cons

- It is a point estimate (no representation of uncertainty).
  Ideally we'd like to compute the posterior over θ: $p(\theta|\mathcal{D}) \propto p(\mathcal{D}|\theta)p(\theta)$ (Bayesian).
- May run into cases that are wired (lack of robustness).
- Over-fitting.
- Existence and uniqueness is not guaranteed.

MLE for a univariate Gaussian

Setup

Given $\mathcal{D} := \{x_1, ..., x_N\}, x_i \in \mathbb{R}$.

Assumption

$$x \overset{\text{i.i.d}}{\sim} p_\theta(x) = \mathcal{N}(\mu, \sigma^2), \quad \theta := \{\mu, \sigma^2\}$$

.

Likelihood

$$p(\mathcal{D}|\theta) = p(x_1, ..., x_N|\theta) = \prod_{i=1}^{N} p(x_i|\theta) \tag{1.30}$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{1}{2\sigma^2}(x_i - \mu)^2) = (\frac{1}{\sqrt{2\pi\sigma^2}})^N \exp(-\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i - \mu)^2) \tag{1.31}$$

$$\theta_{MLE} = \arg\max p(\mathcal{D}|\theta) = \arg\max \log p(\mathcal{D}|\theta) \tag{1.32}$$

$$= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{1.33}$$

Taking gradients and set to zero:

$$0 = \frac{\partial}{\partial\mu}\log p(\mathcal{D}|\theta) \Rightarrow \frac{1}{2\sigma^2}\sum_{i=1}^{N} 2(x_i - \mu) = 0 \Rightarrow \sum_{i=1}^{N} x_i - N\mu = 0 \tag{1.34}$$

$$\Rightarrow \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{1.35}$$

Is this critical point a maximum?

$$\frac{\partial^2}{\partial\mu^2}\log p(\mathcal{D}|\theta) = -\frac{N}{\sigma^2} < 0,$$

hence μ_{MLE} is a global and unique maximizer.

Chapter 2

Probability theory primer through the lens of Bayesian linear regression

2.1 Statistical comparisons

To compare two numbers we can look at their difference or their ratio. The same can be done if we want to compare two probability densities! Either therough a density difference, or a density ratio. Our focus here will be on density ratios since they are ubiquitous in machine learning.

$$r(x) := \frac{p(x)}{q(x)} \tag{2.1}$$

is the density ratio between two probability densities p, q.

Intuitively this tells us "how much" do we need to correct $q(x)$ for it to match $p(x)$ i.e., $p(x) = r(x)q(x)$ where $r(x)$ is the correction factor.
This shows up a lot!...For example:

- (1) Bayes theorem: $p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}$.
- (2) Divergences and maximum likelihood:
  $\mathbb{KL}[p(x)||q(x)] = \int p(x) \log \frac{p(x)}{q(x)} dx \rightarrow$ maximum likelihood estimation is equivalent to minimizing this divergence.
- (3) Importance sampling: $\int p(y|x)p(x)dx = \int p(y|x)q(x)\frac{p(x)}{q(x)}dx = \mathbb{E}_{x\sim q(x)}[p(y|x)r(x)]$.
- (4) Mutual information: $I[X,Y] = \int\int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} dxdy = \mathbb{KL}[p(x,y)||p(x)p(y)]$ where $p(x,y) = p(x|y)p(y)$, so we have $I[X,Y] = \int\int p(x,y) \log \frac{p(x|y)}{p(x)} dxdy = \int \mathbb{KL}[p(x|y)||p(x)]p(y)dy = \mathbb{E}_{y\sim p(y)}[\mathbb{KL}[p(x|y)||p(x)]]$.

2.2 Linear regression

- It is the workhorse of statistics.
- It is not just about lines and planes!

Setup: Given $\mathcal{D} = ((x_1, y_1), ..., (x_n, y_n))$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.
Goal: Choose $f : \mathbb{R}^d \rightarrow \mathbb{R}$ to predict the value of y for a new x.
Remark: Using basis functions we can represent nonlinear functions. E.g., $f(x) = w^T x$, $w \in \mathbb{R}^d \Rightarrow f(x) = \sum_{j=1}^d w_j x_j$ or $f(x) = w^T\varphi(x) = w^T z$, $z = \varphi(x)$, $\varphi : \mathbb{R}^d \rightarrow \mathbb{R}^m \Rightarrow f(x) = \sum_{j=1}^m w_j \varphi_j(x)$ where $\varphi(x) = (\varphi_1(x), ..., \varphi_m(x))$ are the features or basis function e.g., polynomials, Fourier, wavelet, radial basis functions, etc.

Linear means that the model is linear in the paramters w!

Time to make some choices and define our model:
We choose a discriminative (vs generative) model approach to model $p(y|x)$.
First assume a family $p_\theta(y|x)$, parameterized by $\theta \in \Theta$, and estimate θ using $\mathcal{D}$.
Which family to use? Our first choice would by a Gaussian!
Hence: $p_\theta(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))$, $\theta = (\mu, \sigma^2)$ (functions on $\mathbb{R}^d$). Note that this notation is just shorthand notation for convenience.
What $\mu(x)$ and $\sigma^2(x)$ to use?
Gaussian linear regression models the data $\mathcal{D}$ by assuming:

$$p_\theta(y|x) = \mathcal{N}(y|w^T x, \sigma^2), \quad i.e., \mu(x) = w^T x, \sigma^2(x) = \sigma^2, \tag{2.2}$$

$$\theta = \{w, \sigma^2\}, w \in \mathbb{R}^d, \sigma^2 > 0 \tag{2.3}$$

Alternatively, one can think of this model as:

$$y = w^T x + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2), \quad i.e., \text{the noise is modelled as a Gaussian random variable.} \tag{2.4}$$

Also, one can make other choices, e.g., Laplace distribution for robust regression.

Estimate θ Using Maximum Likelihood Estimation (MLE)

Setup: Given $\mathcal{D} = ((x_1, y_1), ..., (x_n, y_n))$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $\theta = (\mu, \sigma^2)$ (we will assume that σ^2 is known).
Model: We assume that $y_1, ..., y_n$ are random independent and identically distributed variables as modelled by $y_i \sim \mathcal{N}(w^T x_i, \sigma^2)$.
Then $\theta_{MLE} = \arg\max_{\theta \in \Theta} p(\mathcal{D}|\theta)$ where $p(\mathcal{D}|\theta)$ is the likelihood.
Now we need to write down our likelihood in terms of the model. (note that only the y's are random, the X's are deterministic).

$$p(\mathcal{D}|\theta) = p(y_1, ...y_n|x_1, ..., x_n, \theta) \tag{2.5}$$

$$= \prod_{i=1}^n p(y_i|x_i, \theta) = \prod_{i=1}^n \mathcal{N}(w^T x_i, \sigma^2) \tag{2.6}$$

$$= \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp[-\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - w^T x_i)^2] \tag{2.7}$$

$$\Rightarrow p(\mathcal{D}|\theta) = (\frac{1}{\sqrt{2\pi\sigma^2}})^n \exp[-\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw)] \tag{2.8}$$

Here we should note that we can write $\sum_{i=1}^n (y_i - w^T x_i)^2 = (y - Xw)^T(y - Xw) = \|y - Xw\|$ where X is the $(n \times d)$ design matrix.

$$\Rightarrow -\log p(\mathcal{D}|\theta)) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y - Xw)^T(y - Xw) := \mathcal{L}(w) \tag{2.9}$$

$$(y - Xw)^T(y - Xw) = \frac{1}{2}(y^T y - y^T Xw - (Xw)^T y + (Xw)^T(Xw)) = \frac{1}{2}y^T y - y^T Xw + wX^T Xw. \tag{2.10}$$

Hence:

$$\nabla_w \mathcal{L}(w) = -X^T y + X^T Xw = -X^T y + X^T Xw. \tag{2.11}$$

Setting to zero yields: $\nabla_w \mathcal{L}(w) = 0 \Rightarrow w_{MLE} = (X^T X)^{-1} X^T y$. where $(X^T X)$ needs to be invertible. See discussion below:
$X^T X$ is invertible if the columns of X are linearly independent.

$(X^TX)^{-1}X^T := X^\dagger$ is called Moore-Penrose pseudo-inverse.
To check whether this is actually a minimum we can examine the Hessian:
$H = \nabla_w^2 \mathcal{L}(w) = X^TX \rightarrow$ this is symmetric positive semi-definite. Assuming that X^TX is invertible then $\mathcal{L}$ is strictly convex in w and w_{MLE} is a unique minimizer.
Geometric interpretation as projection on the feature space.

Basis functions MLE:

$$w_{MLE} = (\phi^T\phi)^{-1}\phi^T y, \qquad \varphi : \mathbb{R}^d \rightarrow \mathbb{R}. \tag{2.12}$$

$$\text{where} \quad \phi = [\varphi_1(x), ..., \varphi_m(x)] \in \mathbb{R}^{(n\times m)}. \tag{2.13}$$

2.3 Bayesian Linear regression

Why Bayesian linear regression?

- MLE is prone to overfitting?
- Placing a prior on w and using MAP will fix the overfitting issue, but we also want some representation of uncertainty, i.e., have access to the predictive distribution $p(y^*|x^*, \mathcal{D})$.

Setup: Given $\mathcal{D} = ((x_1, y_1), ..., (x_n, y_n))$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$, $\theta = (\mu, \sigma^2)$ (we will assume that σ^2 is known).
Model: $y_1, ..., y_n$ are iid given w, $y_i \sim \mathcal{N}(w^Tx_i, a^{-1})$, $a = \frac{1}{\sigma^2}$ is called the precision. For on $w \sim \mathcal{N}(0, b^{-1}I)$, a multi-variate (independent since the covariance is diagonal) Gaussian prior on $w \in \mathbb{R}^d : w = (w_1, ..., w_d)$ with $a, b > 0$.
We will also assume that a, b are known, hence the model parameters are just the weights $\theta = \{w\}$.
Remark: we can also use basis functions to model non-linearities.
Likelihood:

$$p(\mathcal{D}|\theta) \propto \exp[-\frac{a}{2}(y - Xw)^T(y - Xw)] \tag{2.14}$$

with $y = (y_1, ..., y_n)$ and $X = \begin{bmatrix} x_{11} & \dots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nd} \end{bmatrix}$.
Posterior (according to the Bayes rule):

$$p(\mathcal{D}|\theta) \propto p(\mathcal{D}|w)p(w) \tag{2.15}$$

$$p(\mathcal{D}|\theta) \propto \exp[-\frac{a}{2}(y - Xw)^T(y - Xw) - \frac{b}{2}w^Tw]. \tag{2.16}$$

Notice that the exponent is a quadratic in w. This hints that the posterior is Gaussian.
We can actually derive this by "completing the square".
To see this, let us re-write:

$$a(y - Xw)^T(y - Xw) - bw^Tw = a(y^Ty - 2w^TX^Ty + w^TX^TXw) + bw^Tw \tag{2.17}$$

$$ay^Ty - 2aw^TX^Ty + w^T(aX^TX + bI)w \tag{2.18}$$

We can make this look like the exponent of a Gaussian by noticing that:
In general, $(x - \mu)^T\Lambda(x - \mu) = x^T\Lambda x - 2x^T\Lambda\mu + \mu^T\Lambda\mu$ where the last term is constant.
To match terms, let $\Lambda := aX^TX + bI$ (precision matrix).

We also want $aw^T X^T y = w^T \Lambda \mu$ or $aX^T y = \Lambda \mu$, hence let $\mu := a\Lambda^{-1} X^T y$.
Therefore, the posterior is Gaussian!

$$p(w|\mathcal{D}) = \mathcal{N}(w|\mu, \Lambda^{-1}), \qquad \begin{cases} \Lambda = aX^T X + bI, \\ \mu = a\Lambda^{-1} X^T y \end{cases} \tag{2.19}$$

MAP estimate: $w_{MAP} = \arg\max p(w|\mathcal{D})$, since $p(w|\mathcal{D})$ is Gaussian then the MAP estimate is its mode, e.g.,

$$w_{MAP} = \mu = a(aX^T X + bI)^{-1} X^T y = (X^T X + \frac{b}{a} I)^{-1} X^T y \tag{2.20}$$

compare this to w_{MLE} to see the effect of regularization! Here $\frac{b}{a} I$ term serves as regularization.
Also notice that maximizing $p(w|\mathcal{D})$ is equivalent to minimizing $\|y - Xw\|_2^2 + \lambda \|w\|_2^2$, with $\lambda = \frac{b}{a}$.

2.4 Predictive distribution for Bayesian linear regression

Given a new test point x^*, we want to predict the corresponding $y^* : p(y^*|x^*, \mathcal{D})$.
Recall that by our construction $y^* \sim \mathcal{N}(w^T x^*, a^{-1})$ is independent of $y_1, ..., y_n$.
In general, the predictive distribution is computed by marginalizing out any uncertain variables or parameters.
Here, this translates to:

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, \mathcal{D}, w) p(w|x^*, \mathcal{D}) dw \tag{2.21}$$

$$= \int \mathcal{N}(y^*|w^T x^*, a^{-1}) \mathcal{N}(w|\mu, \Lambda^{-1}) dw \tag{2.22}$$

$$\propto \int \exp[-\frac{a}{2}(y - Xw)^T (y - Xw)] \exp[-\frac{1}{2}(w - \mu)^T \Lambda (w - \mu)] dw \tag{2.23}$$

Note that conditioning on x is simply to enhance clear notation. y does not depends on x in a statistical since, since x is deterministic.

Our goal is to bring this integral in the form $\int \mathcal{N}(w|\dots) g(y^*) dw = g(y^*) \propto \mathcal{N}(y^*|\dots)$.
To do this, we will employ the usual trick of "completing the square".
After some lengthy derivation we can arrive to the final results:

$$p(y^*|x^*, \mathcal{D}) = \mathcal{N}(y^*|u, \frac{1}{\lambda}), \qquad \begin{cases} u = \mu^T x^*, \ \mu = w_{MAP} = (X^T X + \lambda I)^{-1} X^T y, \\ \frac{1}{\lambda} = \frac{1}{a} + x^{*T} \Lambda^{-1} x^*, \ \Lambda = aX^T X + bI. \end{cases} \tag{2.24}$$

Chapter 3

Optimization: gradients and Hessians, gradient descent, Newton's algorithm, stochastic gradient descent

3.1 Optimization

Setup: Given a model with parameters $\theta = (\theta_1, \theta_2, ..., \theta_l)$ and a loss function $J(\theta)$ our goal is to identify θ^* such that:

$$\theta^* = \arg\min J(\theta) \tag{3.1}$$

We need to identify the critical points for which $\nabla_\theta J(\theta) = 0$. This condition is true for minimum, maximum and saddle points.

Gradients

$$\nabla_\theta f(\theta) = \begin{bmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_d} \end{bmatrix} \tag{3.2}$$

Gradient descent:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta_n) \tag{3.3}$$

where η is the step-size (often called the learning rate). $\theta_{n+1} \in \mathbb{R}^{d\times 1}$, $\theta_n \in \mathbb{R}^{d\times 1}$ and $\nabla_\theta J(\theta_n) \in \mathbb{R}^{d\times 1}$.
This is a first-order method as it relies on a linear approximation of $J(\theta)$ around θ.
Perhaps a higher order approximation of $J(\theta)$ would result in faster convergence? Yes!

Hessian

$$\nabla^2_\theta f(\theta) = \begin{bmatrix} \frac{\partial^2 J(\theta)}{\partial \theta_1^2} & \frac{\partial^2 J(\theta)}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_1 \partial \theta_d} \\ \frac{\partial^2 J(\theta)}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J(\theta)}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_2 \partial \theta_d} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 J(\theta)}{\partial \theta_d \partial \theta_1} & \frac{\partial^2 J(\theta)}{\partial \theta_d \partial \theta_2} & \cdots & \frac{\partial^2 J(\theta)}{\partial \theta_d^2} \end{bmatrix} \text{ is a } (d \times d) \text{ matrix.} \tag{3.4}$$

where $g_n := \nabla_\theta J(\theta_n)$ and $H_n := \nabla^2_\theta J(\theta_n)$.
Let's use a Taylor expansion of $J(\theta)$ around θ_n:

$$J(\theta) \approx J(\theta_n) + g_n^T(\theta - \theta_n) + \frac{1}{2}(\theta - \theta_n)^T H_n(\theta - \theta_n), \tag{3.5}$$

$$= J(\theta) + g_n^T(\theta - \theta_n) + \frac{1}{2}[\theta^T H_n \theta - 2\theta^T H_n \theta_n + \theta_n^T H_n \theta_n] \tag{3.6}$$

$$\Rightarrow \nabla_\theta J(\theta) = 0 + g_n^T + H_n\theta - H_n\theta_n. \tag{3.7}$$

The minimum should satisfy $\nabla_\theta J(\theta) = 0$, hence:

$$\Rightarrow -g_n^T = H_n\theta - H_n\theta_n \Rightarrow \theta - \theta_n = -H_n^{-1} g_n^T \tag{3.8}$$

Newton's algorithm:

$$\theta_{n+1} = \theta_n - H_n^{-1} g_n^T \tag{3.9}$$

Utilizes the geometry better than gradient descent by using curvature information.

Example: Linear regression $p(y|x^T x, \sigma^2)$

Recall the loss function for linear regression:

$$\mathcal{L}(w) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}(y - Xw)^T(y - Xw). \tag{3.10}$$

Then

$$\nabla_w \mathcal{L}(w) = -X^T y + X^T X w \tag{3.11}$$

and

$$\nabla^2_w \mathcal{L}(w) = X^T X. \tag{3.12}$$

Hence, the parameter updates are:

$$\text{Gradient descent:} \quad w_{n+1} = w_n - \eta[X^T X w_n - X^T y] \tag{3.13}$$

$$\text{Newton:} \quad w_{n+1} = w_n - (X^T X)^{-1}[X^T X w_n - X^T y]. \tag{3.14}$$

Remark: similar updates can be made for σ^2! η can be different (also adaptive over iterations!) for every parameter! Can also be rigorously tuned using "line search" methods but is not practical in machine learning applications.

Limitations:

- Gradient descent converges slowly. Choosing η is an art.
- Scalability for big data.
- Exact Hessians are often very hard to compute $\rightarrow$ Quasi-Newton methods.
- Suffers from pathological curvature cases.

3.2 Stochastic gradient descent

In many machine learning applications the loss functions factorize across data points, i.e., they can be written as:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta) \qquad \text{(i.e., see linear regression)}. \tag{3.15}$$

Each term $J_i(\theta)$ is typically associated with the i-th observation/data point.
In this case, a standard "full batch" gradient descent approach would take the form:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta J(\theta_n) = \theta_n - \eta \sum_{i=1}^{n} \nabla_\theta^{(i)}(\theta_n) \tag{3.16}$$

In stochastic gradient descent, the true gradient is approximated by the gradient at a single example:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta^{i}(\theta_n) \tag{3.17}$$

A compromise between the two-extremes is to approximate the gradient over a "mini-batch" of data:

$$\theta_{n+1} = \theta_n - \eta \sum_{i=1}^{m} \nabla_\theta^{i} J(\theta_n) \tag{3.18}$$

A complete looping cycle over the entire data-set is called an "epoch".
Convergence of SGD has been analyzed using theory from convex minimization and stochastic approximation, and it is known that, given an appropriate learning rate and some mild assumption, it converges to the global minimum for convex problems, or a local minimum otherwise.

Gradient descent variants

SGD with momentum

$$u_{n+1} = \gamma u_n + \eta \nabla_\theta J(\theta_n) \tag{3.19}$$
$$\theta_{n+1} = \theta_n - u_{n+1} \tag{3.20}$$

where:

- η is the learning rate.
- $\gamma = 0 \rightarrow$ standard gradient descent.
- $\gamma = 0.9$ is a typical value used in applications.

Intuition: smooths out oscillations at steep valleys.

Nesterov acceleration

$$u_{n+1} = \gamma u_n + \eta \nabla_\theta J(\theta_n - \gamma u_n) \tag{3.21}$$
$$\theta_{n+1} = \theta_n - u_{n+1} \tag{3.22}$$

Intuition: prevent momentum from going too fast.

Adaptive learning rate methods

RMSprop

$\mathbb{E}[g^2]_n :=$ average of the square gradients at iteration n, $g_n := \nabla_\theta J(\theta_n)$.

$$\mathbb{E}[g^2]_{n+1} = \gamma \mathbb{E}[g^2]_n + (1-\gamma) g_n^2, \tag{3.23}$$

$$\theta_{n+1} = \theta_n - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_n + \epsilon}} g_n \tag{3.24}$$

This is standard gradient descent update with adaptive learning rate. Usually $\gamma = 0.9$ and $\eta = 0.001$.

Adam

$$m_{n+1} = \theta_1 m_n + (1-\theta_1) g_n, \tag{3.25}$$

$$v_{n+1} = \theta_2 v_n + (1-\theta_2) g_n^2 \tag{3.26}$$

Estimates of the first moment (mean) and the second moment (variance) of the gradients. Here m_0 and v_0 are usually initialized to zero, and the above estimates are biased towards zero. To counteract this bias, we can consider the correction:

$$\hat{m}_n = \frac{m_n}{1-\theta_1^n}, \tag{3.27}$$

$$\hat{v}_n = \frac{v_n}{1-\theta_2^n}, \tag{3.28}$$

and perform the update:

$$\theta_{n+1} = \theta_n - \frac{\eta}{\sqrt{\hat{v}_n} + \epsilon} \hat{m}_n. \tag{3.29}$$

3.3 Logistic classification (many applications in biostatistics, medicine, social sciences)

Examples: suppose you are on actuary and you want to build a model the probability that someone may die in the next 10 years:

$$p(\text{death}|x), \quad x = (x_1, x_2, x_3) \quad x_1 = \text{age}, \quad x_2 = \text{m/F} \in 0, 1 \quad x_3 = \text{cholestorol}. \tag{3.30}$$

The simplest model would be to consider a linear combination of the input variables:

$$w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 = w^T x, \quad x = (x_1, x_2, x_3) \tag{3.31}$$

but this is not a probability! We can fix that by mapping this with a sigmoid function:

Hence our model is:

$$p(y|x) = \sigma(w^T x), \quad \sigma(a) = \frac{1}{1+e^{-a}} : \text{Logistic function} \tag{3.32}$$

Intuition: This is trying to fit plane that discriminates between the two events {death, not death}.

Formal definition

Setup: Given $\mathcal{D} = ((x_1, y_1), ..., (x_n, y_n))$, $x_i \in \mathbb{R}^d$, $y_i \in \{0, 1\}$.
Model: $y_i \sim Ber(\sigma(w^T x_i))$, y_i are iid.
Pros:

- Interpretable (= the model parameters have an interpretable meaning, e.g. $w_i > 0 \Rightarrow$ the probability of death increases with age.) $\rightarrow$ that is why it is popular in medicine, etc.
- Reveals which variables are moer influential.
- Small number of parameters: just $(d + 1)$. It is a simple model that is statistically easy to train.
- Computationally efficient to estimate w.
- Easy extension to multi-class classification.
- Forms the foundation for more complex models like neural nets and GLM.

Cons:

- Being a simple model, its performance is often inferior to more sophisticated methods.

Maximum likelihood estimation

$$w_{MLE} = \arg\max_w p(\mathcal{D}|w), \quad p(\mathcal{D}|w) = \prod_{i=1}^{n} p(y_i|x_i, w), \tag{3.33}$$

$$\text{Let} \quad a_i = \sigma(w^T x_i), \quad \text{then} \quad p(\mathcal{D}|w) = \prod_{i=1}^{n} a_i^{y_i}(1 - a_i)^{1-y_i}, \tag{3.34}$$

$$\Rightarrow \mathcal{L}(w) = -\log p(\mathcal{D}|w) = -\sum_{i=1}^{n} y_i \log a_i + (1 - y_i) \log(1 - a_i) \quad \rightarrow \text{Binary cross entropy.} \tag{3.35}$$

Iterative reweighted least squares

Recall Newton's method:

$$w_{t+1} = w_t - H^{-1} g, \quad H = X^T A X, \quad g = X^T (a - y) \tag{3.36}$$

$$\Rightarrow w_{t+1} = w_t - (X^T A X)^{-1} X^T A [X w_t - A^{-1}(a - y)] X^T (a - y) \quad \text{assuming that } X^T A X \text{ is invertible.} \tag{3.37}$$

We can re-write this:

$$w_{t+1} = (X^T A X)^{-1} X^T A [X w_t - A^{-1}(a - y)] = (X^T A X)^{-1} X^T A z_t, \tag{3.38}$$

where $z_t = X w_t - A^{-1}(a - y)$.
Recall the solution to linear regression: $w_{MLE} = (X^T X)^{-1} X^T y = (X^T A X)^{-1} X^T A y$.
Then $w_{t+1} = (X^T A X)^{-1} X^T A z_t$ is the solution to a weighted least squares problem where A is a weights matrix.
In practice, instead of inverting $H = X^T A X$, one solves the system $Hu = f$ using CG.
If you run into cases when H is not invertible you can switch back to gradient descent.

Multi-class logistic regression

Model:

$$p(y=c|x,W)=\frac{\exp(w_c^T x)}{\sum_{c'=1}^{c}\exp(w_{c'}x)} \tag{3.39}$$

soft-max function is a generalization of the logistic function for multiple classes. Here w_c is the c-column of W.

Now y is a one-hot-encoding vector $y=\begin{bmatrix}0&0&1&0\\1&0&0&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$ and $y_{ic}:=\mathbf{1}(y_i=c)$.

$$p(\mathcal{D}|w)=\prod_{i=1}^{n}\prod_{c=1}^{c}p(y_i=c|x,w), \tag{3.40}$$

$$\Rightarrow \log p(\mathcal{D}|w)=\sum_{i=1}^{n}[(\sum_{c=1}^{c}y_{ic}w_c^T x_i)-\log(\sum_{c'=1}^{c}\exp(w_{c'}^T x_i)), \tag{3.41}$$

this is the multi-class cross-entropy loss.
Gradient: Recall that $a=\sigma(w^T x)$.

$$\Rightarrow \log a=\log\sigma(w^T x)=\log(\frac{1}{1+e^{-w^T x}})=-\log(1+e^{-w^T x}), \tag{3.42}$$

$$\text{and}\quad \log(1-a)=\log(1-\sigma(w^T x))=\log(1-\frac{1}{1+e^{-w^T x}})=\log(\frac{e^{-w^T x}}{1+e^{-w^T x}})=-w^T x-\log(\frac{1}{1+e^{-w^T x}}) \tag{3.43}$$

$$\frac{\partial}{\partial w}\log a=-\frac{-xe^{-w^T x}}{1+e^{-w^T x}}=x(1-a) \tag{3.44}$$

$$\frac{\partial}{\partial w}\log(1-a)=-x+x(1-a)=-ax. \tag{3.45}$$

Therefore,

$$\nabla_{w_j}\mathcal{L}(w)=-\sum_{i=1}^{n}y_i x_{ij}(1-a_i)-(1-y_i)x_{ij}a_i \tag{3.46}$$

$$=-\sum_{i=1}^{n}y_i x_{ij}-y_i x_{ij}a_i-x_{ij}a_i+y_i x_{ij}a_i \tag{3.47}$$

$$=\sum_{i=1}^{n}(a_i-y_i)x_{ij} \tag{3.48}$$

In matrix vector notation: $\nabla_{w_j}\mathcal{L}(w)=x^T(a-y)$.
Note: we cannot analytically solve for w since in $\nabla_w\mathcal{L}(w)=0$, w shows up in a nonlinear fashion.
Hessian: $\frac{\partial^2}{\partial w_i\partial w_j}\mathcal{L}(w)=\sum_{i=1}^{n}x_{ij}(\frac{\partial}{\partial w_k}a_i)=\sum_{i=1}^{n}x_{ij}x_{ik}a_i(1-a_i)=z_j^T A z_k$. We can write this is because $\sum_{i=1}^{n}x_{ij}x_{ik}a_i(1-a_i)$ is in a quadratic form, where $z_j=(x_{1j,\ldots,x_{nj}})^T\rightarrow$ j-th column of X, the design matrix and $A=\begin{bmatrix}a_1(1-a_1)&\ldots&\ldots&0\\\vdots&\ddots&\ddots&0\\\vdots&\ddots&\ddots&0\\0&\ldots&\ldots&a_n(1-a_n)\end{bmatrix}$.
$\Rightarrow \nabla_w^2\mathcal{L}(w)=X^T AX\rightarrow$ one can show that this is positive-semidefinite and therefore $\mathcal{L}(w)$ is convex.
Applying Newton's algorithm to this problem results in an iterative algorithm called iterative reweighted least squares.

Chapter 4

Deep neural networks: back-propagation, over-fitting and regularization, automatic differentiation

4.1 Neural Networks

Pros:

- Adaptive features /basis functions (parametric). Recent breakthroughs thanks to abundance in data and computing resources.
- Flexible non-linear regression models taht can approximate any function!

Cons:

- Likelihood function no longer convex.
- Overfitting in data-scarce scenarios.

Intuition a stack of parametrized logistic regression models.

The linear models we covered so far are based on linear combinations of fixed basis functions $\varphi_i(x)$: $y = f(\sum_{j=1}^{d} w_j \varphi_j(x))$:

$$f : \begin{cases} \text{linear}, \ \varphi : \text{identity} \rightarrow \text{Linear regression}, \\ \text{linear}, \ \varphi : \text{nonlinear} \rightarrow \text{Linear regression with basis}, \\ \text{logistic}, \ \varphi : \text{identity} \rightarrow \text{Logistic regression}. \end{cases} \tag{4.1}$$

Our goal now is to make the basis functions $\varphi_j(x)$ depends on parameters and then allow these parameters to be adjusted along with the coefficients during model training.
This leads to the basic feed-forward neural network (no-feedback connections like RNNs) model which can be described as a series of functional transformations.

$$\text{i.e.} \quad y = w^T \varphi(x:\theta) \tag{4.2}$$

$$\text{First layer:} \quad h_q^{(1)} = f^{(1)}(\sum_{i=1}^{d} w_{iq}^{(1)} x_i + b_q^{(1)}), \quad q = 1, ..., Q^{(1)} \tag{4.3}$$

$$\text{In matrix vector notation:} \quad H^{(1)} = f^{(1)}(XW^{(1)} + b^{(1)}), \tag{4.4}$$

$$\text{where:} \quad H^{(1)} \in \mathbb{R}^{N \times Q^{(1)}}, \quad X \in \mathbb{R}^{N \times D}, \quad W^{(1)} \in \mathbb{R}^{D \times Q^{(1)}} \quad \text{and } b^{(1)} \in \mathbb{R}^{1 \times Q^{(1)}} \tag{4.5}$$

$$\text{The output layer is:} \quad y = f^{(L)}(H^{(L-1)}W^{(L)} + b^{(L)}) \tag{4.6}$$

$$\text{where:} \quad y \in \mathbb{R}^{N \times Q^{(L)}}, \quad H^{(L-1)} \in \mathbb{R}^{N \times Q^{(L-1)}}, \tag{4.7}$$

$$W^{(L)} \in \mathbb{R}^{Q^{(L-1)} \times Q^{(L)}} \quad \text{and } b^{(L)} \in \mathbb{R}^{1 \times Q^{(L)}} \tag{4.8}$$

Given data: $((x_1, y_1), ..., (x_N, y_N))$, $x_i \in \mathbb{R}^0$, $y_i \in \mathbb{R}^{QL}$ we have to make the following choices regarding the network architecture:

- Number of hidden layers: L
- Dimensionality of each layer: $Q^{(1)}, ..., Q^{(L)}$
- Activation function: sigmoid, tanh, ReLU, etc.

4.2 Most common activation functions

- Identity: $f(x) = x,\ f'(x) = 1,$ Range $(-\infty, +\infty),\ C^\infty$.
- Logistic: $f(x) = \frac{1}{1+e^{-x}},\ f'(x) = f(x)(1 - f(x)),$ Range $(0, 1),\ C^\infty$.
- Tanh: $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1,\ f'(x) = 1 - f(x)^2,$ Range $(-1, +1),\ C^\infty$.
- ReLU: $f(x) = \begin{cases} 0, & x < 0, \\ x, & x \geq 0. \end{cases}$, $f'(x) = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0. \end{cases}$, Range $[0, +\infty),\ C^0$.
- Exponential: $f(a, x) = \begin{cases} a(e^x - 1), & x < 0, \\ x, & x \geq 0. \end{cases}$, $f'(x) = \begin{cases} f(a, x), & x < 0, \\ 1, & x \geq 0. \end{cases}$, Range $(-a, +\infty)$, $\begin{cases} C^1, & a = 1, \\ C^0, & \text{otherwise.} \end{cases}$.
- Sine: $f(x) = \sin(x),\ f'(x) = \cos(x),$ Range $[-1, +1],\ C^\infty$.
- Gaussian: $f(x) = e^{-x^2},\ f'(x) = -2xe^{-x^2},$ Range $(0, +1],\ C^\infty$.

4.3 Most common output units

- Linear: $y = H^{(L-1)}W^{(L)} + b^{(L)}$.
- Logistic: $y = \sigma(H^{(L-1)}W^{(L)} + b^{(L)}), \sigma(a) = \frac{1}{1+e^{-a}} \rightarrow$ binary classification.
- Soft-max: $z = H^{(L-1)}W^{(L)} + b^{(L)}$, softmax$(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, $i = 1, ..., N, j = 1, ..., C$ corresponding to multi-class classification.

4.4 Training

$$p(y|x,\theta) = \mathcal{N}(y|f(x,\theta),\sigma^2) \tag{4.9}$$

$$\Rightarrow -\log p(y|x,\theta) = \frac{1}{2\sigma^2}\sum_{i=1}^{N}[f(x_i;\theta)-y_i]^2 + \frac{N}{2}\log(2\pi\sigma^2). \tag{4.10}$$

Specially, we are interested in estimating the network parameters θ. For that, if suffices to minimize the sum-of-squares error loss:

$$\text{Regression:} \quad \mathbb{E}(\theta) = \frac{1}{2}\sum_{i=1}^{N}[f(x_i;\theta)-y_i]^2, \quad \theta^* = \arg\min \mathbb{E}(\theta) \tag{4.11}$$

$$\text{Binary classification:} \quad \mathbb{E}(\theta) = -\sum_{i=1}^{N} y_i \log f(x_i;\theta) + (1-y_i)\log(1-f(x_i;\theta)) \tag{4.12}$$

And perform Gradient descent:

$$\theta_{n+1} = \theta_n - \eta\nabla_\theta \mathbb{E}(\theta) \tag{4.13}$$

4.5 Overfitting an regularization

$\mathbb{L}_2$ parameter regularization (weight decay)

This is the most common form of regularization.
Modifies the loss function as

$$\mathbb{E}(\theta) = \frac{1}{2}\sum_{i=1}^{n}[f(x_i;\theta)-y_i]^2 + \frac{\lambda}{2}w^T w. \tag{4.14}$$

Here the first term corresponds to a Gaussian, second term represent the prior on the weights: $p(w) = \mathcal{N}(0,\lambda^{-1}I)$.
Drives the weights closer to the origin.
Caution: This may be inconsistent with the scaling properties of network mapping (See Bishop 5.5.1).
Ideally, we would like a regularizer that is invariant under linear transformations (i.e., invariant to scaling of the weights and shift of the biases).
Such a regularizer is given by: $\frac{\lambda_1}{2}\sum_{w\in W_1} w^2 + \frac{\lambda_2}{2}\sum_{w\in W_2} w^2$.
More generally, we can consider priors of the form:

$$p(w) \propto \exp(\frac{1}{2}\sum_k a_k \|w\|_k^2), \quad \text{where} \quad \|w\|_k^2 := \sum_{j\in W_k} w_j^2. \tag{4.15}$$

Then, we can learn the optimal a_k via MLE (connections with ARD).

$\mathbb{L}_1$ parameter regularization

Modifies the loss function as

$$\mathbb{E}(\theta) = \frac{1}{2}\sum_{i=1}^{n}[f(x_i;\theta)-y_i]^2 + \alpha\|w\|_1. \tag{4.16}$$

Here $\|w\|_1 := \sum_j |w_j|$ is the $\mathbb{L}_1$-norm.
This induces sparsity in the model parameters (for large enough α), i.e. discards features are not needed.

Early stopping

When training large models that overfit, we often observe that the training and test errors decrease steadily in time, but at some point the test error starts increasing.

Dropout

The standard neural net hidden layer is:

$$H^{(l)} = f(H^{(l-1)}W^{(l)} + b^{(l)}), \tag{4.17}$$

with dropout this becomes:

$$r_j^{(l)} \sim \text{Bernoulli}(p) \tag{4.18}$$

$$z^{(l)} = r^{(l)}\dot{H}^{(l-1)} \quad \text{(element wise)} \tag{4.19}$$

$$H^{(l)} = f(z^{(l)}W^{(l)} + b^{(l)}). \tag{4.20}$$

Data augmentation

In classification tasks we can increase the size of the ...

Network initialization

Xavier initialization

Suppose we have an input X and a linear neuron with random weights W that outputs y. Then:

$$y = w_1x_1 + w_2x_2 + ... + w_dx_d \tag{4.21}$$

Let's take a look at the variance of each term:

$$\text{Var}[w_ix_i] = \mathbb{E}[x_i]^2\text{Var}(w_i) + \mathbb{E}[w_i]^2\text{Var}(x_i) + \text{Var}(w_i)\text{Var}(x_i). \tag{4.22}$$

If our inputs and weights have mean zero then this simplifies to:

$$\text{Var}[w_ix_i] = \text{Var}(w_i)\text{Var}(x_i). \tag{4.23}$$

If we also assume that the x_i and w_i are all iid, then we can work out the variance of y:

$$\text{Var}[y] = \text{Var}[w_1x_1 + w_2x_2 + ... + w_dx_d] = n\text{Var}[w_i]\text{Var}[x_i]. \tag{4.24}$$

In other words, the variance of the output is the variance of the input, scaled by $\text{Var}[w_i]$. So if we want the variance of the input and the variance of the output to be the same, then $\text{Var}[w_i]$ should be 1.

$$\text{Var}[w_i] = \frac{1}{d} = \frac{1}{d_{in}}. \tag{4.25}$$

If we now repeat the same analysis for the backpropagated signal (see Glorot + Bengio) we similarly get that $\text{Var}[w_i] = \frac{1}{d_{out}}$. As these constraints can only be satisfied if $d_{in} = d_{out}$ (which is rarely the case), we can just compromise by taking the average:

$$\text{Var}[w_i] = \frac{2}{d_{in} + d_{out}}. \tag{4.26}$$

Leading to the so called Xavier initialization:

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}, \frac{\sqrt{6}}{\sqrt{d_{in} + d_{out}}}], \quad \text{or} \quad w \sim \mathcal{N}(0, \frac{2}{d_{in} + d_{out}}). \tag{4.27}$$

Chapter 5

Image classification with convolutional neural networks

5.1 Convolutional Neural Networks (CNNs/ConvNets)

Very similar to ordinary NNs:

- They are made up of neurons that have differentiable weights and biases.
- Each neuron receives same inputs, performs a dot product, optimally followed by a non-linearity.
- The forward-pass and loss function are fully differentiable.

$\Rightarrow$ They can address the unfavorable complexity of NNs for high-dimensional inputs, by making the explicit assumption that the inputs "live" on a grid (e.g., regularly sampled timeseries, images, etc). This assumption allows us to encode certain structure and properties into the architecture.
E.g., take a single image of resolution $32 \times 32 \times 3$. Then a single neuron in the first layer of a fully connected NN would add $32 \times 32 \times 3 = 3072$ weights.
Ok, this may sound manageable, but consider a more realistic $200 \times 200 \times 3$ image.
This would correspond to 120,000 weights, just for a single neuron! ($\rightarrow$ overfitting). $\rightarrow$ ConvNets are NN that use convolution instead of general matrix multiplication in at least one of their layers.

5.2 Convolution: Definition in 1D

Assume we are given a time-series $x(t)$ sampled at regular intervals (x could be noisy).
To filter out the noise we would like to compute some weighted average of the measurements. To do so, we can choose a weighting function $w(s)$ and obtian a new smoothed function $s(t)$ as:

$$s(t) = \int_{-\infty}^{\infty} x(s)w(t-s)ds = \int_{-\infty}^{\infty} w(s)x(t-s)ds, \quad s(t) = (x * w)(t), \tag{5.1}$$

convolution is a commutative linear operation.

Discrete convolution in 1D

$$s(t) = (x * w)(t) = \sum_{s=-\infty}^{\infty} x(s)w(t-s) \tag{5.2}$$

Discrete convolution in a 2D image (with $m \times n$ pixels)

Convolution:

$$\text{Convolution:} \quad S(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n) = \sum_m \sum_n K(m,n)I(i-m,j-n) \tag{5.3}$$

$$\text{Correlation:} \quad S(i,j) = \sum_m \sum_n K(m,n)I(i+m,j+n) \tag{5.4}$$

5.3 Classification of images with CNN in PyTorch

Why cross entropy is a good choice of classification error?
Recall the definition of cross entropy and KL-divergence, assume p is the true distribution and q is your model distribution:

$$H(p,q) = -\int p(x)\log(q(x))dx \tag{5.5}$$

$$KL[p||q] = \int p(x)\log(\frac{p(x)}{q(x)})dx = \int p(x)\log(p(x))dx - \int p(x)\log(q(x))dx = \text{const} + H(p,q) \tag{5.6}$$

that means minimizing the cross entropy is equivalent to minimizing the KL divergence. In other words, it tells you that your surrogate model is trying to find a better way to match the predicted density with your original data distribution.

Chapter 6

Recurrent neural networks and LSTMs

6.1 Recurrent Neural Networks

Most prediction tasks we've looked at so far involved pretty simple outputs, such as real values or discrete categories. However, more often than not, we are interested in predicting more complex structures such as images or sequences.
If both inputs and outputs are sequences, we refer to this as sequence to sequence prediction.

Examples

- Languages modeling: modeling the distribution over English text.
- Speech to text or text to speech translation.
- Caption generation: we take an image as input and want to produce a natural language description of the image.
- Machine translation.

Definition

We have a sequence data set of $\{y_t : t = 1, ..., T\}$, $y_t \in \mathbb{R}^d$.
Our goal is to model the next sequence value $\hat{y}_t$ as a function of the previous values (lags) $y_{t-1}, y_{t-2}, ...$
Assuming two lags, this translates into learning the function: $\hat{y}^t = f(y_{t-1}, y_{t-2})$.
This defines a recurrent neural network architecture (i.e.):

$$\hat{y}_t = h_t V + c = \tanh(h_{t-1} W + y_{t-1} U + b) \tag{6.1}$$
$$h_{t-1} = \tanh(h_{t-2} W + y_{t-2} U + b) \tag{6.2}$$
$$h_{t-2} = 0. \tag{6.3}$$

Notice how the parameters W, U, b are shared.
They can be trained by minimizing the MSE loss:

$$\mathcal{L}(\theta) := \frac{1}{T-2} \sum_{t=3}^{T} (y_t - \hat{y}_t)^2, \quad \theta := \{W, U, b, V, c\}. \tag{6.4}$$

Examples

Predict the dynamics of a sine wave, i.e.,

$$y(t) = \sin(\pi t) \tag{6.5}$$

$y_t = f(y_{t-1}, y_{t-2})$, $y_t \in \mathbb{R}^{N\times D}$ is a sequence of d-values and y_{t-1}, y_{t-2} are sequences constructed by taking lags (i.e.).

$$y_t := (y(2), y(3), ..., y(t)), \tag{6.6}$$
$$y_{t-1} := (y(1), y(2), ..., y(t-1)), \tag{6.7}$$
$$y_{t-2} := (y(0), y(1), ..., y(t-2)), \tag{6.8}$$
$$\tag{6.9}$$

Inputs: $X \in \mathbb{R}^{L\times N\times D} : (y_{t-1}, y_{t-2})$.
Outputs: $Y \in \mathbb{R}^{N\times D} : \hat{y}_t$.
Forward pass:

$$h_{t-2} = 0 \tag{6.10}$$
$$h_{t-1} = \tanh(h_{t-2}W + X[0,:,:]U + b) \tag{6.11}$$
$$h_t = \tanh(h_{t-1}W + X[1,:,:]U + b) \tag{6.12}$$
$$\hat{y}_t = h_t V + c \tag{6.13}$$

6.2 Long short-term memory network

In an LSTM one replaces the hidden units: $h_t = \tanh(h_{t-1}W + y_{t-1}U + b)$ with:

$$h_t := o_t \odot \tanh(s_t) \quad \rightarrow \text{output vector} \quad (y_{t-1} : \text{input vector}) \quad \rightarrow \text{output gate} \tag{6.14}$$
$$o_t := \sigma(h_{t-1}W_o + y_{t-1}U_o + b_o) \tag{6.15}$$
$$s_t := f_t \odot s_{t-1} + i_t \odot \tilde{s}_t \tag{6.16}$$
$$\tilde{s}_t := \tanh(h_{t-1}W_s + y_{t-1}U_s + b_s) \tag{6.17}$$
$$i_t = \sigma(h_{t-1}W_i + y_{t-1}U_i + b_i) \quad \rightarrow \text{external input gate} \tag{6.18}$$
$$f_t = \sigma(h_{t-1}W_f + y_{t-1}U_f + b_f) \quad \rightarrow \text{forget gate} \tag{6.19}$$

The parameters $\theta := \{W_o, U_o, b_o, W_s, U_s, b_s, W_i, U_i, b_i, W_f, U_f, b_f\}$ are shared across lags and can be learned by minimizing the MSE loss.

6.3 Gated Recurrent Unit

An alternative unit with less parameters:

$$z_t = \sigma(h_{t-1}W_z + y_{t-1}U_z + b_z) \quad \rightarrow \text{update gate vector} \tag{6.20}$$
$$r_t = \sigma(h_{t-1}W_r + y_{t-1}U_r + b_r) \quad \rightarrow \text{reset gate vector} \tag{6.21}$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tanh[(r_t \odot h_{t-1})W_h + y_{t-1}U_n + b_n] \tag{6.22}$$

With parameters $\theta := \{W_z, U_z, b_z, W_r, U_r, b_r, W_n, U_n, b_n\}$.

Chapter 7

General concepts of supervised learning

7.1 The general framework

- There is an unknown distribution p on $\mathcal{X} \times \mathcal{Y}$.
- The training set $S = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_m, y_m)\}$, is an IID sample from p.
- The hypothesis is a function $\hat{f} : \mathcal{X} \to \mathcal{Y}$.
- The learning algorithm is a function $\mathcal{A} : S \to \hat{f}$.
- The hypothesis space $\mathcal{F}$ is the space of functions accessible to $\mathcal{A}$.

We want f to be good on future examples drawn from p, where "good" is defined in terms of a loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$.

How can we ever be sure that we can find a good f, especially when $mathcalF$ is huge? This is what Statistical Learning Theory is about.

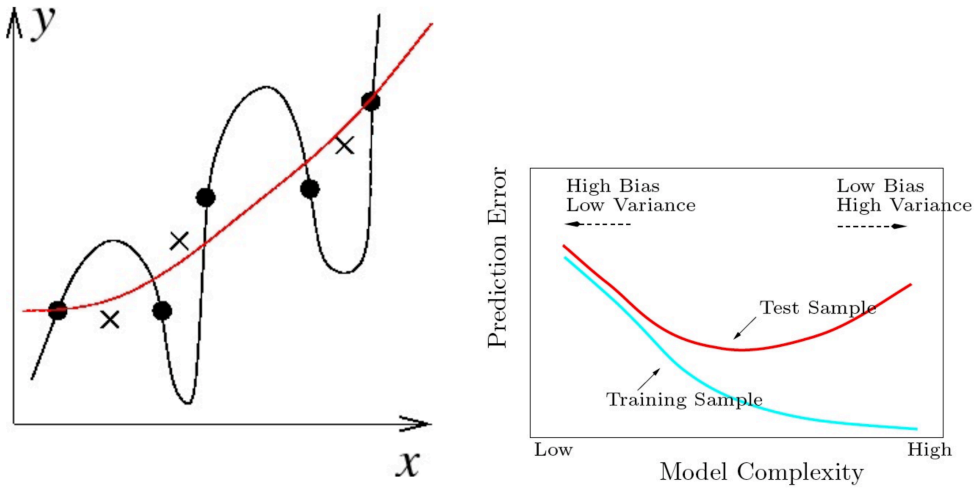
7.2 Three notions of error

- The training error or empirical error: $\epsilon_{emp}[\hat{f}] = \frac{1}{m} \sum_{i=1}^{m} l(\hat{f}(\boldsymbol{x}_i, y_i))$. This is what we can actually measure, and what many learning algorithms try to minimize. $\to$ Empirical risk minimization (ERM).
- The testing error: $\epsilon_{test}[\hat{f}] = \frac{1}{m'} \sum_{i=1}^{m'} l(\hat{f}(\boldsymbol{x}'_i, y'_i))$
- The true error:$\epsilon_{true}[\hat{f}] = \mathbb{E}_{(x,y)\sim p} l(\hat{f}(\boldsymbol{x}, y))$. This is what an ideal algorithm would minimize, but it can't be measured since p is unknown.

7.3 Overfitting and underfitting tradeoff

In general, because $\hat{f}$ is explicitly chosen to minimize $\epsilon_{emp}[\hat{f}]$, it will tend to be an overoptimistic estimate of the error, i.e., $\epsilon_{true}[\hat{f}] > \epsilon_{emp}[\hat{f}]$ and $\epsilon_{test}[\hat{f}] > \epsilon_{emp}[\hat{f}]$. This is called overfitting. See figure 7.3. Question: How can we avoid overfitting? Restrict hypothesis space! But how much?

The hypothesis space should be big enough but not too big. Alternatively, we need to add a regularizer.



7.4 Regularized risk minimization (RRM)

The most general framework for finding the right balance between overfitting and underfitting in supervised learning is RRM:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} [\frac{1}{m} \sum_{i=1}^{m} l(f(x_i), y_i) + \lambda \Omega(f)] \tag{7.1}$$

Terminology:

- $\mathcal{F}$: hypothesis space.
- $l(\hat{y}, y)$: loss function.
- $\Omega : \mathcal{F} \rightarrow \mathbb{R}^+$: regularization functional.

The right compromise is found by tuning the regularization parameter λ.
Two good examples could be Lasso and Ridge for linear regression:

$$L_{lasso} = \|y - X\beta\|_2^2 + \lambda \|\beta\|_1, \tag{7.2}$$

$$L_{Ridge} = \|y - X\beta\|_2^2 + \lambda \|\beta\|_2^2, \tag{7.3}$$

$$\tag{7.4}$$

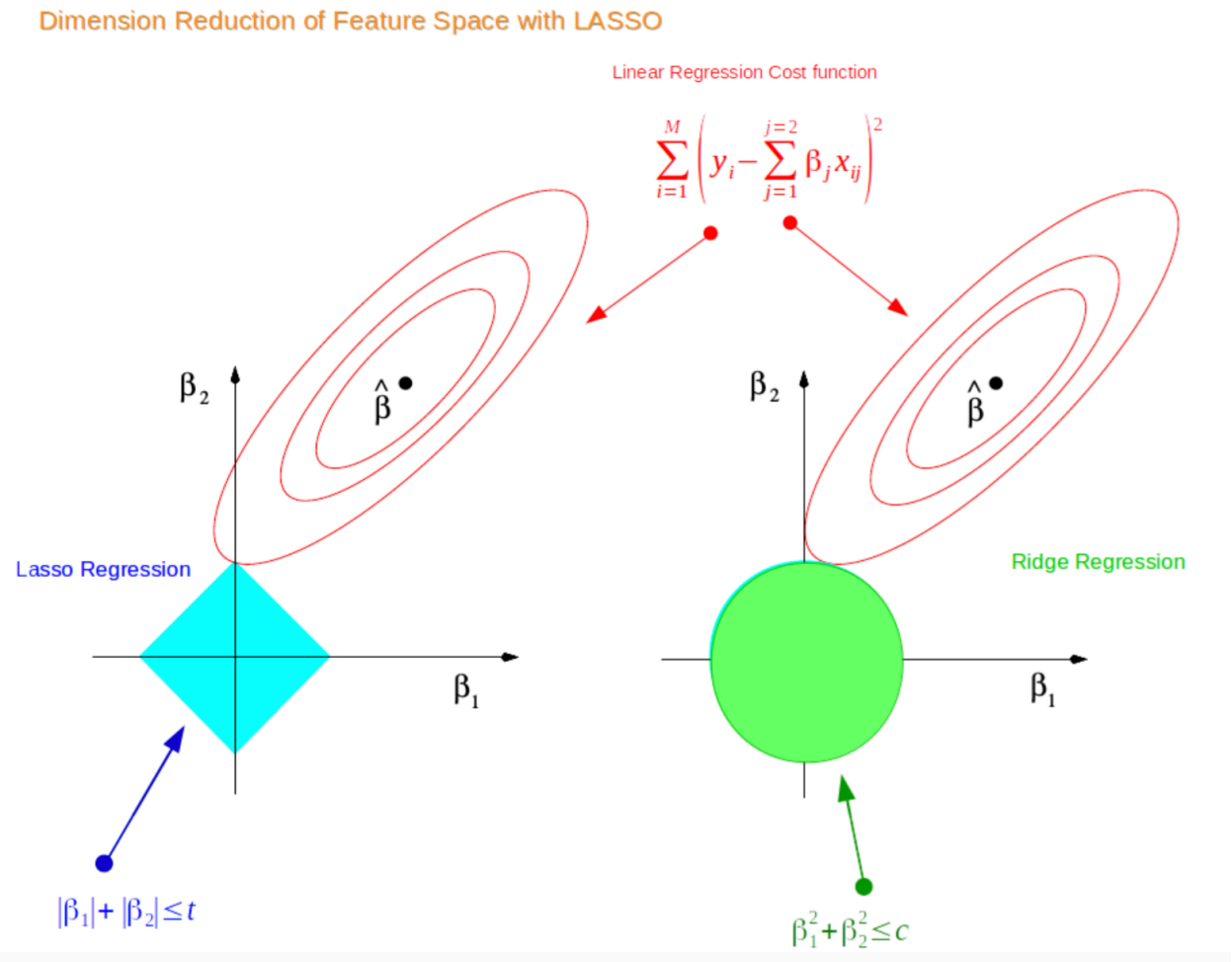
Lasso is good at imposing sparsity and ridge would also has the mechanism to regularize the parameters feasible space. More could be found in Tibshirani's paper. Also, see figure 7.4

7.5 Statistical Learning Theory

Concentration of measure:
The reason that we have any hope of learning a good $\hat{f}$ at all is that for any fixed $f \in \mathcal{F}$, over the choice of training or testing sets,

$$\mathbb{E}(\epsilon_{emp}[\hat{f}]) = \mathbb{E}(\epsilon_{test}[\hat{f}]) = \mathbb{E}(\epsilon_{true}[\hat{f}])$$

Moreover, as the size of training testing sets increases, these quantities are more and more concentrated around $\mathbb{E}(\epsilon_{true}[\hat{f}])$. However, a very unlucky choice of training set can always mess us up.



7.6 Generalization Bounds

Statistical Learning Theory proves bounds of the form:

$$\mathbb{P}[\epsilon_{true}[\hat{f}] > \epsilon_{emp}[\hat{f}+\epsilon]] < \delta \tag{7.5}$$

where ϵ is a complicated function depending on δ, the size of the function class $\mathcal{F}$, and the nature of the regularization. $\rightarrow$ Probably Approximately Correct (PAC) bounds.
Example: if the VC dimensions of $\mathcal{F}$ is d, then:

$$\mathbb{P}[\epsilon_{true}[\hat{f}] > \epsilon_{emp}[\hat{f}+\sqrt{\frac{d(\log(\frac{2m}{d}+1))+\log(4/\delta)}{m}}]] < \delta \tag{7.6}$$

In reality even the best such bounds are ridiculously loose.

7.7 Cross validation

Holdout set:
The generalization bounds from statistical learning theory are framed entirely in terms of the training set, therefore they reveal fundamental properties of the learning algorithm. Unfortunately, they are almost always incred-

ibly loose.
The more practical way to estimate ϵ_{true} is to split the training data into:

- The true training set, used to train the algorithm.
- A holdout set $\{(x_1, y_1), (x_2, y_2)..., (x_p, y_p)\}$ which is only used to compute the holdout error $\epsilon_{h.o.}[\hat{f}] = \frac{1}{p}\sum_{i=1}^{p} l(\hat{f}(x_i), y_i)$.

Holdout error is an unbiased estimator of $\epsilon_{true}[\hat{f}]$, i.e., $\mathbb{E}[\epsilon_{h.o.}[\hat{f}] = \epsilon_{true}[\hat{f}]$.

7.8 k-fold cross validation

k-fold cross validation uses the holdout idea to set internal parameters θ of learning algorithms (e.g., k in k-NN):

- Set θ to some value.
- Split the training set into k roughly equal parts $S_1, S_2, ..., S_k$.
- For each i, train the algorithm on $S_1, ..., S_{i-1}, S_{i+1}, ..., S_k$ to get $\hat{f}_i$.
- Compute the holdout error of each $\hat{f}_i$ on the corresponding S_i.
- Average to get an estimator of $\epsilon_{true}[\hat{f}_\theta]$.
- Iterate over a range of θ values and finally set θ to be the value that minimizes the cross-validation error.

Chapter 8

Sampling and quantification of uncertainty

8.1 Sampling Methods

- Given $p(x)$, draw samples.
- Estimate samples, learn $p(x)$.
- Estimate statistics.
- Do Bayesian inference.

Why is sampling so powerful and useful?
Approximation: It allows us to approximate expectations:

- estimate statistics.
- posterior inference.

Sampling: visualization of typical draws from a complicated distribution (then perhaps cluster the samples?)
$\rightarrow$ Why expectations?

- Any probability is an expectation, e.g., $p(x \in A) = \mathbb{E}[\mathbb{1}_{\{x \in A\}}], \mathbb{1}_{\{x \in A\}} := \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$
- Approximation is needed for intractable sums or integrals. Many sum or integrals can be written as expectations.

Pros of sampling approximation (mc, mcmc, IS, etc):

- (1) They are easy to use/implement/understand
- (2) They are very general purpose.
- (3) It has well understood assymptotical theoretical guarantees (but it may then innefrecion in practice)

Cons:

- (1) They are too easy to use/implement/understand. Often used inappropriately.
- (2) They tend to be slow compared to exact or deterministic approximation (they requie many samples)
- (3) Getting "good/representative" samples may be difficult/inefficient.
- (4) It can be difficult to assess the performance of the method.

8.2 Different scenarios

- Evaluate a probability: Given x, compute $p(x) = \frac{\tilde{p}(x)}{z_p}$, or the unnormalized $\tilde{p}(x)$.
- Sample from a distribution: Given $p(x)$, generate a representative sample x.
- Evaluate statistics /moments: Given $p(x)$, compute $\mathbb{E}_{x\sim p(x)}[f(x)]$.
- Grand challenge: x is high-dimensional, p, f are complex.

8.3 Monte Carlo approximation

History

It is a trivial idea, yet the formal mathematical foundations were not established till the 1940s.
Enrico Fermi (Nobel prize for work in radiation in 1938, at the age of 37!), Italian physicist.
$\rightarrow$ He was doing Monte Carlo approximations by hand (very tedious)! (to try to estimate the outcomes of an experiment.)
John Von Neumann (one of the greatest American mathematicians):(1940s) Passionate about building and designing computers.
Stan Ulam (math/unclear physics).
$\Rightarrow$ All these folks were working at Los Alamos during WWII to develop the nuclear bomb...
After the war, in 1946, Ulam during a sick day wanted to estimate/calculate the probability of jetting the perfect "hand" in solitaire. He couldn't do it analytically, hence he thought about it approximately using computers and sampling.
These folks got together and developed the modern foundations of Monte Carlo approximation.
Why it it called Monte-Carlo?
The story says that Ulam had an uncle that liked card games "games of chance" and gambling of the Monte Carlo casino.

Example

- What is the average height of the students in this class? $\mathbb{E}[h] = \frac{1}{N}\sum_{i=1}^{N} h_i$.
- What is the average height of the people in center city?
  $\mathbb{E}_{p\in C}[h(p)] := \frac{1}{|C|}\sum_{p\in C} h(p) \approx \frac{1}{S}\sum_{i=1}^{S} h(p_S)$, but we don't know how many people are in center city ...
  for a random survey of S people in C, $p_S \sim$ are drawn iid from C.
- Making predictions using a probabilistic medel: $p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$
  $\rightarrow p(x^*|\mathcal{D}) = \int p(x^*|\theta,\mathcal{D})p(\theta|\mathcal{D})d\theta \approx \frac{1}{S}\sum_{s=1}^{S} p(x^*|\theta^{(s)},\mathcal{D}), \theta^{(s)} \sim p(\theta|\mathcal{D})$.

8.4 Importance Sampling (not a sampling method!)

It is an extension of Monte Carlo for approximating intractable integrals.
Recall this assumes we can easily sample from $p(x)$!
Importance sampling can help us deal with cases where we can't generate samples from $p(x)$. But even in cases where we can draw samples from $p(x)$, importance sampling can help us increase the accuracy. (convergence rate of our estimate!)

Setup

Assume $p(x)$ is a density. Then:

$$\mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)p(x)dx = \int [f(x)\frac{p()x}{q(x)}]q(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)\frac{p(x_i)}{q(x_i)}, x_i \sim q(x) \tag{8.1}$$

$$\forall q(x) \text{ pdf such that } q(x) = 0 \Rightarrow p(x) = 0, \text{ i.e., p is absolutely continuous with respect to q.} \tag{8.2}$$

$\Rightarrow$

$$\mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)w(x_i) = \hat{\mu}_n^{IS}, \text{ where } x_i \sim q(x) \text{ serves as proposal distribution.} \tag{8.3}$$

$$\text{and} w(x_i) := \frac{p(x_i)}{q(x_i)} \rightarrow \text{importance weight.} \tag{8.4}$$

Remarks

- (1) $\mathbb{E}_{x\sim q(x)}[\hat{\mu}_n^{IS}] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{x\sim q(x)}[f(x_i)\frac{p(x_i)}{q(x_i)}] = \frac{1}{n}\sum_{i=1}^{n}\int f(x_i)\frac{p(x_i)}{q(x_i)}q(x_i)dx = \hat{\mu}_b$
  Hence the IS is unbiased, consistent, and converges as $O(\frac{1}{\sqrt{n}})$.
- (2) Var[$\hat{\mu}_n^{IS}$] = Var[$f(x)\frac{p(x)}{q(x)}$], this is different from the simple mc variance estimate, and this hints on how one can use IS to improve the accuracy of our estimator, depending on how we choose $q(x)$.

In fact, it is straightforward to solve for the optimal theoretical value of $q^*(x)$, but in practice it may be very difficult to sample from that distribution.

Scenarios

- (1) Can't sample from p, use IS to correct for sampling from a tractable distribution q (Then, choose q to be close to q).
- (2) Whether or not we can sample from p, use IS to improve upon the basic mc estimator.

8.5 Monte Carlo approximation

Goal

Approximate a quantity of interest (e.g., an expectation) using samples.
$\mathbb{E}[f(x)]$, $x \in \mathbb{R}^d$ and the expectation is intractable.

Definition

If $x_1, ..., x_n \sim p$, iid then: $\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$ is a basic Monte Carlo estimator. Also ($\mathbb{E}_{x\sim p(x)}[f(x)] := \int f(x)p(x)dx$) this is just the sample mean.

Remarks

- (1) $\mathbb{E}[\hat{\mu}_n] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[f(x_i)] \xrightarrow{n\rightarrow\infty} \mathbb{E}[f(x)]$, hence $\hat{\mu}_n$ is an unbiased estimator.
- (2) $\hat{\mu}_n \xrightarrow{p} \mathbb{E}[f(x)]$ as $n \rightarrow \infty$ (convergence in probability: $\forall \epsilon > 0, p(|\hat{\mu}_n - \mathbb{E}[f(x)]| < \epsilon) \rightarrow 1$). Hence $\hat{\mu}_n$ is a consistent estimator. (i.e., when n is large then we get the correct answer with very high probability assuming that Var$[f(x)] < \infty$ by the weak law of large numbers).

- (3) $\text{Var}[\hat{\mu}_n] = \frac{1}{n^2}\text{Var}[f(x_i)] = \frac{1}{n}\text{Var}[f(x)] \xrightarrow{n\to\infty} 0. \Rightarrow \frac{1}{\sqrt{n}}\text{std}[f(x)]$, hence the convergence rage is $O(\frac{1}{\sqrt{n}})$ (regardless of the dimension of x!!).
- (4) Combing (1) and (3) we could derive: $\text{MSE}[\hat{\mu}_n] = \text{bias}^2 + \text{var} = \text{var} = \frac{1}{n}\text{Var}[f(x)] \xrightarrow{n\to\infty} 0.$

Notes

Despite the fact that we know this rate of convergence, it may be very difficult to know what the actual error is, since we don't know the variance $\text{Var}[f(x)]$. (... remember we're just trying to approximate $\mathbb{E}[f(x)]$).
A practical limitation: One needs to be able to efficient sample from $p(\cdot)$!
E.g., approximating $\pi \approx 3.14$ using mc.

8.6 Importance sampling for unnormalized distributions

Often we may know $p(x)$ or $q(x)$ only up to a normalizing constant.
E.g., Posterior inference: $p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta)$.

Setup

Assume $p(x) = \frac{\tilde{p}(x)}{z_p}$, $q(x) = \frac{\tilde{q}(x)}{z_q}$, $\int \tilde{q}(x)dx = z_q > 0$ and $\int \tilde{p}(x)dx = z_p > 0$.
An alternative unit with less parameters:

$$\Rightarrow \mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \int f(x)\frac{\tilde{p}(x)}{\tilde{q}(x)}\frac{z_q}{z_p}q(x)dx, \tag{8.5}$$

Assume we can sample efficiently from $q(x)$, and can evaluate $\tilde{p}(x)$, $\tilde{q}(x)$.

$$\Rightarrow \mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)\frac{\tilde{p}(x)}{\tilde{q}(x)}\frac{z_q}{z_p}q(x)dx \approx \frac{z_q}{z_p}\frac{1}{n}\sum_{i=1}^{n} f(x_i)\tilde{w}(x_i), \quad x_i \sim q(x),\ \tilde{w}(x) = \frac{\tilde{p}(x)}{\tilde{q}(x)}. \tag{8.6}$$

But we still don't know the normalizing constants ... we'll do a Monte-Carlo approximation to approximate the ratio of normalizing constants:

$$\frac{z_q}{z_p} = \frac{1}{z_p}\int \tilde{p}(x)dx = \int \frac{\tilde{p}(x)}{\tilde{q}(x)}q(x)dx = \mathbb{E}_{x\sim q(x)}[\frac{\tilde{p}(x)}{\tilde{q}(x)}] \approx \frac{1}{n}\sum_{i=1}^{n}\tilde{w}(x_i) \tag{8.7}$$

$$\Rightarrow \mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{\frac{1}{n}\sum_{i=1}^{n} f(x_i)\tilde{w}(x_i)}{\frac{1}{n}\sum_{i=1}^{n}\tilde{w}(x_i)} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)\hat{w}(x_i), \quad x_i \sim q(x), \text{ where } \hat{w}(x_i) = \frac{\tilde{w}(x_i)}{\sum_{i=1}^{n}\tilde{w}(x_i)}. \tag{8.8}$$

here $\hat{w}(x_i)$ is playing a role like "approximate importance weights".
Keep in mind that we made two approximations to get here! Hence, this is expected to perform worse than regular IS.

How to choose a good importance sampling proposal distribution?

Example: Suppose that $f(x)$ is the return of some investment and we want to approximate the expected return using IS.

Conclusion:
Choose $q(x)$ to be large when $|f(x)|p(x)$ is large!
Caution:
It is difficult to assess how good or bad our estimator is ...

8.7 Rejection sampling

It's a general method for generating exact samples from general distribution. (Typically in low dimension ...)

Uniform case

Goal: Generate samples from a uniform distribution on same complicated set.
We assume that we can evaluate the indicator function $\mathbf{1}_{\{x\in A\}} = \begin{cases} 1, & x \in A \\ 0, & x \notin 0 \end{cases}$ (to some accuracy).

Basic idea: Draw uniform samples from a larger simple set B.
Then evaluate $\mathbf{1}_{\{x\in A\}}$ for each sample and choose whether the sample should be accepted or rejected.

That's a general strategy in sampling methods: Sample from a simpler distribution set, check a deterministic, or random condition to decide whether to accept or reject the sample.

Proposition: If $A \subset B$, and $y_1, y_2, ... \sim$ Uniform(B) and iid, and $X = y_k$, where $k = \min\{k : y_k \in A\}$ this is a random variable itself.

Then $X \sim$ Uniform(A).

Non-uniform case (with densities)

Goal: Sample from a complicated, pdf $p(x)$, $x \in \mathbb{R}^d$.
Assume we are given $\tilde{p}(x)$, the un-normalized density, i.e., $p(x) = \frac{\tilde{p}(x)}{z_p}$, $z_p > 0$, $z_p = \int \tilde{p}(x)dx$.
Algorithm:

- (1) Choose a pdf $q(x)$, such that: $\exists c > 0$, such that $cq(X) \geq \tilde{p}(x)$, $forallx$ (we can set c as $c = \max(\frac{\tilde{p}(x)}{q(x)})$) and it is easy to sample from.
- (2) Sample x from $q(x)$, then sample $y \sim$ Uniform$(0, cq(x))$ (given x)
- (3) If $y \leq \tilde{p}(x)$, then $z = x$ otherwise we "reject" x and return to step #2.

Output $z \sim p(x)$.
Key limitation: Very hard to design efficient proposal distributions in high-dimensions...

8.8 Markov Chain Monte Carlo (one of the top-10 algorithms of the 20th century)

A powerful and general purpose tool for approximating expectations and sampling from complex high-dimensional distribution:

Goal: Sample from $p(x)$, or compute/approximate $\mathbb{E}_{x\sim p(x)}[f(x)]$, $x \in \mathbb{R}^d$.

Recall basic mc: $\mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$, x_i are iid from $p(x)$.

$p(x)$ maybe way to complicated to sample from. Importance sampling or rejection sampling face severe difficulty in high dimension as it becomes extremely difficult to design efficient proposal distributions.

Intuition of mcmc

Idea: Start at some x_0, find a region of high probability and then navigate/explore the space by moving randomly, yet staying close to the regions of high-probability. ($p(x)$ has some structure, the mass concentrates on a "manifold" in high-dimensions).

Original paper by Metropolis $\{et.al.\}$ (1953) (Also paper by Ulam and Metropolis, 1949 introduced the idea of using Markov Chain).

Phase diagram from chemistry

"hard-disks in a box model" to study/model the properties of a material use mcmc to try to simulate this model on compute expectation $\mathbb{E}_{x\sim p(x)}[f(x)], x \in \mathbb{R}^{2n}$ where $p(x)$ is the probability density over all valid configuration/state (such as Boltzmann distribution $p(x) \propto e^{-\frac{E}{kT}}$, E is the state energy and T is the thermo dynamics temperature, k is Plank constant). This is complex due to no overlap constrain periodic boundaries.

Idea: Start from initial configuration x_0 and start exploring the space of all valid configurations.

How to explore? Define some admissible moves:
At each step/iteration, take a particle, draw a box of predetermined size, and draw a uniform point in that box. Then according to some random acceptance rule (the Metropolis rule) move the particle to that new location. Hence at each iteration we may obtain a new configuration, x_i. The sequence of accepted configurations will be used to approximate the desired expectation: $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$. But now the x_i's are not iid.

Ergodic Theorem for Markov Chain (G.D Birkhoff, Fields medal 2010 Elan Lindsetrauss)

Define Markov Chains first!

A dynamical system is ergodic if it can reach all possible states in finite time regardless of the initial condition x_0.

Recall:

$$\text{Monte Carlo: } \mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \; x_i \text{ iid draw from } p(x) \tag{8.9}$$

$$\text{mcmc: } \mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_{i=1}^{n} f(x_i) \; x_i \text{ drawn from a Markov Chain.} \tag{8.10}$$

We would like to prove that this estimate is unbiased and consistent as $n \to \infty$.
This is done using Ergodic theory.

Theorem 1 *If $(x_0, x_1, ..., x_n)$ is an irreducible (time-homogeneous), discrete Markov Chain, with stationary distribution π, then: $\frac{1}{n}\sum_{i=1}^{n} f(x_i) \xrightarrow[n\to\infty]{a.s.} \mathbb{E}_{x\sim\pi}[f(x)]$, $\forall$ bounded functions $f : x \to \mathbb{R}$. If, further, the chain is aperiodic, then $p(X_n = x|X_0 = x_0) \xrightarrow[n\to\infty]{} \pi(x)$. Hence x_n is a good sample from $\pi(x)$.*

What is a Markov Chain

$x_0 \to x_1 \to ... \to x_n$ where $x_i \in \mathcal{X} \to$ a countable set and $x_i := (x_0, ..., x_n)$.

$\begin{cases} p(x_i|x_0,...,x_{i-1}) = p(x_i|x_{i-1}) & \text{Markov property} \\ p(x_0,...,x_n) = p(x_0)p(x_1|x_0)p(x_2|x_1)...p(x_n|x_{n-1}) \end{cases}$ If a discrete Markov chain is irreducable, has a stationary distribution, and is aperiodic, then it is an Ergodic Markov chain.

Definition:
A m.c. (x_i) is (discrete/time) time-homogeneous if $p(x_{i+1} = b|x_i = a) = T_{ab}, \forall i, \forall a, b \in \mathcal{X}$ for some matrix T (could be an "infinite" matrix for continuous time m.c we'd have a transition kernel.)
i.e., the transition probabilities do not depend on time (the index i).
T is called the transition matrix of the m.c, and it is a stochastic matrix, i.e., $\sum_b T_{ab} = 1$ (the raws sum to one).

Definition:
A pmf π on $\mathcal{X}$ is a stationary/invariant distribution (with respect to T) if $\pi T = \pi$, i.e., $\sum_{a\in\mathcal{X}} \pi_a T_{ab} = \pi_b, \forall b \in \mathcal{X}$.
This reminds us of an eigenvector equation. In fact, π is called a left-eigenvector and it has an eignvalue 1.

Definition:
A m.c. (x_i) is irreducible if $\forall a, b \in \mathcal{X}, \exists t \geq 0$ such that $p(X_t = b|X_0 = a) > 0$.
Hence, no matter where we start from, we can reach every state b.

Definition:
An irreducible m.c (x_i) is called aperiodic if:

$$\forall a \in \mathcal{X}, gcd\{t : p(X_t = a|X_0 = a) > 0\} = 1 \tag{8.11}$$

where gcd means greatest common divisor. $R_a = \{t : p(X_t = a|X_0 = b) > 0\}$ is the set of times for which if we start at a, we get back to a at some point.

Examples of Markov Chains

(1) $\mathcal{X} = \{1, 2\}$
Transition matrix $T = \begin{bmatrix} p & q \\ p & q \end{bmatrix}$, it is a stochastic matrix since $p + q = 1$ (the raws sum to one).
The chain is irreducible since $p, q > 0$, hence we're guaranteed to visit all states (that wouldn't be the case if one of p or q was zero).
It turns out it is also aperiodic.
It also has a stationary distribution $\pi = (p, q)$ since $\pi T = \pi$, for such π.

(2) $\mathcal{X} = \{1, 2, 3, 4\}$
Transition matrix $T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 1 & 0 \end{bmatrix}$. This is symmetric random walk with reflecting boundaries.

- irreducibility (Yes!)
- aperiodicity (No $\rightarrow$ in fact it is periodic.)

8.9 The Metropolis Algorithm

Setup:
Given a pmf π defined on a set of states $\mathcal{X}$ (countable), and a function $f : \mathcal{X} \rightarrow \mathbb{R}$.

Goal:
Approximately sample from π, or approximate $\mathbb{E}_{x\sim\pi}[f(x)]$ (π and f can be very complicated).

Approach:
Construct a Markov chain with stationary distribution $\pi(x)$, in such a way that the m.c. is easy to sample from, and then rely on the ergodic theorem to calculate the decided approximations.

Terminology:

- Proposal matrix := stochastic matrix (i.e., all entries are non-negative and all raws sun to one) ($Q = Q_{ab}, a, b \in \mathcal{X}$ with $Q_{ab} = Q(a, b)$).
- $\pi(x) = \frac{\tilde{\pi}(x)}{z}$, $z > 0$. The Metropolis algorithm only requires us to evaluate the un-normalized distribution $\tilde{\pi}(x)$.

Algorithm:

- (1) Choose a symmetric proposal matrix Q (Note that for Metropolis-Hasting, Q needs not be symmetric).
- (2) Initialize a state $x_0 \in \mathcal{X}$.
- (3) For $i = 0, 1, 2, ..., n-1$:
  - Sample x (proposal)from $Q(x_i, x)$, i.e. $p(x|x_i) = Q(x_i, x)$.
  - Sample u from a uniform distribution $U(0, 1)$.
  - Accept or reject according to the (random) Metropolis rule:
    If $u < \frac{\tilde{\pi}(x)}{\tilde{\pi}(x_i)}$ then $x_{i+1} = x$, else $x_{i+1} = x_i$.
- (4) Output $x_0, ..., x_{n-1}$
  Then $\mathbb{E}_{x\sim\pi}[f(x)] \approx \frac{1}{n}\sum_{k=0}^{n-1} f(x_k)$, and x_{n-1} is an approximate sample from $\pi(x)$.

Example of the Metropolis Algorithm ("Hard-disks in a box model" 1953)

A box of non-overlapping rigid particles (molecules) of fixed radius.

- N particles
- periodic boundary conditions
- a theoretical model for phase transitions

The goal is to find the corresponding equation of state: e.g. for ideal gases: $PV = nRT$.

Approach

Assume a system configuration $x = (r_1, s_1, ..., r_N, s_N) \in \mathbb{R}^{2N} \to [0, 1]^{2N}$.
Assume a probability distribution over all possible states:

$$\pi(x) = \frac{1}{z} e^{-\frac{\epsilon(x)}{kT}} \cdot I_{\{x\}}, \to \text{the Boltzmann distribution.} \tag{8.12}$$

$$\tilde{\pi}(x) = e^{-\frac{\epsilon(x)}{kT}} \cdot I_{\{x\}}, \to \text{Un-normalized distribution.} \tag{8.13}$$

where $I_{\{x\}}$ means x has to be valid.

- $\epsilon(x)$: Energy of the system (easy to compute for a given state).
- T: Temperature of the system.
- k: Boltzmann constant.
- z: Normalizing constant (partition function) $z = \int e^{-\frac{\epsilon(x)}{kT}} \pi(x) dx \rightarrow$ very high-dimensional integral.

Use the Metropolis algorithm to simulate this system, to find the equation of state.
We want to compute expectations with respect to $\pi(x)$.
To choose a T and N (also the volume is known), and we want to estimate the pressure, which is a function of x, i.e., $\mathbb{E}_{x\sim\pi(x)}[f(x)]$.

Metropolis algorithm

- Choose a symmetric proposal matrix Q: Uniformly randomly choose a^k particle, uniformly sample a point in the box, i.e., $Q(x, x') = \frac{1}{N}\mathbf{1}_{\{(r_l,s_i)=(r'_l,s'_i)\forall i \text{ except } i=k,|r_k-r'_k|\leq\alpha,|s_k-s'_k|\leq\alpha\}}/C$, where C: # of points in the $2\alpha \times 2\alpha$ box. * for $x = x'$, $Q(x, x) = \frac{1}{C}$.
- Iterate:
  - Sample a new state from the proposal distribution.
  - For $i = 0, ..., n-1$ sample a u, uniform(0, 1).
  - Evaluate: $\tilde{\pi}(x) = e^{-\frac{\epsilon(x)}{kT}}\mathbf{1}_{\{x \text{is valid}\}}$, and check if $u < \frac{\tilde{\pi}(x)}{\tilde{\pi}(x_i)}$ $\begin{cases} Yes, & x_{i+1} = x \\ No, & x_{i+1} = x_i \end{cases}$
  - 123

8.10 Gibbs Sampling

Suppose we have parameters $\theta = (\theta_1, \theta_2, ..., \theta_d)$ and some data x.

Our goal is to find the posterior distribution $p(\theta|x)$.
Gibbs sampling allows us to generate samples from the posterior as:

- (1) Pick some initial $\theta^{(i)} = (\theta_1^{(i)}, \theta_2^{(i)}, ..., \theta_d^{(i)})$.
- (2) Sample $\theta_1^{(i+1)} \sim p(\theta_1|\theta_2^{(i)}, ..., \theta_d^{(i)}, x)$
  $\theta_2^{(i+1)} \sim p(\theta_2|\theta_1^{(i)}, \theta_3^{(i)}, ..., \theta_d^{(i)}, x)$
  $\theta_3^{(i+1)} \sim p(\theta_3|\theta_1^{(i)}, \theta_2^{(i)}, \theta_4^{(i)}, ..., \theta_d^{(i)}, x)$
  .
  .
  .
  $\theta_d^{(i+1)} \sim p(\theta_d|\theta_1^{(i)}, \theta_2^{(i)}, ..., \theta_{d-1}^{(i)}, x)$
- (3) Increment $i = i + 1$, and repeat M times to draw M samples.

Pros: Does not require tuning any parameters (e.g., vs mcmc that requires choosing a proposal distribution).

Cons: Assumes knowledge of the conditional densities which may be hard to derive in practice.

Bayesian Linear Regression

$$y_i \sim \mathcal{N}(y_i|w_0 + w_1 x, \gamma^{-1}) \iff y_i = w_0 + w_1 x_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \gamma^{-1}). \tag{8.14}$$

$$\text{Likelihood}: \ \mathcal{L}(y_1, ..., y_n, x_1, ..., x_n | w_0, w_1, \gamma) = \prod_{i=1}^{n} \mathcal{N}(y_i | w_0 + w_1 x_i, \gamma^{-1}) \tag{8.15}$$

$$\text{Priors}: \ w = (w_0, w_1) \sim \mathcal{N}(\begin{bmatrix}\mu_0 \\ \mu_1\end{bmatrix}, \begin{bmatrix}\lambda_0^{-1} & 0 \\ 0 & \lambda_1^{-1}\end{bmatrix}), \ \theta := \{w_0, w_1, \gamma\}, \ \gamma \sim \text{Gamma}(\gamma|a, \theta) \tag{8.16}$$

General approach:

- (i) Write down the posterior conditional density in log form.
- (ii) Throw away all terms that don't depend on the current sampling variable.
- (iii) Pretend this is the density for your variable of interest and all other variables are fixed.

Gibbs update for w_0

$$p(w_0|w_1, \gamma, x, y) \propto p(y|x, w_0, w_1, \gamma) p(w_0) \tag{8.17}$$

where $p(y|x, w_0, w_1, \gamma)$ is the likelihood and $p(w_0) = \mathcal{N}(\mu_0, \gamma_0^{-1})$.

$$\log p(w_0|w_1, \gamma, x, y) \propto -\frac{\gamma}{2} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 - \frac{\lambda_0}{2}(w_0 - \mu_0)^2 \tag{8.18}$$

$$\propto \gamma \sum_{i=1}^{n} (y_i - w_1 x_i) w_0 - \frac{\lambda_0}{2} w_0^2 + \lambda_0 \mu_0 w_0 - \frac{\gamma}{2} n w_0^2 \tag{8.19}$$

Expand out and drop all terms that don't depend on w_0.
This implies that:

$$p(w_0|w_1, \gamma, x, y) \sim \mathcal{N}(w_0 | \frac{\lambda_0 \mu_0 + \gamma \sum_{i=1}^{n} (y_i - w_1 x_i)}{\lambda_0 + n\gamma}, (\lambda_0 + n\gamma)^{-1}) \tag{8.20}$$

Gibbs update for w_1

Similarly for w_1:

$$p(w_1|w_0, \gamma, x, y) \propto p(y|x, w_0, w_1, \gamma) p(w_1) \tag{8.21}$$

where $p(y|x, w_0, w_1, \gamma)$ is the likelihood and $p(w_1) = \mathcal{N}(\mu_1, \gamma_1^{-1})$. Still we expand the log likelihood and drop all terms that don't depend on w_1.

$$\log p(w_1|w_0, \gamma, x, y) \propto -\frac{\gamma}{2} \sum_{i=1}^{n} (y_i - w_0 - w_1 x_i)^2 - \frac{\lambda_1}{2}(w_1 - \mu_1)^2 \tag{8.22}$$

$$\propto \gamma \sum_{i=1}^{n} (y_i - w_1 x_i) w_1 x_i - \frac{\lambda_1}{2} w_1^2 + \lambda_1 \mu_1 w_1 - \frac{\gamma}{2} \sum_{i=1}^{n} w_0^2 x_i^2 \tag{8.23}$$

This implies that:

$$p(w_1|w_0, \gamma, x, y) \sim \mathcal{N}(w_1 | \frac{\lambda_1 \mu_1 + \gamma \sum_{i=1}^{n} (y_i - w_0) x_i}{\lambda_1 + \gamma \sum_{i=1}^{n} x_i^2}, (\lambda_1 + \gamma \sum_{i=1}^{n} x_i^2)^{-1}) \tag{8.24}$$

Gibbs update for γ

Recall that:

$$\text{Gamma}(x|\alpha,\beta) \propto \beta^\alpha x^{\alpha-1} e^{-\beta x}, \quad \log \text{Gamma}(x|\alpha,\beta) \propto (\alpha-1)\log(x) - \beta x. \tag{8.25}$$

$$p(\gamma|w_0,w_1,x,y) \propto p(y|x,w_0,w_1,\gamma)p(\gamma) \tag{8.26}$$

$$\propto \frac{n}{2}\log(\gamma) - \frac{\gamma}{2}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2 + (\alpha-1)\log(\gamma) - \beta\gamma \tag{8.27}$$

This implies that:

$$p(\gamma|w_0,w_1,x,y) \sim \text{Gamma}(\alpha - 1 + \frac{n}{2}, \beta + \frac{1}{2}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)^2). \tag{8.28}$$

Chapter 9

Gaussian processes, multi-output Gaussian processes and multi-fidelity modeling

9.1 Gaussian Processes (recap)

Prior

$$y = f(x) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_n^2 I), x \in \mathbb{R}^d, y \in \mathbb{R} \tag{9.1}$$

$$f(x) \sim \mathcal{GP}(0, K(x, x'; \theta)) \rightarrow \begin{bmatrix} f(x) \\ f(x') \end{bmatrix} \sim \mathcal{GP}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(x,x) & k(x,x') \\ & k(x',x') \end{bmatrix}) \tag{9.2}$$

$$k(x, x'; \theta) = \sigma_f^2 \exp(-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i - x_i')^2}{\theta_i^2}) \tag{9.3}$$

$$\theta := \{\theta, \sigma_n^2\} = \{\sigma_f^2, \theta_1, ...\theta_d, \sigma_n^2\} \tag{9.4}$$

Training

Data $\{X, y\}, X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^{n \times 1}$

$$p(y|x) = \mathcal{N}(0, k(X, X; \theta) + \sigma_n^2 I), \quad K = k(X, X; \theta) + \sigma_n^2 I \tag{9.5}$$

$$\Rightarrow -\log p(y|X) = \frac{1}{2} y^T K^{-1} y + \frac{1}{2}\log|K| + \frac{n}{2}\log 2\pi \tag{9.6}$$

K is $n \times n$, full, symmetric positive definite $\Rightarrow O(n^3)$. Gradient $\nabla_\theta \log p(y|x)$ can be computed analytically or through automatic differentiation.

Prediction/Posterior (using the optimal trained θ^*)

$$\begin{bmatrix} f(x^*) \\ y \end{bmatrix} \sim \mathcal{GP}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} k(x^*,x^*) & k(x^*,X) \\ & k(X,X) \end{bmatrix}) \tag{9.7}$$

$$\Rightarrow -\log p(f(x^*)|X, y) = \mathcal{N}(k(x^*,x)K^{-1}y, k(x^*,x^*) - k(x^*,X)K^{-1}k(X,x^*)) \tag{9.8}$$

where $\mu(x^*) = k(x^*,x)K^{-1}y$ and $\Sigma(x^*,x^*) = k(x^*,x^*) - k(x^*,X)K^{-1}k(X,x^*)$.

9.2 Gaussian Processes (A Bayesian non-parametric approach to non-linear regression)

Recall linear regression with basis function:

$$y_i = w^T\varphi(x_i), y_i \in \mathbb{R}, x_i \in \mathbb{R}^d, \varphi : \mathbb{R}^d \to \mathbb{R}^m, i = 1, ..., n, \quad i.e., \varphi(x) = (\varphi_1(x), ..., \varphi_m(x)) \tag{9.9}$$

Minimizing the quadratic error with least squares or via MLE, we can obtain the optimal w^*:

$$w^* = (\phi^T\phi)^{-1}\phi^T y, \text{ where } \phi = \begin{bmatrix} \varphi_1(x_1) & \dots & \varphi_m(x_1) \\ \vdots & & \vdots \\ \varphi_1(x_n) & \dots & \varphi_m(x_n) \end{bmatrix}, \ y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{9.10}$$

Say now we want to predict at a new x^*:

$$y^* = w^{T*}\varphi(x^*) = [(\phi^T\phi)^{-1}\phi^T y]^T\varphi(x^*) = \varphi(x^*)^T(\phi^T\phi)^{-1}\phi^T y \tag{9.11}$$

The elements of $K = \phi^T\phi$ are:

$$K_{ij} = \sum_{i=1}^{m} \varphi_m(x_i)\varphi_m(x_j) \xrightarrow[m\to\infty]{\text{let}} K(x, x') = \int_0^\infty \varphi_m^T(x)\varphi_m(x)dm \tag{9.12}$$

According to Mercer's theorem, $K = \phi^T\phi$ (parametric) and $K(x, x') = \phi\phi^T$ (non-parametric) is the Mercer kernel. The first one, data is stored/summarized in a set of parameters. The second one, the data acts like parameters.

$$y = f(x) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_n^2 I), \tag{9.13}$$

$$f(x) \sim \mathcal{GP}(f|0, K(x, x'; \theta)) \tag{9.14}$$

$$\tag{9.15}$$

Bayes rule:

$$p(f|y, X) = \frac{p(y|f, X)p(f)}{p(y|X)}, \ p(y|X) = \int p(y|X, f)p(f) \tag{9.16}$$

$$X \in \mathbb{R}^{n\times d} \begin{bmatrix} x_{11} & \dots & x_{1d} \\ \vdots & & \vdots \\ x_{m1} & \dots & x_{nd} \end{bmatrix}, \ y \in \mathbb{R}^{n\times 1} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \tag{9.17}$$

$$p(f|X) = \mathcal{N}(0, K) \Rightarrow \log p(f|X) = -\frac{1}{2}f^T K^{-1} f - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{9.18}$$

$$p(y|f, X) = \mathcal{N}(f, \sigma_n^2 I) \xrightarrow[f]{\text{marginalize}} p(y|X) = \int p(y|X, f)p(f)df \tag{9.19}$$

$$\Rightarrow \log p(y|X) = -\frac{1}{2}\log|K + \sigma_n^2 I| - \frac{1}{2}y^T(K + \sigma_n^2 I)^{-1}y - \frac{n}{2}\log 2\pi \tag{9.20}$$

9.3 Multi-output Gaussian Process Regression

Prior

$$x \in \mathbb{R}^d, y_1 \in \mathbb{R}, y_2 \in \mathbb{R} \tag{9.21}$$

$$y_1 = f_1(x) + \epsilon_1, \quad f_1(x) \sim \mathcal{GP}(0, K_1(x, x'; \theta_1)) \tag{9.22}$$

$$y_2 = f_2(x) + \epsilon_2, \quad f_2(x) \sim \mathcal{GP}(0, K_2(x, x'; \theta_2)) \tag{9.23}$$

$$c(x) \sim \mathcal{GP}(0, K_c(x, x'; \theta_1)) \tag{9.24}$$

$$\epsilon_1 \sim \mathcal{N}(0, \sigma_{n1}^2), \ \epsilon_2 \sim \mathcal{N}(0, \sigma_{n2}^2) \tag{9.25}$$

$\theta := \{\sigma_{f1}^2, \theta_1^{(1)}, ..., \theta_d^{(1)}, \sigma_{f2}^2, \theta_1^{(2)}, ..., \theta_d^{(2)}, \sigma_c^2, \theta_1^{(c)}, ..., \theta_d^{(c)}, \sigma_{n1}^2, \sigma_{n2}^2\}$

Training

Data: $\{X_1, y_1\}, \{X_2, y_2\}$

$$p(y_1, y_2|X_1, X_2) \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{11}(X_1, X_1') & K_{12}(X_1, X_2')\\ & K_{22}(X_2, X_2')\end{bmatrix}) \tag{9.26}$$

$$K_{11}(X_1, X_1') := k_1(X_1, X_1'; \theta_1) + \sigma_{n1}^2 I \tag{9.27}$$

$$K_{12}(X_1, X_2') := k_c(X_1, X_2'; \theta_c) \Rightarrow -\log p(y_1, y_2|X_1, X_2) = \frac{1}{2} y^T K^{-1} y + \frac{1}{2}\log|K| + \frac{n_1+n_2}{2}\log 2\pi \tag{9.28}$$

$$K_{22}(X_2, X_2') := k_2(X_2, X_2'; \theta_2) + \sigma_{n1}^2 I, \quad \text{where } y := \begin{bmatrix}y_1\\y_2\end{bmatrix} \tag{9.29}$$

Prediction

$$\begin{bmatrix}f_1(x^*)\\ \begin{bmatrix}y_1\\y_2\end{bmatrix}\end{bmatrix} \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{11}(x^*, x^*) & K_{11}(x^*, X_1) & K_{12}(x^*, X_2)\\ & K & \end{bmatrix}) \tag{9.30}$$

$$\Rightarrow p(f_1(x^*|y)) = \mathcal{N}([K_{11}(x^*, X_1)K_{12}(x^*, X_2)]K^{-1}y, K_{11}(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)) \tag{9.31}$$

$$\text{where} \quad k(x^*, X) = [K_{11}(x^*, X_1)K_{12}(x^*, X_2)] \tag{9.32}$$

$$\begin{bmatrix}f_2(x^*)\\ \begin{bmatrix}y_1\\y_2\end{bmatrix}\end{bmatrix} \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{22}(x^*, x^*) & K_{12}(x^*, X_1) & K_{22}(x^*, X_2)\\ & K & \end{bmatrix}) \tag{9.33}$$

$$\Rightarrow p(f_2(x^*|y)) = \mathcal{N}([K_{12}(x^*, X_1)K_{22}(x^*, X_2)]K^{-1}y, K_{22}(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)) \tag{9.34}$$

$$\text{where} \quad k(x^*, X) = [K_{12}(x^*, X_1)K_{22}(x^*, X_2)] \tag{9.35}$$

$$\tag{9.36}$$

9.4 Multi-fidelity Gaussian Process Regression

Prior

Data $x \in \mathbb{R}^d, y_L \in \mathbb{R}, y_H \in \mathbb{R}$.

$$y_L = f_L(x) + \epsilon_L, \quad f_L(x) \sim \mathcal{GP}(0, K_L(x, x'; \theta_L)) \tag{9.37}$$

$$y_H = f_H(x) + \epsilon_H, \quad f_H(x) = \rho f_L(x) + \delta(x) \tag{9.38}$$

$$\delta(x) \sim \mathcal{GP}(0, K_H(x, x'; \theta_H)) \quad \delta(x) \perp f_L(x) \tag{9.39}$$

$\theta := \{\sigma_f^2, \theta_1^L, ..., \theta_d^L, \sigma_{f_H}^2, \theta_1^H, ..., \theta_d^H, \rho, \sigma_{n_L}^2, \sigma_{n_H}^2\}$.

Training

Data: $\{x_L, y_L\}, \{x_H, y_H\}, n_L \gg n_H$.

$$p(y_L, y_H|X_L, X_H) \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{LL}(X_L, X_L'; \theta_{LL}) & K_{LH}(X_L, X_H'; \theta_{LH})\\ & K_{HH}(X_H, X_H'; \theta_{HH})\end{bmatrix}) \tag{9.40}$$

$$K_{LL}(X_L, X_L'; \theta_{LL}) = k_L(X_L, X_L'; \theta_L) + \sigma^2_{n_L} I, \quad \theta_{LL} = \{\theta_L, \sigma_n^2\} \tag{9.41}$$

$$K_{LH}(X_L, X_H'; \theta_{LH}) = \rho k_L(X_L, X_H'; \theta_L), \quad \theta_{LH} = \{\theta_L, \rho\} \tag{9.42}$$

$$K_{HH}(X_H, X_H'; \theta_{HH}) = \rho^2 k_L(X_H, X_H'; \theta_L) + k_H(X_H, X_H'; \theta_H) + \sigma^2_{n_H} I, \quad \theta_{HH} = \{\theta_L, \theta_H \sigma^2_{n_H}, \rho\} \tag{9.43}$$

$$\Rightarrow -\log p(y_L, y_H|X_1, X_2) = \frac{1}{2} y^T K^{-1} y + \frac{1}{2}\log|K| + \frac{n_L + n_H}{2}\log 2\pi \tag{9.44}$$

where $y = \begin{bmatrix}y_L\\y_H\end{bmatrix}$.

Prediction

$$\begin{bmatrix}f_L(x^*)\\ \begin{bmatrix}y_L\\y_H\end{bmatrix}\end{bmatrix} \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{LL}(x^*, x^*) & K_{LL}(x^*, X_L) & K_{LH}(x^*, X_H)\\ & K & \end{bmatrix}) \tag{9.45}$$

$$\Rightarrow p(f_L(x^*|y)) = \mathcal{N}([K_{LL}(x^*, X_L) K_{LH}(x^*, X_H)] K^{-1} y, K_{LL}(x^*, x^*) - k(x^*, X) K^{-1} k(X, x^*)) \tag{9.46}$$

$$\begin{bmatrix}f_H(x^*)\\ \begin{bmatrix}y_L\\y_H\end{bmatrix}\end{bmatrix} \sim \mathcal{N}(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}K_{HH}(x^*, x^*) & K_{LH}(x^*, X_L) & K_{HH}(x^*, X_H)\\ & K & \end{bmatrix}) \tag{9.47}$$

$$\Rightarrow p(f_H(x^*|y)) = \mathcal{N}([K_{LH}(x^*, X_L) K_{HH}(x^*, X_L)] K^{-1} y, K_{HH}(x^*, x^*) - k(x^*, X) K^{-1} k(X, x^*)) \tag{9.48}$$

9.5 Gaussian Process Classification (Binary case)

$$\pi(x) := p(y = +1|x) = \sigma(f(x)), \quad f(x)\mathcal{GP}(0, K(x, x'; \theta)) \tag{9.49}$$

Given $\mathcal{D} = ((x_1, y_1), ..., (x_n, y_n)), x_i \in \mathbb{R}^d, y_i \in \{+1, -1\}, i = 1, ...n$ we are interested in predicting $\pi(x^*)$.

Bayes rule

$$p(f|X, y) = \frac{p(y|f)p(f|X)}{p(y|X)} \tag{9.50}$$

Since the denominator is independent of f, MAP estimation corresponds to maximizing $\log p(y|f) + \log p(f|X)$. Since the posterior is intractable we will seek a Gaussian approximation using the Laplace approximation. To this end:

$$\psi(f) := \log p(y|f) + \log p(f|X) = \log p(y|f) - \frac{1}{2} f^T K^{-1} f - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{9.51}$$

$$\nabla_f \psi(f) = \nabla_f \log p(y|f) - K^{-1} f \tag{9.52}$$

$$\nabla^2_f \psi(f) = \nabla^2_f \log p(y|f) - K^{-1} = -W - K^{-1}, \tag{9.53}$$

where $W := -\nabla^2_f \log p(y|f)$ is a diagonal Hessian matrix since the likelihood factorizes (i.e., the data y_i are independent once cnoditioned on f_i).

For binary classification, we can use the logit likelihood:

$$\log p(y_i|x_i) = -\log \frac{1}{1+e^{-y_i f_i}} \quad \text{with gradients} \tag{9.54}$$

$$\frac{\partial}{\partial f_i} \log p(y_i|f_i) = t_i - \pi_i, \quad \frac{\partial^2}{\partial f_i^2} \log p(y_i|f_i) = -\pi_i(1-\pi_i), \tag{9.55}$$

where $\pi_i := p(y_i = 1|f_i)$ and $t = \frac{y+1}{2}$.
$\psi(f)$ is maximized at: $\nabla_f \psi(f) = 0 \Rightarrow \hat{f} = K(\nabla \log p(y|\hat{f}))$.
This is a non-linear equation that we can solve using Newton's method:

$$f_{m+1} = f_m - (\nabla_f^2 \psi)^{-1} \nabla_f \psi = f_m + (K^{-1} + w)^{-1}(\nabla_f \log p(y|f_m) - K^{-1} f_m) \tag{9.56}$$

$$f_{m+1} = (K^{-1} + W)^{-1}(W f_m + \nabla_f \log p(y|f_m)). \tag{9.57}$$

Then the approximate Gaussian posterior takes the form:

$$p(f|X,y) \approx q(f|X,y) = \mathcal{N}(\hat{f}, (K^{-1}+W)^{-1}) \tag{9.58}$$

Posterior predictions

$$\mathbb{E}_q[f^*|X,y,x^*] = K(x,x^*)^T K^{-1} \hat{f} = K(x,x^*)^T \nabla_f \log p(y|\hat{f}) \tag{9.59}$$

$$\mathbb{V}\text{ar}_q[f^*|X,y,x^*] = K(x^*,x^*) - K(x,x^*)^T (K + W^{-1})^{-1} K(x^*,x) \tag{9.60}$$

Averaged predictive probability

$$\pi(x^*) \simeq \mathbb{E}_q[\pi(x^*)|X,y,x^*] = \int \sigma(f^*) q(f^*|X,y,x^*) df^* \tag{9.61}$$

due to the sigmoid nonlinearity this is different than the sigmoid of the expectation of f:

$$\pi(x^*) = \sigma(\mathbb{E}_q[f^*|X,y,x^*]) \tag{9.62}$$

Practical implementation

Many computations can be expressed in terms of the symmetric positive definite matrix: $B = I + W^{1/2} K W^{1/2}$, which can be computed in $O(n^2)$ cost since W is diagonal.

Then:

- $(K^{-1}+W)^{-1} = K - K W^{1/2} B^{-1} W^{1/2} K$
- $(K + W^{-1})^{-1} = W^{1/2} B^{-1} W^{1/2} K$

and $\mathbb{V}\text{ar}_q[f^*|X,y,x^*] = K(x^*,x^*) - v^T v$, where $v := L \setminus (W^{1/2} K(x^*,x^*))$.

Training

For training we need to compute a Laplace approximation to the marginal likelihood:

$$p(y|X) = \int p(y|f) p(f|X) df = \int \exp(\psi(f)) df \tag{9.63}$$

Using a Taylor expansion of $\psi(f)$ locally around the mode $\hat{f}$ we obtain:

$$\psi(f) \approx \psi(\hat{f}) - \frac{1}{2}(f-\hat{f})^T A (f-\hat{f}) \tag{9.64}$$

Therefore an approximation to the marginal likelihood $q(y|X)$ can be obtained as:

$$p(y|X) \approx q(y|X) = \exp(\psi(\hat{f})) \int \exp[-\frac{1}{2}(f-\hat{f})A(f-\hat{f})]df \tag{9.65}$$

The Gaussian integral can be evaluated analytically to obtain:

$$-\log q(y|X) = \frac{1}{2}\hat{f}^T K f - \log p(y|\hat{f}) + \frac{1}{2}\log|B|. \tag{9.66}$$

Chapter 10

Bayesian optimization and active learning

10.1 Active Learning

Given a trained GP model with predictive posterior distribution:

$$p(f(x^*)|X, y, x^*) = \mathcal{N}(\mu(x^*), \Sigma(x^*)) \tag{10.1}$$

We can acquire new data points in order to minimize the posterior uncertainty:

$$x_{n+1} = \arg\max_x \Sigma(x) \tag{10.2}$$

where

$$\Sigma(x) := K(x, x) - K(x, X)(K + \sigma_n^2 I)^{-1} k(X, x) \tag{10.3}$$

10.2 Bayesian Optimization

$$x_{n+1} = \arg\max_x \alpha(x; \mathcal{D}) \tag{10.4}$$

Termination criterion

- Maximum number of iterations
- Distance between consecutive samples, i.e., $\|x_{n+1} - x_n\| < \epsilon$
- Threshold on acquisition function, i.e., $\max \alpha(x; \mathcal{D}) < \epsilon$

Sampling from a GP

$$S(x) = \mu(x) + Lz, \quad \text{, where } \Sigma(x) = LL^T, \quad \text{and } z \sim \mathcal{N}(0, I) \tag{10.5}$$

Bayesian Optimization Acquisition Functions

Probability of improvement

Let $f_{min} := \min(x)$, i.e., the minimum value observed so far.
We want to evaluate $f(x)$ at the point that is most likely to improve this value.
This corresponds to the following utility function:

$$u(x) = \begin{cases} 0, & f(x) > f_{min}, \\ 1, & f(x) \leq f_{min}. \end{cases} \tag{10.6}$$

The probability of improvement acquisition function is the expected utility:

$$\alpha_{PI}(x;\mathcal{D}) = \mathbb{E}[u(x)|x,\mathcal{D}] = \int_{-\infty}^{f_{min}} \mathcal{N}(f|\mu(x),K(x,x))df = \Phi(f_{min}|\mu(x),K(x,x)) \tag{10.7}$$

where $\Phi(f_{min}|\mu(x),K(x,x))$ is the Gaussian cdf.

Expected improvement

$$u(x) = \max\{0, f_{min} - f(x)\} \tag{10.8}$$

$$\text{Then, } \alpha_{EI} = \mathbb{E}[u(x)|x,\mathcal{D}] = \int_{-\infty}^{f_{min}} (f_{min} - f)\mathcal{N}(f|\mu(x),K(x,x))df \tag{10.9}$$

$$= (f_{min} - \mu(x))\phi(f_{min}|\mu(x),K(x,x)) + K(x,x)\mathcal{N}(f_{min}|\mu(x),K(x,x)) \tag{10.10}$$

where $\phi(f_{min}|\mu(x),K(x,x))$ is the normal cdf and $\mathcal{N}(f_{min}|\mu(x),K(x,x))$ is the normal pdf.

$$x_{n+1} = \arg\max_{x\in\mathcal{X}} \alpha(x,\mathcal{D}) \tag{10.11}$$

Entropy search

We seek to minimize the uncertainty we have in the location of the optimal value:

$$x^* = \arg\min_{x\in\mathcal{X}} f(x) \tag{10.12}$$

Our belief over f induces a distribution over x^*, i.e., $p(x^*|\mathcal{D})$, however, there is no closed form for this distribution.
Entropy search seeks to evaluate points so as to minimize the entropy of the induced distribution $p(x^*|\mathcal{D})$. The utility measure corresponds to the reduction in entropy given a new measurement:

$$u(x) = \mathbb{H}[x^*|\mathcal{D}] - \mathbb{H}[x^*|\mathcal{D},x,f(x)] \tag{10.13}$$

$$\alpha_{ES}(x;\mathcal{D}) = \mathbb{E}[u(x)|x,\mathcal{D}] \tag{10.14}$$

To compute this, a series of approximations need to be made.

Chapter 11

Probabilistic scientific computing: Gaussian process regression meets differential equations

11.1 Gaussian Processes meets differential equations

Setup

Given a linear operator: $\mathcal{L}_x u(x) = f(x)$ and data $\{x_u, y_u\}, \{x_f, y_f\}, x \in \mathbb{R}^d, y_u, y_f \in \mathbb{R}$.

- $\{x_u, y_u\}$ are a few measurements of $u(x)$ (could be boundary or initial data, or not) $y_u = u(x) + \epsilon_u$
- $\{x_f, y_f\}$ are a few measurements of $f(x)$ (both $u(x)$ and $f(x)$ are unknown black-box functions) $y_u = u(x) + \epsilon_u$

Model

$$u(x) \sim \mathcal{GP}(0, K_{uu}(x, x'; \theta)) \tag{11.1}$$

$$\Rightarrow f(x) \sim \mathcal{GP}(0, K_{ff}(x, x'; \theta)), \text{ with } K_{ff}(x, x'; \theta) = \mathcal{L}_x \mathcal{L}_{x'} K_{uu}(x, x'; \theta)) \tag{11.2}$$

$$\epsilon_u \sim \mathcal{N}(0, \sigma_{nu}^2),\ \epsilon_f \sim \mathcal{N}(0, \sigma_{nf}^2), \quad \theta := \{\sigma_f^2, \theta_1, ..., \theta_d, \sigma_{nu}^2, \sigma_{nf}^2\} \tag{11.3}$$

Training

$$\begin{bmatrix} y_u \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{uu} & K_{uf} \\ & K_{ff} \end{bmatrix}), X = \begin{bmatrix} x_u \\ x_f \end{bmatrix} \tag{11.4}$$

And

$$K_{uu} = K_{uu}(x_u, x_u'; \theta) + \sigma_{nu}^2 I \tag{11.5}$$

$$K_{uf} = \mathcal{L}_{x'} K_{uu}(x_u, x_f'; \theta) \Rightarrow -\log p(y|X) = \frac{1}{2} y^T K^{-1} y + \frac{1}{2} \log |K| + \frac{n_u + n_f}{2} \log 2\pi \tag{11.6}$$

$$K_{ff} = \mathcal{L}_x \mathcal{L}_{x'} K_{uu}(x_f, x_f'; \theta) + \sigma_{nf}^2 I \tag{11.7}$$

Prediction

$$\begin{bmatrix} u(x^*) \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{uu}(x^*.x^*) & K_{uu}(x^*, X) \\ & K \end{bmatrix}) \tag{11.8}$$

$$\Rightarrow p(u(x^*)|X, y, x^*) = \mathcal{N}(K_{uu}(x^*, X) K^{-1} y, K_{uu}(x^*, x^*) - k_{uu}(x^*, X) K^{-1} k_{uu}(X, x^*)) \tag{11.9}$$

where $\mu_u(x^*) = K_{uu}(x^*, X)K^{-1}y$ and $\Sigma_u(x^*) = K_{uu}(x^*, x^*) - k_{uu}(x^*, X)K^{-1}k_{uu}(X, x^*)$.
Similarly,

$$\begin{bmatrix} f(x^*) \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{ff}(x^*.x^*) & K_{fu}(x^*, X) \\ & K \end{bmatrix}) \tag{11.10}$$

$$\Rightarrow p(f(x^*)|X, y, x^*) = \mathcal{N}(K_{fu}(x^*, X)K^{-1}y, K_{ff}(x^*, x^*) - k_{fu}(x^*, X)K^{-1}k_{fu}(X, x^*)) \tag{11.11}$$

11.2 Machine Learning of Linear Differential Equations

Setup

Given data $\{x_u, y_u\}, \{x_f, y_f\}$, we want to learn a model:

$$y_u = u(x_u) + \epsilon_u \quad u(x) \sim \mathcal{GP}(0, K(x, x; \theta)) \tag{11.12}$$

$$y_f = f(x_f) + \epsilon_f \quad f(x) \sim \mathcal{GP}(0, g(x, x; \theta, \lambda)) \tag{11.13}$$

$$\mathcal{L}_x^\lambda u(x) = f(x) \quad \Rightarrow g(x, x; \theta, \lambda) = \mathcal{L}_x^\lambda \mathcal{L}_{x'}^\lambda K(x, x'; \theta) \tag{11.14}$$

Parameters: $\theta = \{\sigma_f^2, \theta_1, ..., \theta_d, \lambda, \sigma_{nu}^2, \sigma_{nf}^2\}$

Training

$$\theta^* = \arg\min - \log p(y|X) = \frac{1}{2}\log|K| + \frac{1}{2}y^T K^{-1} y + \frac{n_u + n_f}{2}\log 2\pi \tag{11.15}$$

$$y = \begin{bmatrix} y_u \\ y_f \end{bmatrix}, X = \begin{bmatrix} x_u \\ x_f \end{bmatrix}, \begin{bmatrix} y_u \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{uu} & K_{uf} \\ & K_{ff} \end{bmatrix}) \tag{11.16}$$

And

$$K_{uu} = K_{uu}(x_u, x_u'; \theta) + \sigma_{nu}^2 I \tag{11.17}$$

$$K_{uf} = \mathcal{L}_{x'} K_{uu}(x_u, x_f'; \theta) \tag{11.18}$$

$$K_{ff} = \mathcal{L}_x \mathcal{L}_{x'} K_{uu}(x_f, x_f'; \theta) \tag{11.19}$$

Prediction

$$\begin{bmatrix} u(x^*) \\ y_u \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(x^*.x^*) & [K(x^*, x_u) & \mathcal{L}_{x'} K(x^*, x_f)] \\ & K & \end{bmatrix}) \tag{11.20}$$

$$\Rightarrow p(u(x^*)|X, y, x^*) = \mathcal{N}([K(x^*, x_u)\mathcal{L}_{x'} K(x^*, x_f)]K^{-1}y, k(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)) \tag{11.21}$$

And,

$$\begin{bmatrix} f(x^*) \\ y_u \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \mathcal{L}_x \mathcal{L}_{x'} K(x^*.x^*) & [\mathcal{L}_x K(x^*, x_u) & \mathcal{L}_x \mathcal{L}_{x'} K(x^*, x_f)] \\ & K & \end{bmatrix}) \tag{11.22}$$

$$\Rightarrow p(f(x^*)|X, y, x^*) = \mathcal{N}([\mathcal{L}_x K(x^*, x_u)\mathcal{L}_x \mathcal{L}_{x'} K(x^*, x_f)]K^{-1}y, \mathcal{L}_x \mathcal{L}_{x'} k(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)) \tag{11.23}$$

where we conclude $\mu_u(x^*) = [K(x^*, x_u)\mathcal{L}_{x'} K(x^*, x_f)]K^{-1}y$, $\Sigma_u(x^*) = k(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)$, $\mu_f(x^*) = [\mathcal{L}_x K(x^*, x_u)\mathcal{L}_x \mathcal{L}_{x'} K(x^*, x_f)]K^{-1}y$ and $\Sigma_f(x^*) = \mathcal{L}_x \mathcal{L}_{x'} k(x^*, x^*) - k(x^*, X)K^{-1}k(X, x^*)$.

A worked out example: 1D Euler-Bernoulli beam

$$EI\frac{d^4}{dx^4}u(x) = f(x), \quad \text{Boundary conditions} \begin{cases} u(0) = 0 \\ u_x(0) = 0 \\ u_{xx}(1) = 0 \\ u_{xxx}(1) = 0 \end{cases} \tag{11.24}$$

In this case, we can define the differential operator:

$$\mathcal{L}_x u(x) = f(x), \quad \mathcal{L}_x := EI\frac{\partial^4}{\partial x^4} = \frac{d^4}{dx^4} \tag{11.25}$$

Model

$$u(x) \sim \mathcal{GP}(0, K(x,x';\theta)) \Rightarrow f(x) \sim \mathcal{GP}(0, g(x,x';\theta)), \rightarrow g(x,x';\theta) = \mathcal{L}_x\mathcal{L}'_x K(x,x';\theta) \tag{11.26}$$

Data

$\{x_u, y_u\}, \{x_f, y_f\}$, i.e., $x_u = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, y_u = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$$y_u = u(x_u), \quad y_f = f(x_f) + \epsilon_f, \quad \epsilon_f \sim \mathcal{N}(0, \sigma_{nf}^2 I) \tag{11.27}$$

Training

$$\begin{bmatrix} y_u \\ y_{ux} \\ y_{uxx} \\ y_{uxxx} \\ y_f \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{uu} & K_{uu'} & K_{uu''} & K_{uu'''} & K_{uf} \\ & K_{u'u'} & K_{u'u''} & K_{u'u'''} & K_{u'f} \\ & & K_{u''u''} & K_{u''u'''} & K_{u''f} \\ & & & K_{u'''u'''} & K_{u'''f} \\ & & & & K_{ff} \end{bmatrix}) \tag{11.28}$$

with

$$K_{uu} = K(x_u^0, x_u^0; \theta) \quad K_{u'u'} = \frac{\partial}{\partial x}\frac{\partial}{\partial x'}K(x_u^1, x_u^1; \theta) \quad K_{u''u''} = \frac{\partial^2}{\partial x^2}\frac{\partial^2}{\partial x'^2}K(x_u^2, x_u^2; \theta) \tag{11.29}$$

$$K_{uu'} = \frac{\partial}{\partial x'}K(x_u^0, x_u^1; \theta) \quad K_{u'u''} = \frac{\partial}{\partial x}\frac{\partial^2}{\partial x'^2}K(x_u^1, x_u^2; \theta) \quad K_{u''u'''} = \frac{\partial^2}{\partial x^2}\frac{\partial^3}{\partial x'^3}K(x_u^2, x_u^3; \theta) \tag{11.30}$$

$$K_{uu''} = \frac{\partial^2}{\partial x'^2}K(x_u^0, x_u^2; \theta) \quad K_{u'u'''} = \frac{\partial}{\partial x}\frac{\partial^3}{\partial x'^3}K(x_u^1, x_u^3; \theta) \quad K_{u''f} = \frac{\partial^2}{\partial x^2}\mathcal{L}_{x'}K(x_u^2, x_f; \theta) \tag{11.31}$$

$$K_{uu'''} = \frac{\partial^3}{\partial x'^3}K(x_u^0, x_u^3; \theta) \quad K_{u'f} = \frac{\partial}{\partial x}\mathcal{L}_{x'}K(x_u^1, x_f; \theta) \quad K_{uf} = \mathcal{L}_{x'}K(x_u^0, x_f; \theta) \tag{11.32}$$

$$K_{u'''u'''} = \frac{\partial^3}{\partial x^3}\frac{\partial^3}{\partial x'^3}K(x_u^3, x_u^3; \theta) \quad K_{u'''f} = \frac{\partial^3}{\partial x^3}\mathcal{L}_{x'}K(x_u^3, x_f; \theta) \quad K_{ff} = \mathcal{L}_x\mathcal{L}_{x'}K(x_f, x_f; \theta) + \sigma_{nf}^2 I \tag{11.33}$$

$$\Rightarrow -\log p(y|X) = \frac{1}{2}\log|K| + \frac{1}{2}y^T K^{-1} y + \frac{n_u + n_f}{2}\log 2\pi \tag{11.34}$$

11.3 Numerical Gaussian Processes

Nonlinear equations and GPs don't go well together...

$$\mathcal{N}_x u(x) = f(x), \quad u(x) \sim \mathcal{GP}(0, K(x, x'; \theta)), \text{ then } f(x) \text{ is not a GP...} \tag{11.35}$$

Key idea: Linearize in time!
e.g., Burger's equation:

$$u_t + uu + x = \nu u_{xx} \Rightarrow u_t = -uu_x + \nu u_{xx}, \quad \text{with initial and boundary conditions} \begin{cases} u(x,0) = \sin(\pi x) \\ u(0,t) = 0 \\ u(1,t) = 0 \end{cases} \tag{11.36}$$

Pick a time-discretization scheme: e.g., backward Euler:

$$\frac{u^n - u^{n-1}}{\Delta t} = -u^n u_x^n + \nu u_{xx}^n \Rightarrow u^n = n^{n-1} - \Delta t u^n u_x^n + \nu \Delta t u_{xx}^n \tag{11.37}$$

$$\text{approximate with the posterior mean of the previous step } \mu^{n-1} \tag{11.38}$$

$$\Rightarrow u^{n-1} = u^n + \Delta t \mu^{n-1} u_x^n - \nu \Delta t u_{xx}^n \tag{11.39}$$

$$\Rightarrow \mathcal{L}_x u^n = u^{n-1} \rightarrow \text{this is the form we like!} \tag{11.40}$$

Then

$$u^n(x) \sim \mathcal{GP}(0, K(x, x'; \theta)) \Rightarrow u^{n-1}(x) \sim \mathcal{GP}(0, \mathcal{L}_x \mathcal{L}_x K(x, x'; \theta)) \tag{11.41}$$

And we can write the likelihood:

$$\begin{bmatrix} u^n \\ u^{n-1} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{u,u}^{n,n} & K_{u,u}^{n,n-1} \\ & K_{u,u}^{n-1,n-1} \end{bmatrix}) \tag{11.42}$$

Where

$$K_{u,u}^{n,n} = K(x_u^n, x_u^n; \theta) \tag{11.43}$$

$$K_{u,u}^{n,n-1} = \mathcal{L}_{x'} K(x_u, x_u^{n-1}; \theta) \tag{11.44}$$

$$K_{u,u}^{n-1,n-1} = \mathcal{L}_x \mathcal{L}_{x'} K(x_u^{n-1}, x_u^{n-1}; \theta) + \sigma_{nu}^2 I \tag{11.45}$$

Prediction

$$\begin{bmatrix} u^n(x^*) \\ u^n \\ u^{n-1} \end{bmatrix} \sim \mathcal{N}(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{u,u}^{n,n}(x^*.x^*) & [K_{u,u}^{n,n}(x^*, x_u^n) & \mathcal{L}_{x'} K(x^*, x_{u^{n-1}})] \\ & K & \end{bmatrix}) \tag{11.46}$$

$$\Rightarrow p(u^n(x^*)|u^n, u^{n-1}, x^*) = \mathcal{N}([K_{u,u}^{n,n}(x^*, x_{u^n}) \mathcal{L}_{x'} K(x^*, x_{u^{n-1}})] K^{-1} \begin{bmatrix} u^n \\ u^{n-1} \end{bmatrix}, \tag{11.47}$$

$$K_{u,u}^{n,n}(x^*, x^*) - K(x^*, X) K^{-1} K(X, x^*) + K(x^*, X) K^{-1} \begin{bmatrix} 0 & 0 \\ 0 & \Sigma^{n-1,n_1} \end{bmatrix} K^{-1} K(X, x^*)) \tag{11.48}$$

where $\mu^n = [K_{u,u}^{n,n}(x^*, x_{u^n}) \mathcal{L}_{x'} K(x^*, x_{u^{n-1}})] K^{-1} \begin{bmatrix} u^n \\ u^{n-1} \end{bmatrix}$ and

$\Sigma^{n,n}(x^*, x^*) = K_{u,u}^{n,n}(x^*, x^*) - K(x^*, X) K^{-1} K(X, x^*) + K(x^*, X) K^{-1} \begin{bmatrix} 0 & 0 \\ 0 & \Sigma^{n-1,n_1} \end{bmatrix} K^{-1} K(X, x^*)$.

Chapter 12

Unsupervised learning: principal component analysis, Gaussian process latent variable models

The January 12, 2019 Version

- *Hotelling transformation.*
- *Karhunen-Loeve decomposition.*
- *Proper orthogonal decomposition.*
- *Singular value decomposition*

12.1 Principal Component Analysis

Setup: Given data $(x_1, ..., x_n), x_i \in \mathbb{R}^d$.
Goal: Encoder the data in a low-dimensional representation: $z_i = f(x_i), z_i \in \mathbb{R}^q, q \ll c$. i.e., project the data on to a low-dimensional subspace.

Maxinum variance formulation (Hotelling 1933)

To begin with consider a 1-dimensional subspace, i.e., $q = 1$.
We can define the coordinate of this subspace $u_1 \in \mathbb{R}^d$ to have unit norm $\|u_1\| = u_1^T u_1 = 1$.
Each data point x_n is projected onto that dimension as $u_1^T x_n$ (scalar quantity).
Hence the mean of all projected data is $u_1^T \bar{x}$, where:

$$\bar{x} := \frac{1}{n} \sum_{i=1}^{n} x_i, \qquad \text{is the sample mean} \tag{12.1}$$

and the variance of the projected data is: $\frac{1}{n} \sum_{i=1}^{n} \{u_1^T x_n - u_1^T \bar{x}\}^2 = u_1^T S u_1$, where S is the sample covariance matrix:

$$S := \frac{1}{n} \sum_{i=1}^{n} (x_n - \bar{x})(x_n - \bar{x})^T. \tag{12.2}$$

We seek the direction u_1 for which captures the most variance in the data, i.e. ("best" summarizes the data).

$$u_1^* = \arg\max_{u_1} \{u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1)\} \tag{12.3}$$

where $\lambda_1(1-u_1^T u_1)$ is the Lagrange multiplier to enforce that u_1 is a unit vector.

Setting the derivative with respect to u_1 to zero we get a critical point:

$$Su_1 = \lambda_1 u_1, \tag{12.4}$$

which implies that u_1 should be an eigenvector of S.
If we left multiply with u_1^T and use that $u_1^T u_1 = 1$, we obtain:

$$u_1^T S u_1 = \lambda_1 \tag{12.5}$$

Hence the variance will be maximum if we set the eigenvector u_1 to be the one that corresponds to the maximum eigenvalue of S.
This eigenvector is known as the first principle component.

How about u_2?

$$u_2^* = \arg\max_{u_2}\{u_2^T S u_2 + \lambda_2(1-u_2^T u_2)\},\ s.t.\ u_2 \perp u_1 \tag{12.6}$$

If we want the q first eigenvectors we can employ efficient techniques such as the power method.

Minimum error formulation

Assume a complete set of orthonormal D-dimensional basis vector $\{u_i\}, i = 1, ..., d$:

$$u_i^T S u_j = \delta_{ij} \tag{12.7}$$

Because this basis is complete, we can represent each data point in $\mathbb{R}^d$ as:

$$x_n = \sum_{i=1}^{d} a_{ni} u_i, \tag{12.8}$$

where the coefficients a_{ni} are different for different data-points.

This simply corresponds to a rotation of the original coordinate system to a new system defined by the $\{u_i\}$, and the original D component $(x_{n1}, ..., x_{nd})$ are now replaced by $(a_{n1}, ..., a_{nd})$.
Taking an inner product with respect to u_j we get: $a_{nj} = x_n^T u_j$. without loss of generality we can write:

$$x_n = \sum_{i=1}^{d} (x_n^T u_i) u_i \tag{12.9}$$

Our goal though is to restrict this representation to a low-dimensional subspace by keeping only the first q basis vectors:

$$\tilde{x}_n = \sum_{i=1}^{q} z_{ni} u_i + \sum_{i=q+1}^{d} b_i u_i \tag{12.10}$$

where z_{ni} depend on each data-point and b_i are constants that are the same for all data-points.

We can now define a loss:

$$J = \frac{1}{n}\sum_{i=1}^{n} \|x_n - \tilde{x}_n\| \tag{12.11}$$

First minimize with respect to $z_{ni} \Rightarrow z_{nj} = x_n^T u_j$ (set derivative to zero and use orthogonality), then with respect to $b_i \Rightarrow b_j = \bar{x}^T u_j, j = q+1, ..., d$.
Using $x_n = \sum_{i=1}^{d} (x_n^T u_i) u_i$, we have:

$$x_n - \tilde{x}_n = \sum_{i=q+1}^{d} \{(x_n - \bar{x}_n)^T u_i\} u_i \tag{12.12}$$

Hence the discrepancy vector $x_n - \tilde{x}_n$ "lives" in the space that is orthogonal to:

$$J = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=q+1}^{d} (x_i^T u_j - \bar{x}^T u_j)^2 = \sum_{j=q+1}^{d} u_j^T S u_j \tag{12.13}$$

Firstly, we need to identify u_j by minimizing the loss. As before this will yield: $Su_i = \lambda_i u_i$, i.e. the principal components correspond to eigenvectors of the sample covariance matrix.

Given the optimal $\{u_i\}$ we can also see that: $J = \sum_{i=q+1}^{d} \lambda_j$.

Practical implementation

Recall SVD:

$$M = U\Sigma V, \quad M \in \mathbb{R}^{n \times d},\ U \in \mathbb{R}^{n \times n},\ \Sigma \in \mathbb{R}^{n \times d},\ W \in \mathbb{R}^{d \times d}. \tag{12.14}$$

where

- U is orthogonal matrix
- Σ is diagonal matrix
- V^T is orthogonal matrix

Given a data matrix $X \in \mathbb{R}^{n \times d}$, we first normalize it to have a zero mean, i.e. $\mathbb{E}[X] = 0$.
Then we construct an unbiased sample covariance matrix: $S = \frac{1}{n} X^T X$.
Compute the eigenvalue decomposition or the SVD of S:

$$S = W \Lambda W^T \tag{12.15}$$

Choose the dimension (q) of the latent space and correspondingly keep the largest q eigenvalues and eigenvectors.
Then the encoding is

$$Z = XW, \qquad Z \in \mathbb{R}^{n \times q},\ X \in \mathbb{R}^{n \times d},\ W \in \mathbb{R}^{d \times q} \tag{12.16}$$

We can also reconstruct X from the encoded values as:

$$X = ZW^T, \qquad X \in \mathbb{R}^{n \times d},\ Z \in \mathbb{R}^{n \times q},\ W \in \mathbb{R}^{d \times q} \tag{12.17}$$

12.2 Probabilistic PCA

We can formulate PCA in a probabilistic setting as:

$$p(z) = \mathcal{N}(z|0, I), \qquad p(x|z) = \mathcal{N}(x|xW^T + \mu, \sigma^2 I) \tag{12.18}$$

and use MLE to determine the unknown model parameters: $\theta := \{W, \mu, \sigma^2\}$.
To do so, we need an expression for the marginal likelihood:

$$p(x) = \int p(x|z)p(z)dz \tag{12.19}$$

Because this is a linear-Gaussian model this has a closed form solution:

$$p(x) = \mathcal{N}(x|\mu, C), \qquad \text{where} C = WW^T + \sigma^2 I \tag{12.20}$$

Since

$$\mathbb{E}[x] = \mathbb{E}[zW^T + \mu + \epsilon] = \mu \tag{12.21}$$

$$cov[x] = \mathbb{E}[(zW^T + \mu + \epsilon)(zW^T + \mu + \epsilon)^T] = \mathbb{E}[Wzz^TW^T] + \mathbb{E}[\epsilon\epsilon^T] = WW^T + \sigma^2 I \tag{12.22}$$

Moreover, we can obtain the posterior over the latent variables as:

$$p(z|x) = \mathcal{N}(z|M^{-1}(X - \mu)W, \sigma^2 M^{-1}), \qquad \text{where} M = W^TW + \sigma^2 I \tag{12.23}$$

To estimate the model parameters we have:

$$-\log p(X|\mu, W, \sigma^2) = -\sum_{i=1}^{n} \log p(x_i|W, \mu, \sigma^2) \tag{12.24}$$

$$= \frac{nd}{2}\log 2\pi + \frac{n}{2}\log|C| + \frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^T C^{-1}(x_i - \mu) \tag{12.25}$$

Then we can obtain:

$$\mu_{ML} = \bar{x}, \Rightarrow -\log p(X|W, \mu, \sigma^2) = \frac{n}{2}\{d\log(2\pi) + \log|C| + Tr(C^{-1}S)\} \tag{12.26}$$

where S is the sample covariance.
Maximizing this with respect to W and σ^2 we can also get:

$$W_{ML} = U_m(L_m - \sigma^2 I)^{1/2}R, \tag{12.27}$$

where U_m is a $d \times q$ matrix whose columns are the eigenvectors of S. L_m is diagonal containing the eigenvalues of S, and R is an arbitrary orthogonal matrix.

$$\sigma^2_{ML} = \frac{1}{d-q}\sum_{i=q+1}^{d}$$

12.3 Tricks of the trade

(1) Variational bounds

We have some loss function $\mathcal{L}(\theta)$ that is intractable to compute (it typically involves some intractable marginalizaiton). We can't evaluate it, let alone minimize it!
e.g., in latent variable models:

$$p(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)} \tag{12.28}$$

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p(z)dz \tag{12.29}$$

In PPCA, we assume that $p_\theta(x|z)$ and $p(z)$ are Gaussian, and the dependence of z on x was linear, i.e.

$$p_\theta(x|z) = \mathcal{N}(x|zW^T + \mu, \sigma^2 I), \quad \theta := \{\mu, W, \sigma^2\} \tag{12.30}$$

Then we could analytically compute

$$p_\theta(x) = \mathcal{N}(x|\mu, C), \quad C := WW^T + \sigma^2 I \tag{12.31}$$

But this won't be the case in general (i.e. $p_\theta(x|z)$ and $p(z)$ could be non-Gaussian and/or the dependence of x on z could be non-linear). Hence $p_\theta(x)$ would be intractable.

Tricks: Introduce some auxiliary parameters φ and a lower bound:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta) \leq \arg\min_{\theta,\varphi} \hat{\mathcal{L}}(\theta, \varphi) \tag{12.32}$$

Ideas to recall:

- Jensen's inequality: $f(\mathbb{E}_{x\sim p(x)}[x]) \leq \mathbb{E}_{x\sim p(x)}[f(x)]$, if f is a convex function.
- Importance sampling: $\mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)p(x)dx = \int [f(x)\frac{p(x)}{q(x)}]q(x)dx \approx \frac{1}{n}\sum_{i=1}^n f(x_i)\frac{p(x_i)}{q(x_i)}$

Formulation: We typically want to minimize:

$$-\log p_\theta(x) = -\log \int p_\theta(x,z)dz = -\log \int \frac{p_\theta(x,z)}{q_\varphi(z|x)} q_\varphi(z|x)dz \tag{12.33}$$

$$= -\log \mathbb{E}_{z\sim q_\varphi(z|x)}[\frac{q_\theta(x,z)}{q_\varphi(z|x)}] \leq -\mathbb{E}_{z\sim q_\varphi(z|x)}[\log \frac{p_\theta(x,z)}{q_\varphi(z|x)}] \tag{12.34}$$

$$= -\int \log \frac{p_\theta(x,z)p(z)}{q_\varphi(z|x)} q_\varphi(z|x)dz = -\int \log p_\theta(x|z) q_\varphi(z|x)dz - \int \log \frac{p(z)}{q_\varphi(z|x)} q_\varphi(z|x)dz \tag{12.35}$$

$$= -\mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] + \mathbb{KL}[q_\varphi(z|x)||p(z)] \geq -\log p_\theta(x) \tag{12.36}$$

where we note that $\hat{\mathcal{L}}(\theta,\varphi) = -\mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] + \mathbb{KL}[q_\varphi(z|x)||p(z)]$.

Assuming parametric models for $q_\varphi(z|x)$ and $p_\theta(x|z)$, this expression is computable, and can be optimized with respect to $\{\theta, \varphi\}$.

Caution: To optimize this objective we need algorithm. A direct Monte Carlo estimate of this is:

$$\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)\nabla_\varphi \log q_\varphi(z|x)] \approx \frac{1}{n}\sum_{i=1}^n \log p_\theta(x_i|z_i)\nabla_\varphi \log q_\varphi(z_i|x_i) \tag{12.37}$$

where we have used that:

$$\nabla_\varphi \log q_\varphi(z|x) = \frac{\nabla_\varphi q_\varphi(z|x)}{q_\varphi(z|x)} \tag{12.38}$$

Unfortunately, this gradient estimator exhibits very high variance and it is practically useless.
This leads us to the next trick...

(2) Reparametrization trick

If we can find a function $h : (\epsilon, \phi) \to z$ which is differentiable with respect to φ, and a probability distribution $p(\epsilon)$ defined over ϵ that is simple to sample from, such that:

$$q_\varphi(z|x) = h_\varphi(x, \epsilon), \quad \epsilon \sim p(\epsilon) \text{ produces samples } z \sim q_\varphi(z|x) \tag{12.39}$$

Then we can estimate the required gradients as:

$$\nabla_\varphi \mathbb{E}_{z\sim q(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{\epsilon\sim p(\epsilon)}[\nabla_\varphi \log p_\theta(x|h_\varphi(x,\epsilon))] \tag{12.40}$$

now the gradient is not related to the distribution with which we take the expectation!

Derivation:
We want to show:

$$\nabla_\varphi \mathbb{E}_{z\sim q(z|x)}[f(z)] = \mathbb{E}_{\epsilon\sim p(\epsilon)}[\nabla_\varphi f(h(\epsilon,\varphi))] \tag{12.41}$$

Transform $p(\epsilon)$ using the change of variables formula:

$$p(z) = |\frac{d\epsilon}{dz}|p(\epsilon) \Rightarrow |p(z)dz| = |p(\epsilon)d\epsilon| \tag{12.42}$$

Then we re-express the troublesome expectation with respect to the reparametrized density:

$$\nabla_\varphi \mathbb{E}_{z\sim q(z|x)}[f(z)] = \nabla_\varphi \int f(z)q_\varphi(z)dz = \nabla_\varphi \int f(z)p(\epsilon)d\epsilon \tag{12.43}$$

$$= \nabla_\varphi \int f(h(\epsilon,\varphi))p(\epsilon)d\epsilon = \nabla_\varphi \mathbb{E}_{\epsilon\sim p(\epsilon)}[f(h(\epsilon,\varphi))] \tag{12.44}$$

$$= \mathbb{E}_{\epsilon\sim p(\epsilon)}[\nabla_\varphi f(h(\epsilon,\varphi))] \tag{12.45}$$

This returns unbiased Monte-Carlo estimates for the gradients!

(3) Density ratio trick

Density ratios are ubiquitous in statistics and machine learning. They continuously appear when we compare two densities:

$$r(x) = \frac{p(x)}{q(x)} \tag{12.46}$$

e.g., Bayes rules, KL divergence, Importance sampling, etc...

More often than not, in practice it is very hard to estimate density ratios as: $p(x)$, $q(x)$ can be intractable, high-dimensional distributions, which we may only have samples from then but don't know their closed form expression.
Assume we can create a data-set consisting of $2N$ points:

- N data points are drawn from $p(x)$ and are assigned a label $+1$.
- N data points are drawn from $q(x)$ and are assigned a label -1.

By this construction, we can write these probabilities in conditional form:

$$p(x) = p(x|y=+1) \quad , q(x) = p(x|y=-1), \quad \text{Bayes:} p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{12.47}$$

Then:

$$r(x) = \frac{p(x)}{q(x)} = \frac{p(x|y=+1)}{p(x|y=-1)} = \frac{\frac{p(y=+1|x)p(x)}{p(y=+1)}}{\frac{p(y=-1|x)p(x)}{p(y=-1)}} \tag{12.48}$$

$$= \frac{p(y=+1|x)}{p(y=-1|x)} = \frac{p(y=+1|x)}{1-p(y=-1|x)} = \frac{S(x)}{1-S(x)}, \tag{12.49}$$

where $S(x)$ is the output of a discriminator.

Since we have an equal amount of samples for both $p(x), q(x) \Rightarrow p(y=+1) = p(y=-1) = 0.5$. Hence, the problem of density ratio estimation is equivalent to binary classification! And that's something that we are really good at solving:)

Chapter 13

Variational auto-encoders and conditional variational auto-encoders

13.1 Variational Auto-encoder

Setup: We are given data $\{x_i\}, i = 1, ..., n, x_i \in \mathbb{R}^d$ [typically in high-dimensions].
Goal: Learn the distribution that generated the observed data $p(x)$.

To this end, we postulate that there exist a set of latent variables $z \in \mathbb{R}^m$ that explain the variability in x (e.g., if x is an image, z could include the lighting conditions, etc.)

Specifically, we postulate a model $p_\theta(x|z)$ such that we can construct an approximation to $p(x)$ as:

$$p_\theta(x, z) = \int p_\theta(x|z)p(z)dz, \tag{13.1}$$

where $p(z)$ is a prior over the latent variables.
This defines a so called generative model:

$$p_\theta(x, z) = p_\theta(x|z)p(z) \tag{13.2}$$

So, it is natural to ask what those latent variables should be?
In other words, what is the posterior:

$$p(z|x) = \frac{p_\theta(x|z)p(z)}{p_\theta(x)}, \tag{13.3}$$

but this is intractable...
Hence, we will try to approximate this by introducing a set of computable quantities:

$$p(z|x) \approx q_\varphi(z|x) = \mathcal{N}(\mu_\varphi(x), \Sigma_\varphi(x)) \tag{13.4}$$
$$p_\theta(x|z) = \mathcal{N}(\mu_\theta(x), \Sigma_\theta(x)) \tag{13.5}$$
$$p(z) = \mathcal{N}(0, I) \tag{13.6}$$

classical VAE setup of Kingma and Welling.

We would like $q_\varphi(z|x)$ to be a close approximation to the true posterior (i.e.):

$$\mathbb{KL}[q_\varphi(z|x)||p(z|x)] := -\int \log \frac{p(z|x)}{q_\varphi(z|x)} q_\varphi(z|x) dz \tag{13.7}$$

$$= -\int [\log p_\theta(x|z) + \log p(z) - \log p_\theta(x) - \log q_\varphi(z|x)] q_\varphi(z|x) dz \tag{13.8}$$

$$= -\int \log p_\theta(x|z) q_\varphi(z|x) dz + \int \log p_\theta(x) q_\varphi(z|x) dz + \int \log \frac{q_\varphi(z|x)}{p(z)} q_\varphi(z|x) dz \tag{13.9}$$

$$\Rightarrow -\log p_\theta(x) + \mathbb{KL}[q_\varphi(z|x)||p(z|x)] = \mathbb{KL}[q_\varphi(z|x)||p(z)] - \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \tag{13.10}$$

$$\Rightarrow -\log p_\theta(x) \leq \mathbb{KL}[q_\varphi(z|x)||p(z)] - \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \tag{13.11}$$

$$-\log p_\theta(x) \leq \mathcal{L}(\theta, \varphi) \tag{13.12}$$

where we use the reality that $\mathbb{KL}[q_\varphi(z|x)||p(z|x)] \geq 0$ and note the ELBO as $\mathcal{L}(\theta,\varphi) = \mathbb{KL}[q_\varphi(z|x)||p(z)] - \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)]$.

Recall that we need $\nabla_\theta \mathcal{L}$, $\nabla_\varphi \mathcal{L}$, but $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)]$ is troublesome...
Also recall the reparametrizaiton trick:

$$\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{\epsilon\sim p(\epsilon))}[\nabla_\varphi \log p_\theta(x|h_\varphi(x,\epsilon))] \tag{13.13}$$

In this case,

$$z \sim q_\varphi(z|x) = \mathcal{N}(\mu_\varphi(x), \Sigma_\varphi(x)), \tag{13.14}$$

so we can use the following simple re-parametrization:

$$\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{\epsilon\sim p(\epsilon))}[\nabla_\varphi \log p_\theta(x|\mu_\varphi(x) + \epsilon\Sigma_\varphi^{1/2}(x))], \quad \epsilon \sim \mathcal{N}(0, I). \tag{13.15}$$

i.e., if $\epsilon \sim p(\epsilon) = \mathcal{N}(0, I)$, then:

$$z = \mu_\varphi(x) + \epsilon\Sigma_\varphi^{1/2}(x) \sim q_\varphi(z|x) = \mathcal{N}(\mu_\varphi(x), \Sigma_\varphi(x)). \tag{13.16}$$

ELBO calculation

$$x_i \xrightarrow{q_\varphi(z|x)} z_i = \mu_\varphi(x) + \epsilon\Sigma_\varphi^{1/2}(x) \tag{13.17}$$

$$z_i \xrightarrow{p_\theta(z|x)} x_i \tag{13.18}$$

$$\mathbb{KL}[q_\varphi(z|x)||p(z) = \frac{1}{2}[\mu_\varphi(x)^T\mu_\varphi(x) - m - \log|\Sigma_\varphi(x)| + tr\{\Sigma_\varphi(x)\}] \tag{13.19}$$

$$\mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \approx \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(x_i|z_i), \text{ where } z_i = \mu_\varphi(x) + \epsilon\Sigma_\varphi^{1/2}(x), \ \epsilon \sim \mathcal{N}(0, I). \tag{13.20}$$

$$= \frac{1}{2}\log|\Sigma_\varphi(x)| + \frac{1}{2}(x - \mu_\theta(x))^T\Sigma_\theta^{-1}(x)(x - \mu_\theta(x)) + \frac{nd}{2}\log 2\pi \tag{13.21}$$

13.2 Conditional Variational Auto-encoders

Given data $\{x_i.y_i\}, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^s$, we want to learn a conditional probability model.

We will again postulate the existence of latent variables z, such that:

$$p_\theta(y|x) = \int p_\theta(y|x,z) p_\gamma(z|x) dz, \quad z \in \mathbb{R}^m, \quad , m \ll s. \tag{13.22}$$

where we should note that $p(x,y,z) = p(y|x,z)p(z|x)$.
It is again meaningful to ask what those latent variables should be, i.e., what is the posterior?

$$p(z|x,y) = \frac{p_\theta(y|x,z)p_\gamma(z|x)}{p_\theta(y|x)}. \tag{13.23}$$

For general choices of $p_\theta(y|x,z)$ and $p_\gamma(z|x)$ this posterior is intractable, and we will introduce a variational approximation:

$$p(z|x,y) \approx q_\varphi(z|x,y). \tag{13.24}$$

We would like this approximation to minimize the KL-divergence:

$$\mathbb{KL}[q_\varphi(z|x,y)||p(z|x,y)] = -\int \log \frac{p(z|x,y)}{q_\varphi(z|x,y)} q_\varphi(z|x,y)dz \tag{13.25}$$

$$-\int [\log p_\theta(y|x,z) + \log p_\gamma(z|x) - \log p_\theta(y|x) - \log q_\varphi(z|x,y)] q_\varphi(z|x,y)dz \tag{13.26}$$

$$-\int [\log p_\theta(y|x,z) q_\varphi(z|x,y)dz + \int p_\theta(y|x) q_\varphi(z|x,y)dz - \int \log \frac{p_\gamma(z|x)}{q_\varphi(z|x,y)} q_\varphi(z|x,y)dz \tag{13.27}$$

$$\Rightarrow \log p_\theta(y|x) + \mathbb{KL}[q_\varphi(z|x,y)||p(z|x,y)] = \mathbb{KL}[q_\varphi(z|x,y)||p_\gamma(z|x)] - \mathbb{E}_{z\sim q_\varphi(z|x,y)}[\log p_\theta(y|x,z)]. \tag{13.28}$$

The ELBO can be directly computed if we introduce the following representations:

$$p_\theta(y|x,z) = \mathcal{N}(\mu_\theta(x,z), \Sigma_\theta(x,z)) \tag{13.29}$$

$$q_\varphi(z|x,y) = \mathcal{N}(\mu_\varphi(x,y), \Sigma_\varphi(x,y)) \tag{13.30}$$

$$p_\gamma(z|x) = \mathcal{N}(\mu_\gamma(x), \Sigma_\gamma(x)) \tag{13.31}$$

with the work flow:

$$(x,y) \xrightarrow{q_\varphi(z|x)} \mu_\varphi(x,y), \Sigma_\varphi(x,y) \tag{13.32}$$

$$x \xrightarrow{p_\gamma(z|x)} \mu_\gamma(x), \Sigma_\gamma(x) \tag{13.33}$$

$$z = \mu_\varphi(x,y) + \epsilon \Sigma_\varphi^{1/2}(x,y), \quad \epsilon \sim \mathcal{N}(0,I). \tag{13.34}$$

$$(x,z) \xrightarrow{p_\theta(y|x,z)} \mu_\theta(x,z), \Sigma_\theta(x,z) \tag{13.35}$$

Chapter 14

Generative adversarial networks

14.1 Setup

Given data $\{x_i\}, i = 1, ..., n, x_i \in \mathbb{R}^d$, our goal is to learn the true data generating distribution.

- $p(x)$: True data distribution.
- $p_\theta(x)$: Model distribution.
- $q(x)$: Empirical data distribution.

So far we constructed probabilistic models and trained them using maximum likelihood estimate. For e.g., in VAEs we considered a generative model for $p(x)$:

$$p_\theta(x) = \int p(x, z)dz = \int p(x|z)p(z)dz, \quad z \sim p(z), \quad z \in \mathbb{R}^q, \tag{14.1}$$

where $p_\theta(x)$ is the marginal model distribution. Our goal was to make this as close as possible to the true data distribution. To this end we introduced appropriate approximations and constructed a computable lower bound for the marginal likelihood:

$$\mathcal{L}(\theta, \varphi) := -\log p_\theta(x) \leq \mathbb{KL}[q_\varphi(z|x)||p(z)] - \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \tag{14.2}$$

(for the $q_\varphi(z|x)$ not to be confused with the empirical distribution). Given data $\{x_i\}$ drawn from the empirical data distribution $x \sim q(x)$, we evaluated and optimized this objective for $\{\theta^*, \varphi^*\}$.
This MLE approach can be shown to be equivalent (from a theoretical point of view that minimizing the KL divergence between the empirical data distribution and the model marginal distribution):

$$\{\theta^*, \varphi^*\} = \arg\min_{\theta,\varphi} \mathbb{KL}[q(x)||p_\theta(x)] \tag{14.3}$$

14.2 GANs

Here we introduce a generative model:
$G_\theta(z), z \sim p(z)$, and $G_\theta(\cdot)$ is a deterministic map parametrized by θ, such that if $x = G_\theta(z)$, then $x \sim p_\theta(x) \approx p(x), G : \mathbb{R}^q \to \mathbb{R}^d$.

Unlike MLE approaches, here we introduce a discriminator $D_\varphi(x), D : \mathbb{R}^d \to [0, 1]$ that represents the probability that x came from the empirical data distribution $q(x)$, rather than the model distribution $p_\theta(x)$ (x' produced by the generator).

Then, we train the discriminator $D_\varphi(x)$ to assign the correct labels to both samples from $q(x)$ and samples from $p_\theta(x)$. We simultaneously train the generator $G_\theta(z)$ to minimize $\log(1 - D_\varphi(G_\theta(z)))$, i.e., to fool the discriminator that the generated sample came from the data distribution.

14.3 Theoretical results (in the non-parametric limit)

(1)

For $G_\theta(z)$ fixed, the optimal discriminator is:

$$D^*_\varphi(x) = \frac{q(x)}{q(x) + p_\theta(x)}. \tag{14.4}$$

And the training objective for $D_\varphi(x)$ can be reformulated as:

$$c(\theta) = \max_\varphi \mathcal{L}(\theta, \varphi) = \mathbb{E}_{x\sim q(x)}[\log \frac{q(x)}{q(x) + p_\theta(x)}] + \mathbb{E}_{x\sim p_\theta(x)}[\log \frac{p_\theta(x)}{q(x) + p_\theta(x)}]. \tag{14.5}$$

The global minimum for this objective is achieved if and only if $p_\theta(x) = q(x)$.
At that point $c(\theta) = -\log 4$.
This objective can be also re-written as:

$$c(\theta) = -\log 4 + \mathbb{KL}[q(x)||\frac{q(x) + p_\theta(x)}{2}] + \mathbb{KL}[p_\theta(x)||\frac{q(x) + p_\theta(x)}{2}] \tag{14.6}$$

$$\Rightarrow c(\theta) = -\log 4 + 2JSD(q(x)||p_\theta(x)) \tag{14.7}$$

where we note that $JSD(q(x)||p_\theta(x)) = \mathbb{KL}[q(x)||\frac{q(x)+p_\theta(x)}{2}] + \mathbb{KL}[p_\theta(x)||\frac{q(x)+p_\theta(x)}{2}]$ is the Jensen-Shannon divergence between $p_\theta(x)$ and $q(x)$. Unlike KL divergence, this is a symmetric divergence.

(2)

If $G_\theta(z)$ and $D_\varphi(x)$ have enough capacity, and at each step of the algorithm the discriminator is allowed to reach its optimum given $G_\theta(z)$, and $p_\theta(x)$ is updated so as to improve the criterion:

$$\mathbb{E}_{x\sim q(x)}[\log D^*_\varphi(x)] + \mathbb{E}_{x\sim p_\theta(x)}[\log(1 - D^*_\varphi(x))] \tag{14.8}$$

Then, $p_\theta(x)$ converges to $q(x)$.
This procedure defines a zero-sum min-max game between two players, resulting in an objective function:

$$\min_\theta \max_\varphi \mathbb{E}_{x\sim q(x)}[\log D_\varphi(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D_\varphi(G_\theta(z)))] := \mathcal{L}(\theta, \varphi) \tag{14.9}$$

In practice, this may not provide sufficient gradients for $G_\theta(\cdot)$ to train. Early on, the samples provided by $G_\theta(z)$ are poor and $D_\varphi(x)$ rejects them with high confidence because they are clearly different from the training data, and $\log(1 - D_\varphi(G_\theta(z)))$ saturates.

Instead we can train $G_\theta(z)$ to maximize $\log D_\varphi(G_\theta(z))$, i.e.:

$$\max \theta \min \varphi \mathbb{E}_{x\sim q(x)}[\log D_\varphi(x)] + \mathbb{E}_{z\sim p(z)}[\log D_\varphi(G_\theta(z))] \tag{14.10}$$

14.4 GAN algorithm

Normalized zero mean and unit variance data pairs $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1, ..., \hat{\boldsymbol{x}}_N]$, $\hat{\boldsymbol{Y}} = [\hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_N]$, relative number of updates for discriminator and generator in each iteration N_d and N_g.
$i = 0$

Algorithm 1: GAN algorithm.

while $iter < T$ **do**
  while $k < t$ **do**
    Sample mini-batch of m noise samples $\{z_1, ..., z_m\}$ from prior distribution $p(z)$;
    Sample mini-batch of m data points $\{x_1, ..., x_m\}$, ;

$$\varphi_{n+1} = \varphi_n + \eta \nabla_\varphi \{ \frac{1}{m} \sum_{i=1}^{m} [\log[D_\varphi(x_i)] + \log[1 - D_\varphi(G_\theta(z_i))]] \}$$

    where θ is kept fixed.;
    $k = k + 1$;
  end
  Sample mini-batch of m noise samples $\{z_1, ..., z_m\}$ from $p(z)$;

$$\theta_{n+1} = \theta_n + \eta \nabla_\theta \{ \frac{1}{m} \sum_{i=1}^{m} [\log[1 - D_\varphi(G_\theta(z_i))]] \}$$

  iter = iter + 1;
end

14.5 Adversarially Learned Inference

Recall that in GANs we introduced a generative model:

$$x = G_\theta(z), \ z \sim p(z), \ \text{with the goal of } x \sim p_\theta(x) \approx q(x). \tag{14.11}$$

where $G_\theta : \mathbb{R}^q \to \mathbb{R}^d$, deterministic generator / decoder.
To effectively train this model we introduced a discriminator $D_\varphi : \mathbb{R}^d \to [0, 1]$ that learns to distinguish between samples from $p_\theta(x)$ and samples from $q(x)$, leading to an adversarial optimization objective:

$$\min_\theta \max_\varphi \mathbb{E}_{x \sim q(x)}[\log D_\varphi(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D_\varphi(G_\theta(z)))] := \mathcal{L}(\theta, \varphi) \tag{14.12}$$

To summarize the GAN setup aimed to match:

- The empirical data distribution $q(x)$.
- The model marginal distribution $p_\theta(x) = \int p_\theta(x|z)p(z)dz$

by minimizing (in theory) the $JSD(q(x)||p_\theta(x))$.
However, this model is not capable of performing inference over the latent variables (i.e., we don't know what those variables should be given the data, i.e., we don't know the posterior $p(z|x)$).
To this end, we can introduce a new model that is capable of performing inference over z. To this end, we will consider the joint distributions:

- The encoder joint: $q_\varphi(x, z) = q_\varphi(z|x)q(x)$
- The decoder joint: $p_\theta(x, z) = p_\theta(x|z)p(z)$

To learn this model in an adversarial fashion we need again to introduce a discriminator $T_\psi(x, z)$ that learns how to distinguish between samples drawn from $q_\varphi(x, z) = q_\varphi(z|x)q(x)$, and $p_\theta(x, z) = p_\theta(x|z)p(z)$.
Like before, the discriminator is trained to accurately learn how to distinguish samples, whereas the generator is trained to generate joint samples that can fool the discriminator. But this time, we can also learn the encoder $q_\varphi(z|x)$.

In ananaly to GANs, the adversarial objective now becomes:

$$\min_{\theta,\varphi} \max_{\psi} \mathbb{E}_{x\sim q(x)}[\log T_\psi(x,z)] + \mathbb{E}_{z\sim p(z)}[\log(1-T_\psi(x,z))]. \tag{14.13}$$

Again, for fixed $p_\theta(x|z)$ and $q_\varphi(z|x)$, the optimal discriminator is:

$$T^*_\psi(x,z) = \frac{q_\varphi(x,z)}{q_\varphi(x,z)+p_\theta(x,z)}. \tag{14.14}$$

and given this optimal discriminator, the adversarial game above minimized the Jensen-Shannon divergence between $p_\theta(x,z)$ and $q_\varphi(x,z)$.
Matching the joints has also the effect of matching the marginals, i.e., $p_\theta(x) \approx q(x)$ as well as the conditionals $p(z|x) \approx q_\varphi(z|x)$ and $p_\theta(x|z) \approx q(x|z)$.

As in GANs, in order to alleviate the vanishing gradients issues, in practice, we optimize:

$$\max_{\theta,\varphi} \min \psi \mathbb{E}_{x\sim q(x), z\sim q_\varphi(z|x)}[\log(1-T_\psi(x,z))] + \mathbb{E}_{z\sim p(z), x\sim p_\theta(x|z)}[\log T_\psi(x,z)]. \tag{14.15}$$

Proper care needs to be taken when evaluating the gradients using the reparametrization trick, i.e., $p_\theta(x|z) = f_\theta(z,\epsilon)$, $q_\varphi(z|x) = f_\varphi(x,\epsilon)$, $\epsilon \sim \mathcal{N}(0,I)$.