Combined Cheat Sheets of Machine Learning, Deep Learning, Probability and LLMs

Syed Afroz Ali
Data Scientist (Kaggle Grandmaster)
https://www.kaggle.com/pythonafroz
https://www.linkedin.com/in/syed-afroz-70939914/

Probability–the Science of Uncertainty and Data

by Fabián Kozynski

Probability

Probability models and axioms

Definition (Sample space) A sample space Ω is the set of all possible outcomes. The set's elements must be mutually exclusive, collectively exhaustive and at the right granularity.

Definition (Event) An event is a subset of the sample space. Probability is assigned to events.

Definition (Probability axioms) A probability law $\mathbb{P}$ assigns probabilities to events and satisfies the following axioms:

Nonnegativity $\mathbb{P}(A) \geq 0$ for all events A.

Normalization $\mathbb{P}(\Omega) = 1$.

(Countable) additivity For every sequence of events $A_1, A_2, \ldots$ such that $A_i \cap A_j = \varnothing$: $\mathbb{P}\left(\bigcup_i A_i\right) = \sum_i \mathbb{P}(A_i)$.

Corollaries (Consequences of the axioms)

- $\mathbb{P}(\varnothing) = 0$.
- For any finite collection of disjoint events $A_1, \ldots, A_n$, $\mathbb{P}\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n \mathbb{P}(A_i)$.
- $\mathbb{P}(A) + \mathbb{P}(A^c) = 1$.
- $\mathbb{P}(A) \leq 1$.
- If $A \subset B$, then $\mathbb{P}(A) \leq \mathbb{P}(B)$.
- $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$.
- $\mathbb{P}(A \cup B) \leq \mathbb{P}(A) + \mathbb{P}(B)$.

Example (Discrete uniform law) Assume Ω is finite and consists of n equally likely elements. Also, assume that $A \subset \Omega$ with k elements. Then $\mathbb{P}(A) = \frac{k}{n}$.

Conditioning and Bayes' rule

Definition (Conditional probability) Given that event B has occurred and that $P(B) > 0$, the probability that A occurs is

$$\mathbb{P}(A|B) \triangleq \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}.$$

Remark (Conditional probabilities properties) They are the same as ordinary probabilities. Assuming $\mathbb{P}(B) > 0$:

- $\mathbb{P}(A|B) \geq 0$.
- $\mathbb{P}(\Omega|B) = 1$
- $\mathbb{P}(B|B) = 1$.
- If $A \cap C = \varnothing$, $\mathbb{P}(A \cup C|B) = \mathbb{P}(A|B) + \mathbb{P}(C|B)$.

Proposition (Multiplication rule)

$$\mathbb{P}(A_1 \cap A_2 \cap \cdots \cap A_n) = \mathbb{P}(A_1) \cdot \mathbb{P}(A_2|A_1) \cdots \mathbb{P}(A_n|A_1 \cap A_2 \cap \cdots \cap A_{n-1}).$$

Theorem (Total probability theorem) Given a partition $\{A_1, A_2, \ldots\}$ of the sample space, meaning that $\bigcup_i A_i = \Omega$ and the events are disjoint, and for every event B, we have

$$\mathbb{P}(B) = \sum_i \mathbb{P}(A_i)\mathbb{P}(B|A_i).$$

Theorem (Bayes' rule) Given a partition $\{A_1, A_2, \ldots\}$ of the sample space, meaning that $\bigcup_i A_i = \Omega$ and the events are disjoint, and if $\mathbb{P}(A_i) > 0$ for all i, then for every event B, the conditional probabilities $\mathbb{P}(A_i|B)$ can be obtained from the conditional probabilities $\mathbb{P}(B|A_i)$ and the initial probabilities $\mathbb{P}(A_i)$ as follows:

$$\mathbb{P}(A_i|B) = \frac{\mathbb{P}(A_i)\mathbb{P}(B|A_i)}{\sum_j \mathbb{P}(A_j)\mathbb{P}(B|A_j)}.$$

Independence

Definition (Independence of events) Two events are independent if occurrence of one provides no information about the other. We say that A and B are independent if

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B).$$

Equivalently, as long as $\mathbb{P}(A) > 0$ and $\mathbb{P}(B) > 0$,

$$\mathbb{P}(B|A) = \mathbb{P}(B) \qquad \mathbb{P}(A|B) = \mathbb{P}(A).$$

Remarks

- The definition of independence is symmetric with respect to A and B.
- The product definition applies even if $\mathbb{P}(A) = 0$ or $\mathbb{P}(B) = 0$.

Corollary If A and B are independent, then A and B^c are independent. Similarly for A^c and B, or for A^c and B^c.

Definition (Conditional independence) We say that A and B are independent conditioned on C, where $\mathbb{P}(C) > 0$, if

$$\mathbb{P}(A \cap B|C) = \mathbb{P}(A|C)\mathbb{P}(B|C).$$

Definition (Independence of a collection of events) We say that events $A_1, A_2, \ldots, A_n$ are independent if for every collection of distinct indices $i_1, i_2, \ldots, i_k$, we have

$$\mathbb{P}(A_{i_1} \cap \ldots \cap A_{i_k}) = \mathbb{P}(A_{i_1}) \cdot \mathbb{P}(A_{i_2}) \cdots \mathbb{P}(A_{i_k}).$$

Counting

This section deals with finite sets with uniform probability law. In this case, to calculate $\mathbb{P}(A)$, we need to count the number of elements in A and in Ω.

Remark (Basic counting principle) For a selection that can be done in r stages, with n_i choices at each stage i, the number of possible selections is $n_1 \cdot n_2 \cdots n_r$.

Definition (Permutations) The number of permutations (orderings) of n different elements is

$$n! = 1 \cdot 2 \cdot 3 \cdots n.$$

Definition (Combinations) Given a set of n elements, the number of subsets with exactly k elements is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Definition (Partitions) We are given an n–element set and nonnegative integers $n_1, n_2, \ldots, n_r$, whose sum is equal to n. The number of partitions of the set into r disjoint subsets, with the i^{th} subset containing exactly n_i elements, is equal to

$$\binom{n}{n_1, \ldots, n_r} = \frac{n!}{n_1! n_2! \cdots n_r!}.$$

Remark This is the same as counting how to assign n distinct elements to r people, giving each person i exactly n_i elements.

Discrete random variables

Probability mass function and expectation

Definition (Random variable) A random variable X is a function of the sample space Ω into the real numbers (or $\mathbb{R}^n$). Its range can be discrete or continuous.

Definition (Probability mass funtion (PMF)) The probability law of a discrete random variable X is called its PMF. It is defined as

$$p_X(x) = \mathbb{P}(X = x) = \mathbb{P}\left(\{\omega \in \Omega : X(\omega) = x\}\right).$$

Properties

$p_X(x) \geq 0$, $\forall\, x$.

$\sum_x p_X(x) = 1$.

Example (Bernoulli random variable) A Bernoulli random variable X with parameter $0 \leq p \leq 1$ ($X \sim \text{Ber}(p)$) takes the following values:

$$X = \begin{cases} 1 & \text{w.p. } p, \\ 0 & \text{w.p. } 1-p. \end{cases}$$

An indicator random variable of an event ($I_A = 1$ if A occurs) is an example of a Bernoulli random variable.

Example (Discrete uniform random variable) A Discrete uniform random variable X between a and b with $a \leq b$ ($X \sim \text{Uni}[a, b]$) takes any of the values in $\{a, a+1, \ldots, b\}$ with probability $\frac{1}{b-a+1}$.

Example (Binomial random variable) A Binomial random variable X with parameters n (natural number) and $0 \leq p \leq 1$ ($X \sim \text{Bin}(n, p)$) takes values in the set $\{0, 1, \ldots, n\}$ with probabilities $p_X(i) = \binom{n}{i} p^i (1-p)^{n-i}$.
It represents the number of successes in n independent trials where each trial has a probability of success p. Therefore, it can also be seen as the sum of n independent Bernoulli random variables, each with parameter p.

Example (Geometric random variable) A Geometric random variable X with parameter $0 \leq p \leq 1$ ($X \sim \text{Geo}(p)$) takes values in the set $\{1, 2, \ldots\}$ with probabilities $p_X(i) = (1-p)^{i-1} p$.
It represents the number of independent trials until (and including) the first success, when the probability of success in each trial is p.

Definition (Expectation/mean of a random variable) The expectation of a discrete random variable is defined as

$$\mathbb{E}[X] \triangleq \sum_x x p_X(x).$$

assuming $\sum_x |x| p_X(x) < \infty$.

Properties (Properties of expectation)

- If $X \geq 0$ then $\mathbb{E}[X] \geq 0$.
- If $a \leq X \leq b$ then $a \leq \mathbb{E}[X] \leq b$.
- If $X = c$ then $\mathbb{E}[X] = c$.

Example Expected value of know r.v.

- If $X \sim \text{Ber}(p)$ then $\mathbb{E}[X] = p$.
- If $X = I_A$ then $\mathbb{E}[X] = \mathbb{P}(A)$.
- If $X \sim \text{Uni}[a, b]$ then $\mathbb{E}[X] = \frac{a+b}{2}$.
- If $X \sim \text{Bin}(n, p)$ then $\mathbb{E}[X] = np$.
- If $X \sim \text{Geo}(p)$ then $\mathbb{E}[X] = \frac{1}{p}$.

Theorem (Expected value rule) Given a random variable X and a function $g:\mathbb{R}\to\mathbb{R}$, we construct the random variable $Y=g(X)$. Then

$$\sum_y yp_Y(y)=\mathbb{E}[Y]=\mathbb{E}[g(X)]=\sum_x g(x)p_X(x).$$

Remark (PMF of $Y=g(X)$) The PMF of $Y=g(X)$ is $p_Y(y)=\sum_{x:g(x)=y}p_X(x)$.

Remark In general $g(\mathbb{E}[X])\neq\mathbb{E}[g(X)]$. They are equal if $g(x)=ax+b$.

Variance, conditioning on an event, multiple r.v.

Definition (Variance of a random variable) Given a random variable X with $\mu=\mathbb{E}[X]$, its variance is a measure of the spread of the random variable and is defined as

$$\mathrm{Var}(X)\triangleq\mathbb{E}\left[(X-\mu)^2\right]=\sum_x(x-\mu)^2p_X(x).$$

Definition (Standard deviation)

$$\sigma_X=\sqrt{\mathrm{Var}(X)}.$$

Properties (Properties of the variance)

- $\mathrm{Var}(aX)=a^2\,\mathrm{Var}(X)$, for all $a\in\mathbb{R}$.
- $\mathrm{Var}(X+b)=\mathrm{Var}(X)$, for all $b\in\mathbb{R}$.
- $\mathrm{Var}(aX+b)=a^2\,\mathrm{Var}(X)$.
- $\mathrm{Var}(X)=\mathbb{E}[X^2]-(\mathbb{E}[X])^2$.

Example (Variance of known r.v.)

- If $X\sim\mathrm{Ber}(p)$, then $\mathrm{Var}(X)=p(1-p)$.
- If $X\sim\mathrm{Uni}[a,b]$, then $\mathrm{Var}(X)=\frac{(b-a)(b-a+2)}{12}$.
- If $X\sim\mathrm{Bin}(n,p)$, then $\mathrm{Var}(X)=np(1-p)$.
- If $X\sim\mathrm{Geo}(p)$, then $\mathrm{Var}(X)=\frac{1-p}{p^2}$

Proposition (Conditional PMF and expectation, given an event) Given the event A, with $\mathbb{P}(A)>0$, we have the following

- $p_{X|A}(x)=\mathbb{P}(X=x|A)$.
- If A is a subset of the range of X, then:
  $p_{X|A}(x)\triangleq p_{X|\{X\in A\}}(x)=\begin{cases}\frac{1}{\mathbb{P}(A)}p_X(x), & \text{if } x\in A,\\ 0, & \text{otherwise.}\end{cases}$
- $\sum_x p_{X|A}(x)=1$.
- $\mathbb{E}[X|A]=\sum_x xp_{X|A}(x)$.
- $\mathbb{E}[g(X)|A]=\sum_x g(x)p_{X|A}(x)$.

Proposition (Total expectation rule) Given a partition of disjoint events $A_1,\dots,A_n$ such that $\sum_i\mathbb{P}(A_i)=1$, and $\mathbb{P}(A_i)>0$,

$$\mathbb{E}[X]=\mathbb{P}(A_1)\mathbb{E}[X|A_1]+\cdots+\mathbb{P}(A_n)\mathbb{E}[X|A_n].$$

Definition (Memorylessness of the geometric random variable) When we condition a geometric random variable X on the event $X>n$ we have memorylessness, meaning that the "remaining time" $X-n$, given that $X>n$, is also geometric with the same parameter. Formally,

$$p_{X-n|X>n}(i)=p_X(i).$$

Definition (Joint PMF) The joint PMF of random variables $X_1,X_2,\dots,X_n$ is $p_{X_1,X_2,\dots,X_n}(x_1,\dots,x_n)=\mathbb{P}(X_1=x_1,\dots,X_n=x_n)$.

Properties (Properties of joint PMF)

- $\sum_{x_1}\cdots\sum_{x_n}p_{X_1,\dots,X_n}(x_1,\dots,x_n)=1$.
- $p_{X_1}(x_1)=\sum_{x_2}\cdots\sum_{x_n}p_{X_1,\dots,X_n}(x_1,x_2,\dots,x_n)$.
- $p_{X_2,\dots,X_n}(x_2,\dots,x_n)=\sum_{x_1}p_{X_1,X_2,\dots,X_n}(x_1,x_2,\dots,x_n)$.

Definition (Functions of multiple r.v.) If $Z=g(X_1,\dots,X_n)$, where $g:\mathbb{R}^n\to\mathbb{R}$, then $p_Z(z)=\mathbb{P}(g(X_1,\dots,X_n)=z)$.

Proposition (Expected value rule for multiple r.v.) Given $g:\mathbb{R}^n\to\mathbb{R}$,

$$\mathbb{E}[g(X_1,\dots,X_n)]=\sum_{x_1,\dots,x_n}g(x_1,\dots,x_n)p_{X_1,\dots,X_n}(x_1,\dots,x_n).$$

Properties (Linearity of expectations)

- $\mathbb{E}[aX+b]=a\mathbb{E}[X]+b$.
- $\mathbb{E}[X_1+\cdots+X_n]=\mathbb{E}[X_1]+\cdots+\mathbb{E}[X_n]$.

Conditioning on a random variable, independence

Definition (Conditional PMF given another random variable) Given discrete random variables X,Y and y such that $p_Y(y)>0$ we define

$$p_{X|Y}(x|y)\triangleq\frac{p_{X,Y}(x,y)}{p_Y(y)}.$$

Proposition (Multiplication rule) Given jointly discrete random variables X,Y, and whenever the conditional probabilities are defined,

$$p_{X,Y}(x,y)=p_X(x)p_{Y|X}(y|x)=p_Y(y)p_{X|Y}(x|y).$$

Definition (Conditional expectation) Given discrete random variables X,Y and y such that $p_Y(y)>0$ we define

$$\mathbb{E}[X|Y=y]=\sum_x xp_{X|Y}(x|y).$$

Additionally we have

$$\mathbb{E}[g(X)|Y=y]=\sum_x g(x)p_{X|Y}(x|y).$$

Theorem (Total probability and expectation theorems) If $p_Y(y)>0$, then

$$p_X(x)=\sum_y p_Y(y)p_{X|Y}(x|y),$$
$$\mathbb{E}[X]=\sum_y p_Y(y)\mathbb{E}[X|Y=y].$$

Definition (Independence of a random variable and an event) A discrete random variable X and an event A are independent if $\mathbb{P}(X=x\text{ and }A)=p_X(x)\mathbb{P}(A)$, for all x.

Definition (Independence of two random variables) Two discrete random variables X and Y are independent if $p_{X,Y}(x,y)=p_X(x)p_Y(y)$ for all x,y.

Remark (Independence of a collection of random variables) A collection $X_1,X_2,\dots,X_n$ of random variables are independent if

$$p_{X_1,\dots,X_n}(x_1,\dots,x_n)=p_{X_1}(x_1)\cdots p_{X_n}(x_n),\ \forall\,x_1,\dots,x_n.$$

Remark (Independence and expectation) In general, $\mathbb{E}[g(X,Y)]\neq g(\mathbb{E}[X],\mathbb{E}[Y])$. An exception is for linear functions: $\mathbb{E}[aX+bY]=a\mathbb{E}[X]+b\mathbb{E}[Y]$.

Proposition (Expectation of product of independent r.v.) If X and Y are discrete independent random variables,

$$\mathbb{E}[XY]=\mathbb{E}[X]\mathbb{E}[Y].$$

Remark If X and Y are independent, $\mathbb{E}[g(X)h(Y)]=\mathbb{E}[g(X)]\,\mathbb{E}[h(Y)]$.

Proposition (Variance of sum of independent random variables) IF X and Y are discrete independent random variables,

$$\mathrm{Var}(X+Y)=\mathrm{Var}(X)+\mathrm{Var}(Y).$$

Continuous random variables

PDF, Expectation, Variance, CDF

Definition (Probability density function (PDF)) A probability density function of a r.v. X is a non-negative real valued function f_X that satisfies the following

- $\int_{-\infty}^{\infty}f_X(x)\mathrm{d}x=1$.
- $\mathbb{P}(a\le X\le b)=\int_a^b f_X(x)\mathrm{d}x$ for some random variable X.

Definition (Continuous random variable) A random variable X is continuous if its probability law can be described by a PDF f_X.

Remark Continuous random variables satisfy:

- For small $\delta>0$, $\mathbb{P}(a\le X\le a+\delta)\approx f_X(a)\delta$.
- $\mathbb{P}(X=a)=0$, $\forall a\in\mathbb{R}$.

Definition (Expectation of a continuous random variable) The expectation of a continuous random variable is

$$\mathbb{E}[X]\triangleq\int_{-\infty}^{\infty}xf_X(x)\mathrm{d}x.$$

assuming $\int_{-\infty}^{\infty}|x|f_X(x)\mathrm{d}x<\infty$.

Properties (Properties of expectation)

- If $X\ge0$ then $\mathbb{E}[X]\ge0$.
- If $a\le X\le b$ then $a\le\mathbb{E}[X]\le b$.
- $\mathbb{E}[g(X)]=\int_{-\infty}^{\infty}g(x)f_X(x)\mathrm{d}x$.
- $\mathbb{E}[aX+b]=a\mathbb{E}[X]+b$.

Definition (Variance of a continuous random variable) Given a continuous random variable X with $\mu=\mathbb{E}[X]$, its variance is

$$\mathrm{Var}(X)=\mathbb{E}\left[(X-\mu)^2\right]=\int_{-\infty}^{\infty}(x-\mu)^2f_X(x)\mathrm{d}x.$$

It has the same properties as the variance of a discrete random variable.

Example (Uniform continuous random variable) A Uniform continuous random variable X between a and b, with $a<b$, $(X\sim\mathrm{Uni}(a,b))$ has PDF

$$f_X(x)=\begin{cases}\frac{1}{b-a}, & \text{if } a<x<b,\\ 0, & \text{otherwise.}\end{cases}$$

We have $\mathbb{E}[X]=\frac{a+b}{2}$ and $\mathrm{Var}(X)=\frac{(b-a)^2}{12}$.

Example (Exponential random variable) An Exponential random variable X with parameter $\lambda > 0$ ($X \sim Exp(\lambda)$) has PDF

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$

We have $E[X] = \frac{1}{\lambda}$ and $\text{Var}(X) = \frac{1}{\lambda^2}$.

Definition (Cumulative Distribution Function (CDF)) The CDF of a random variable X is $F_X(x) = \mathbb{P}(X \leq x)$.
In particular, for a continuous random variable, we have

$$F_X(x) = \int_{-\infty}^{x} f_X(x)\mathrm{d}x,$$

$$f_X(x) = \frac{\mathrm{d}F_X(x)}{\mathrm{d}x}.$$

Properties (Properties of CDF)

- If $y \geq x$, then $F_X(y) \geq F_X(x)$.
- $\lim_{x \to -\infty} F_X(x) = 0$.
- $\lim_{x \to \infty} F_X(x) = 1$.

Definition (Normal/Gaussian random variable) A Normal random variable X with mean μ and variance $\sigma^2 > 0$ ($X \sim \mathcal{N}(\mu, \sigma^2)$) has PDF

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}.$$

We have $E[X] = \mu$ and $\text{Var}(X) = \sigma^2$.

Remark (Standard Normal) The standard Normal is $\mathcal{N}(0, 1)$.

Proposition (Linearity of Gaussians) Given $X \sim \mathcal{N}(\mu, \sigma^2)$, and if $a \neq 0$, then $aX + b \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$.
Using this $Y = \frac{X-\mu}{\sigma}$ is a standard gaussian.

Conditioning on an event, and multiple continuous r.v.

Definition (Conditional PDF given an event) Given a continuous random variable X and event A with $P(A) > 0$, we define the conditional PDF as the function that satisfies

$$\mathbb{P}(X \in B|A) = \int_B f_{X|A}(x)\mathrm{d}x.$$

Definition (Conditional PDF given $X \in A$) Given a continuous random variable X and an $A \subset \mathbb{R}$, with $P(A) > 0$:

$$f_{X|X\in A}(x) = \begin{cases} \frac{1}{\mathbb{P}(A)} f_X(x), & x \in A, \\ 0, & x \notin A. \end{cases}$$

Definition (Conditional expectation) Given a continuous random variable X and an event A, with $P(A) > 0$:

$$\mathbb{E}[X|A] = \int_{-\infty}^{\infty} f_{X|A}(x)\mathrm{d}x.$$

Definition (Memorylessness of the exponential random variable) When we condition an exponential random variable X on the event $X > t$ we have memorylessness, meaning that the "remaining time" $X - t$ given that $X > t$ is also geometric with the same parameter i.e.,

$$\mathbb{P}(X - t > x|X > t) = \mathbb{P}(X > x).$$

Theorem (Total probability and expectation theorems) Given a partition of the space into disjoint events $A_1, A_2, \ldots, A_n$ such that $\sum_i \mathbb{P}(A_i) = 1$ we have the following:

$$F_X(x) = \mathbb{P}(A_1)F_{X|A_1}(x) + \cdots + \mathbb{P}(A_n)F_{X|A_n}(x),$$
$$f_X(x) = \mathbb{P}(A_1)f_{X|A_1}(x) + \cdots + \mathbb{P}(A_n)f_{X|A_n}(x),$$
$$\mathbb{E}[X] = \mathbb{P}(A_1)\mathbb{E}[X|A_1] + \cdots + \mathbb{P}(A_n)\mathbb{E}[X|A_n].$$

Definition (Jointly continuous random variables) A pair (collection) of random variables is jointly continuous if there exists a joint PDF $f_{X,Y}$ that describes them, that is, for every set $B \subset \mathbb{R}^n$

$$\mathbb{P}((X, Y) \in B) = \iint_B f_{X,Y}(x, y)\mathrm{d}x\mathrm{d}y.$$

Properties (Properties of joint PDFs)

- $f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x, y)\mathrm{d}y$.
- $F_{X,Y}(x, y) = \mathbb{P}(X \leq x, Y \leq y) = \int_{-\infty}^{x} \left[\int_{-\infty}^{y} f_{X,Y}(u, v)\mathrm{d}v \right] \mathrm{d}u$.
- $f_{X,Y}(x) = \frac{\partial^2 F_{X,Y}(x,y)}{\partial x\, \partial y}$.

Example (Uniform joint PDF on a set S) Let $S \subset \mathbb{R}^2$ with area $s > 0$, then the random variable (X, Y) is uniform over S if it has PDF

$$f_{X,Y}(x, y) = \begin{cases} \frac{1}{s}, & (x, y) \in S, \\ 0, & (x, y) \notin S. \end{cases}$$

Conditioning on a random variable, independence, Bayes' rule

Definition (Conditional PDF given another random variable) Given jointly continuous random variables X, Y and a value y such that $f_Y(y) > 0$, we define the conditional PDF as

$$f_{X|Y}(x|y) \triangleq \frac{f_{X,Y}(x, y)}{f_Y(y)}.$$

Additionally we define $\mathbb{P}(X \in A|Y = y) \int_A f_{X|Y}(x|y)\mathrm{d}x$.

Proposition (Multiplication rule) Given jointly continuous random variables X, Y, whenever possible we have

$$f_{X,Y}(x, y) = f_X(x)f_{Y|X}(y|x) = f_Y(y)f_{X|Y}(x|y).$$

Definition (Conditional expectation) Given jointly continuous random variables X, Y, and y such that $f_Y(y) > 0$, we define the conditional expected value as

$$\mathbb{E}[X|Y = y] = \int_{-\infty}^{\infty} x f_{X|Y}(x|y)\mathrm{d}x.$$

Additionally we have

$$\mathbb{E}[g(X)|Y = y] = \int_{-\infty}^{\infty} g(x) f_{X|Y}(x|y)\mathrm{d}x.$$

Theorem (Total probability and total expectation theorems)

$$f_X(x) = \int_{-\infty}^{\infty} f_Y(y)f_{X|Y}(x|y)\mathrm{d}y,$$
$$\mathbb{E}[X] = \int_{-\infty}^{\infty} f_Y(y)\mathbb{E}[X|Y = y]\mathrm{d}y.$$

Definition (Independence) Jointly continuous random variables X, Y are independent if $f_{X,Y}(x, y) = f_X(x)f_Y(y)$ for all x, y.

Proposition (Expectation of product of independent r.v.) If X and Y are independent continuous random variables,

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y].$$

Remark If X and Y are independent, $\mathbb{E}[g(X)h(Y)] = \mathbb{E}[g(X)]\,\mathbb{E}[h(Y)]$.

Proposition (Variance of sum of independent random variables) If X and Y are independent continuous random variables,

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y).$$

Proposition (Bayes' rule summary)

- For X, Y discrete: $p_{X|Y}(x|y) = \frac{p_X(x)p_{Y|X}(y|x)}{p_Y(y)}$.
- For X, Y continuous: $f_{X|Y}(x|y) = \frac{f_X(x)f_{Y|X}(y|x)}{f_Y(y)}$.
- For X discrete, Y continuous: $p_{X|Y}(x|y) = \frac{p_X(x)f_{Y|X}(y|x)}{f_Y(y)}$.
- For X continuous, Y discrete: $f_{X|Y}(x|y) = \frac{f_X(x)p_{Y|X}(y|x)}{p_Y(y)}$.

Derived distributions

Proposition (Discrete case) Given a discrete random variable X and a function g, the r.v. $Y = g(X)$ has PMF

$$p_Y(y) = \sum_{x:g(x)=y} p_X(x).$$

Remark (Linear function of discrete random variable) If $g(x) = ax + b$, then $p_Y(y) = p_X\left(\frac{y-b}{a}\right)$.

Proposition (Linear function of continuous r.v.) Given a continuous random variable X and $Y = aX + b$, with $a \neq 0$, we have

$$f_Y(y) = \frac{1}{|a|} f_X\left(\frac{y - b}{a}\right).$$

Corollary (Linear function of normal r.v.) If $X \sim \mathcal{N}(\mu, \sigma^2)$ and $Y = aX + b$, with $a \neq 0$, then $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$.

Example (General function of a continuous r.v.) If X is a continuous random variable and g is any function, to obtain the pdf of $Y = g(X)$ we follow the two-step procedure:

1. Find the CDF of Y: $F_Y(y) = \mathbb{P}(Y \leq y) = \mathbb{P}(g(X) \leq y)$.
2. Differentiate the CDF of Y to obtain the PDF: $f_Y(y) = \frac{\mathrm{d}F_Y(y)}{\mathrm{d}y}$.

Proposition (General formula for monotonic g) Let X be a continuous random variable and g a function that is monotonic wherever $f_X(x) > 0$. The PDF of $Y = g(X)$ is given by

$$f_Y(y) = f_X(h(y)) \left| \frac{\mathrm{d}h}{\mathrm{d}y}(y) \right|.$$

where $h = g^{-1}$ in the interval where g is monotonic.

Sums of independent r.v., covariance and correlation

Proposition (Discrete case) Let X, Y be discrete independent random variables and $Z = X + Y$, then the PMF of Z is

$$p_Z(z) = \sum_x p_X(x) p_Y(z - x).$$

Proposition (Continuous case) Let X, Y be continuous independent random variables and $Z = X + Y$, then the PDF of Z is

$$f_Z(z) = \int_{-\infty}^{\infty} f_X(x) f_Y(z - x) \mathrm{d}x.$$

Proposition (Sum of independent normal r.v.) Let $X \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and $Y \sim \mathcal{N}(\mu_y, \sigma_y^2)$ independent. Then $Z = X + Y \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_y^2)$.

Definition (Covariance) We define the covariance of random variables X, Y as

$$\mathrm{Cov}(X, Y) \triangleq \mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right].$$

Properties (Properties of covariance)

- If X, Y are independent, then $\mathrm{Cov}(X, Y) = 0$.
- $\mathrm{Cov}(X, X) = \mathrm{Var}(X)$.
- $\mathrm{Cov}(aX + b, Y) = a\,\mathrm{Cov}(X, Y)$.
- $\mathrm{Cov}(X, Y + Z) = \mathrm{Cov}(X, Y) + \mathrm{Cov}(X, Z)$.
- $\mathrm{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$.

Proposition (Variance of a sum of r.v.)

$$\mathrm{Var}(X_1 + \cdots + X_n) = \sum_i \mathrm{Var}(X_i) + \sum_{i \neq j} \mathrm{Cov}(X_i, X_j).$$

Definition (Correlation coefficient) We define the correlation coefficient of random variables X, Y, with $\sigma_X, \sigma_Y > 0$, as

$$\rho(X, Y) \triangleq \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}.$$

Properties (Properties of the correlation coefficient)

- $-1 \leq \rho \leq 1$.
- If X, Y are independent, then $\rho = 0$.
- $|\rho| = 1$ if and only if $X - \mathbb{E}[X] = c(Y - \mathbb{E}[Y])$.
- $\rho(aX + b, Y) = \mathrm{sign}(a)\rho(X, Y)$.

Conditional expectation and variance, sum of random number of r.v.

Definition (Conditional expectation as a random variable) Given random variables X, Y the conditional expectation $\mathbb{E}[X|Y]$ is the random variable that takes the value $\mathbb{E}[X|Y = y]$ whenever $Y = y$.

Theorem (Law of iterated expectations)

$$\mathbb{E}\left[\mathbb{E}[X|Y]\right] = \mathbb{E}[X].$$

Definition (Conditional variance as a random variable) Given random variables X, Y the conditional variance $\mathrm{Var}(X|Y)$ is the random variable that takes the value $\mathrm{Var}(X|Y = y)$ whenever $Y = y$.

Theorem (Law of total variance)

$$\mathrm{Var}(X) = \mathbb{E}\left[\mathrm{Var}(X|Y)\right] + \mathrm{Var}\left(\mathbb{E}[X|Y]\right).$$

Proposition (Sum of a random number of independent r.v.)
Let N be a nonnegative integer random variable.
Let $X, X_1, X_2, \ldots, X_N$ be i.i.d. random variables.
Let $Y = \sum_i X_i$. Then

$$\mathbb{E}[Y] = \mathbb{E}[N]\mathbb{E}[X],$$
$$\mathrm{Var}(Y) = \mathbb{E}[N]\,\mathrm{Var}(X) + (\mathbb{E}[X])^2\,\mathrm{Var}(N).$$

Convergence of random variables

Inequalities, convergence, and the Weak Law of Large Numbers

Theorem (Markov inequality) Given a random variable $X \geq 0$ and, for every $a > 0$ we have

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}.$$

Theorem (Chebyshev inequality) Given a random variable X with $\mathbb{E}[X] = \mu$ and $\mathrm{Var}(X) = \sigma^2$, for every $\epsilon > 0$ we have

$$\mathbb{P}\left(|X - \mu| \geq \epsilon\right) \leq \frac{\sigma^2}{\epsilon^2}.$$

Theorem (Weak Law of Large Number (WLLN)) Given a sequence of i.i.d. random variables $\{X_1, X_2, \ldots\}$ with $\mathbb{E}[X_i] = \mu$ and $\mathrm{Var}(X_i) = \sigma^2$, we define

$$M_n = \frac{1}{n} \sum_{i=1}^{n} X_i,$$

for every $\epsilon > 0$ we have

$$\lim_{n \to \infty} \mathbb{P}\left(|M_n - \mu| \geq \epsilon\right) = 0.$$

Definition (Convergence in probability) A sequence of random variables $\{Y_i\}$ converges in probability to the random variable Y if

$$\lim_{n \to \infty} \mathbb{P}\left(|Y_i - Y| \geq \epsilon\right) = 0,$$

for every $\epsilon > 0$.

Properties (Properties of convergence in probability) If $X_n \to a$ and $Y_n \to b$ in probability, then

- $X_n + Y_n \to a + b$.
- If g is a continuous function, then $g(X_n) \to g(a)$.
- $\mathbb{E}[X_n]$ does not always converge to a.

The Central Limit Theorem

Theorem (Central Limit Theorem (CLT)) Given a sequence of independent random variables $\{X_1, X_2, \ldots\}$ with $\mathbb{E}[X_i] = \mu$ and $\mathrm{Var}(X_i) = \sigma^2$, we define

$$Z_n = \frac{1}{\sigma\sqrt{n}} \sum_{i=1}^{n} (X_i - \mu).$$

Then, for every z, we have

$$\lim_{n \to \infty} \mathbb{P}(Z_n \leq z) = \mathbb{P}(Z \leq z),$$

where $Z \sim \mathcal{N}(0, 1)$.

Corollary (Normal approximation of a binomial) Let $X \sim Bin(n, p)$ with n large. Then S_n can be approximated by $Z \sim \mathcal{N}\left(np, np(1 - p)\right)$.

Remark (De Moivre-Laplace 1/2 approximation) Let $X \sim Bin$, then $\mathbb{P}(X = i) = \mathbb{P}\left(i - \frac{1}{2} \leq X \leq i + \frac{1}{2}\right)$ and we can use the CLT to approximate the PMF of X.

Super VIP Cheatsheet: Machine Learning

Afshine Amidi and Shervine Amidi

October 6, 2018

Contents

1 Supervised Learning

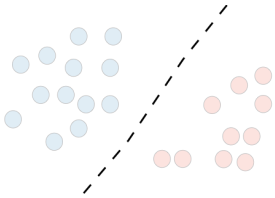
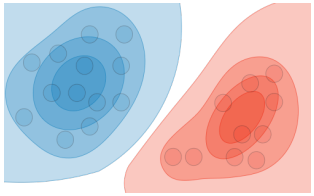
1.1 Introduction to Supervised Learning

Given a set of data points $\{x^{(1)},...,x^{(m)}\}$ associated to a set of outcomes $\{y^{(1)},...,y^{(m)}\}$, we want to build a classifier that learns how to predict y from x.

❒ **Type of prediction** – The different types of predictive models are summed up in the table below:

	Regression	Classifier
Outcome	Continuous	Class
Examples	Linear regression	Logistic regression, SVM, Naive Bayes

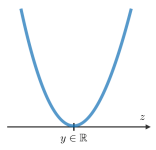
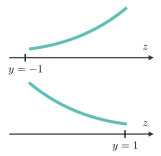
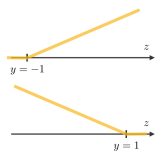
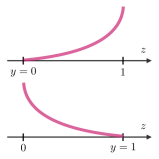
❒ **Type of model** – The different models are summed up in the table below:

	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

1.2 Notations and general concepts

❒ **Hypothesis** – The hypothesis is noted h_θ and is the model that we choose. For a given input data $x^{(i)}$, the model prediction output is $h_\theta(x^{(i)})$.

❒ **Loss function** – A loss function is a function $L:(z,y)\in\mathbb{R}\times Y\longmapsto L(z,y)\in\mathbb{R}$ that takes as inputs the predicted value z corresponding to the real data value y and outputs how different they are. The common loss functions are summed up in the table below:

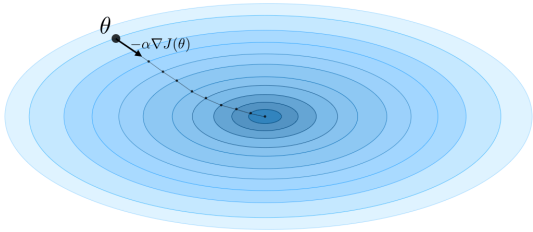
Least squared	Logistic	Hinge	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0,1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

❒ **Cost function** – The cost function J is commonly used to assess the performance of a model, and is defined with the loss function L as follows:

$$\boxed{J(\theta)=\sum_{i=1}^{m}L(h_\theta(x^{(i)}),y^{(i)})}$$

❒ **Gradient descent** – By noting $\alpha\in\mathbb{R}$ the learning rate, the update rule for gradient descent is expressed with the learning rate and the cost function J as follows:

$$\boxed{\theta\longleftarrow\theta-\alpha\nabla J(\theta)}$$



Remark: Stochastic gradient descent (SGD) is updating the parameter based on each training example, and batch gradient descent is on a batch of training examples.

❒ **Likelihood** – The likelihood of a model $L(\theta)$ given parameters θ is used to find the optimal parameters θ through maximizing the likelihood. In practice, we use the log-likelihood $\ell(\theta)=\log(L(\theta))$ which is easier to optimize. We have:

$$\boxed{\theta^{\text{opt}}=\underset{\theta}{\arg\max}\ L(\theta)}$$

❒ **Newton's algorithm** – The Newton's algorithm is a numerical method that finds θ such that $\ell'(\theta)=0$. Its update rule is as follows:

$$\boxed{\theta\leftarrow\theta-\frac{\ell'(\theta)}{\ell''(\theta)}}$$

Remark: the multidimensional generalization, also known as the Newton-Raphson method, has the following update rule:

$$\theta\leftarrow\theta-\left(\nabla^2_\theta\ell(\theta)\right)^{-1}\nabla_\theta\ell(\theta)$$

1.3 Linear models

1.3.1 Linear regression

We assume here that $y|x;\theta\sim\mathcal{N}(\mu,\sigma^2)$

❒ **Normal equations** – By noting X the matrix design, the value of θ that minimizes the cost function is a closed-form solution such that:

$$\boxed{\theta=(X^TX)^{-1}X^Ty}$$

❒ **LMS algorithm** – By noting α the learning rate, the update rule of the Least Mean Squares (LMS) algorithm for a training set of m data points, which is also known as the Widrow-Hoff learning rule, is as follows:

$$\boxed{\forall j, \quad \theta_j \leftarrow \theta_j + \alpha \sum_{i=1}^{m} \left[y^{(i)} - h_\theta(x^{(i)})\right] x_j^{(i)}}$$

Remark: the update rule is a particular case of the gradient ascent.

❒ **LWR** – Locally Weighted Regression, also known as LWR, is a variant of linear regression that weights each training example in its cost function by $w^{(i)}(x)$, which is defined with parameter $\tau \in \mathbb{R}$ as:

$$\boxed{w^{(i)}(x) = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)}$$

1.3.2 Classification and logistic regression

❒ **Sigmoid function** – The sigmoid function g, also known as the logistic function, is defined as follows:

$$\forall z \in \mathbb{R}, \quad \boxed{g(z) = \frac{1}{1+e^{-z}} \in]0,1[}$$

❒ **Logistic regression** – We assume here that $y|x;\theta \sim \text{Bernoulli}(\phi)$. We have the following form:

$$\boxed{\phi = p(y=1|x;\theta) = \frac{1}{1+\exp(-\theta^T x)} = g(\theta^T x)}$$

Remark: there is no closed form solution for the case of logistic regressions.

❒ **Softmax regression** – A softmax regression, also called a multiclass logistic regression, is used to generalize logistic regression when there are more than 2 outcome classes. By convention, we set $\theta_K = 0$, which makes the Bernoulli parameter ϕ_i of each class i equal to:

$$\boxed{\phi_i = \frac{\exp(\theta_i^T x)}{\sum_{j=1}^{K} \exp(\theta_j^T x)}}$$

1.3.3 Generalized Linear Models

❒ **Exponential family** – A class of distributions is said to be in the exponential family if it can be written in terms of a natural parameter, also called the canonical parameter or link function, η, a sufficient statistic $T(y)$ and a log-partition function $a(\eta)$ as follows:

$$\boxed{p(y;\eta) = b(y)\exp(\eta T(y) - a(\eta))}$$

Remark: we will often have $T(y) = y$. Also, $\exp(-a(\eta))$ can be seen as a normalization parameter that will make sure that the probabilities sum to one.

Here are the most common exponential distributions summed up in the following table:

Distribution	η	$T(y)$	$a(\eta)$	$b(y)$
Bernoulli	$\log\left(\frac{\phi}{1-\phi}\right)$	y	$\log(1+\exp(\eta))$	1
Gaussian	μ	y	$\frac{\eta^2}{2}$	$\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{y^2}{2}\right)$
Poisson	$\log(\lambda)$	y	e^η	$\frac{1}{y!}$
Geometric	$\log(1-\phi)$	y	$\log\left(\frac{e^\eta}{1-e^\eta}\right)$	1

❒ **Assumptions of GLMs** – Generalized Linear Models (GLM) aim at predicting a random variable y as a function fo $x \in \mathbb{R}^{n+1}$ and rely on the following 3 assumptions:

(1) $\boxed{y|x;\theta \sim \text{ExpFamily}(\eta)}$ (2) $\boxed{h_\theta(x) = E[y|x;\theta]}$ (3) $\boxed{\eta = \theta^T x}$

Remark: ordinary least squares and logistic regression are special cases of generalized linear models.

1.4 Support Vector Machines

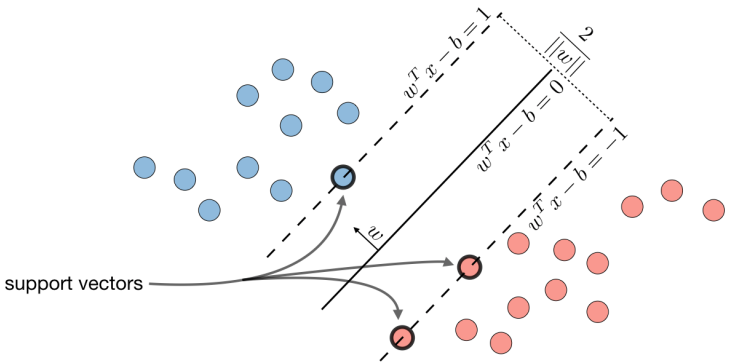
The goal of support vector machines is to find the line that maximizes the minimum distance to the line.

❒ **Optimal margin classifier** – The optimal margin classifier h is such that:

$$\boxed{h(x) = \text{sign}(w^T x - b)}$$

where $(w,b) \in \mathbb{R}^n \times \mathbb{R}$ is the solution of the following optimization problem:

$$\boxed{\min \frac{1}{2}||w||^2} \quad \text{such that} \quad \boxed{y^{(i)}(w^T x^{(i)} - b) \geqslant 1}$$



Remark: the line is defined as $\boxed{w^T x - b = 0}$.

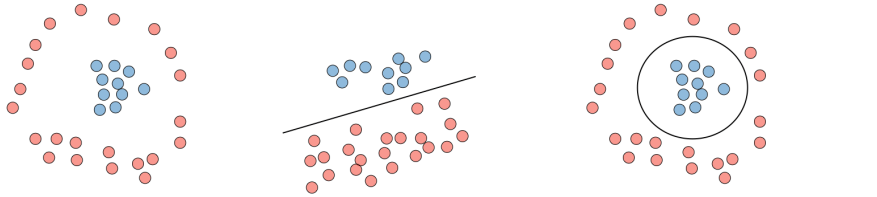
❒ **Hinge loss** – The hinge loss is used in the setting of SVMs and is defined as follows:

$$\boxed{L(z,y) = [1-yz]_+ = \max(0, 1-yz)}$$

❒ **Kernel** – Given a feature mapping ϕ, we define the kernel K to be defined as:

$$\boxed{K(x,z) = \phi(x)^T\phi(z)}$$

In practice, the kernel K defined by $K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$ is called the Gaussian kernel and is commonly used.



Remark: we say that we use the "kernel trick" to compute the cost function using the kernel because we actually don't need to know the explicit mapping ϕ, which is often very complicated. Instead, only the values $K(x,z)$ are needed.

❒ **Lagrangian** – We define the Lagrangian $\mathcal{L}(w,b)$ as follows:

$$\boxed{\mathcal{L}(w,b) = f(w) + \sum_{i=1}^{l}\beta_i h_i(w)}$$

Remark: the coefficients β_i are called the Lagrange multipliers.

1.5 Generative Learning

A generative model first tries to learn how the data is generated by estimating $P(x|y)$, which we can then use to estimate $P(y|x)$ by using Bayes' rule.

1.5.1 Gaussian Discriminant Analysis

❒ **Setting** – The Gaussian Discriminant Analysis assumes that y and $x|y=0$ and $x|y=1$ are such that:

$$\boxed{y \sim \text{Bernoulli}(\phi)}$$

$$\boxed{x|y=0 \sim \mathcal{N}(\mu_0,\Sigma)} \quad \text{and} \quad \boxed{x|y=1 \sim \mathcal{N}(\mu_1,\Sigma)}$$

❒ **Estimation** – The following table sums up the estimates that we find when maximizing the likelihood:

$\widehat{\phi}$	$\widehat{\mu_j}$ $(j=0,1)$	$\widehat{\Sigma}$
$\frac{1}{m}\sum_{i=1}^{m} 1_{\{y^{(i)}=1\}}$	$\frac{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}x^{(i)}}{\sum_{i=1}^{m} 1_{\{y^{(i)}=j\}}}$	$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu_{y^{(i)}})(x^{(i)}-\mu_{y^{(i)}})^T$

1.5.2 Naive Bayes

❒ **Assumption** – The Naive Bayes model supposes that the features of each data point are all independent:

$$\boxed{P(x|y) = P(x_1,x_2,...|y) = P(x_1|y)P(x_2|y)... = \prod_{i=1}^{n} P(x_i|y)}$$

❒ **Solutions** – Maximizing the log-likelihood gives the following solutions, with $k \in \{0,1\}$, $l \in [\![1,L]\!]$

$$\boxed{P(y=k) = \frac{1}{m} \times \#\{j|y^{(j)}=k\}} \quad \text{and} \quad \boxed{P(x_i=l|y=k) = \frac{\#\{j|y^{(j)}=k \text{ and } x_i^{(j)}=l\}}{\#\{j|y^{(j)}=k\}}}$$

Remark: Naive Bayes is widely used for text classification and spam detection.

1.6 Tree-based and ensemble methods

These methods can be used for both regression and classification problems.

❒ **CART** – Classification and Regression Trees (CART), commonly known as decision trees, can be represented as binary trees. They have the advantage to be very interpretable.

❒ **Random forest** – It is a tree-based technique that uses a high number of decision trees built out of randomly selected sets of features. Contrary to the simple decision tree, it is highly uninterpretable but its generally good performance makes it a popular algorithm.

Remark: random forests are a type of ensemble methods.

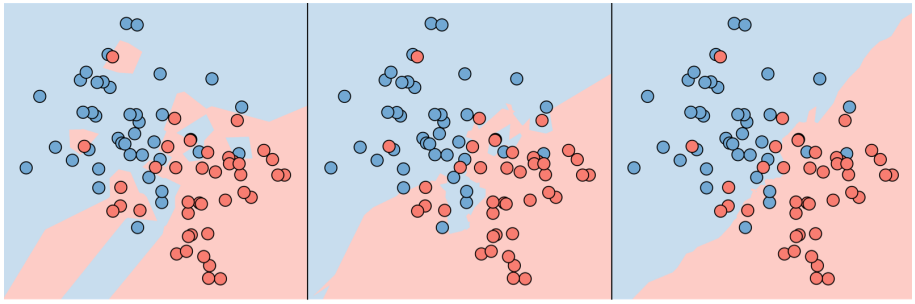
❒ **Boosting** – The idea of boosting methods is to combine several weak learners to form a stronger one. The main ones are summed up in the table below:

Adaptive boosting	Gradient boosting
- High weights are put on errors to improve at the next boosting step - Known as Adaboost	- Weak learners trained on remaining errors

1.7 Other non-parametric approaches

❒ **k-nearest neighbors** – The k-nearest neighbors algorithm, commonly known as k-NN, is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.

Remark: The higher the parameter k, the higher the bias, and the lower the parameter k, the higher the variance.

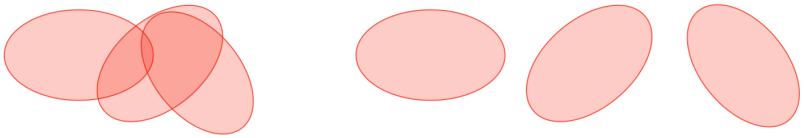


$k = 1$ $k = 3$ $k = 11$

1.8 Learning Theory

❒ **Union bound** – Let $A_1, ..., A_k$ be k events. We have:

$$\boxed{P(A_1 \cup ... \cup A_k) \leqslant P(A_1) + ... + P(A_k)}$$



$A_1 \cup A_2 \cup A_3$ A_1 A_2 A_3

❒ **Hoeffding inequality** – Let $Z_1, .., Z_m$ be m iid variables drawn from a Bernoulli distribution of parameter ϕ. Let $\widehat{\phi}$ be their sample mean and $\gamma > 0$ fixed. We have:

$$\boxed{P(|\phi - \widehat{\phi}| > \gamma) \leqslant 2\exp(-2\gamma^2 m)}$$

Remark: this inequality is also known as the Chernoff bound.

❒ **Training error** – For a given classifier h, we define the training error $\widehat{\epsilon}(h)$, also known as the empirical risk or empirical error, to be as follows:

$$\boxed{\widehat{\epsilon}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{\{h(x^{(i)}) \neq y^{(i)}\}}}$$

❒ **Probably Approximately Correct (PAC)** – PAC is a framework under which numerous results on learning theory were proved, and has the following set of assumptions:

- the training and testing sets follow the same distribution
- the training examples are drawn independently

❒ **Shattering** – Given a set $S = \{x^{(1)},...,x^{(d)}\}$, and a set of classifiers $\mathcal{H}$, we say that $\mathcal{H}$ shatters S if for any set of labels $\{y^{(1)}, ..., y^{(d)}\}$, we have:

$$\boxed{\exists h \in \mathcal{H}, \quad \forall i \in [\![1,d]\!], \quad h(x^{(i)}) = y^{(i)}}$$

❒ **Upper bound theorem** – Let $\mathcal{H}$ be a finite hypothesis class such that $|\mathcal{H}| = k$ and let δ and the sample size m be fixed. Then, with probability of at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + 2\sqrt{\frac{1}{2m}\log\left(\frac{2k}{\delta}\right)}}$$

❒ **VC dimension** – The Vapnik-Chervonenkis (VC) dimension of a given infinite hypothesis class $\mathcal{H}$, noted $\text{VC}(\mathcal{H})$ is the size of the largest set that is shattered by $\mathcal{H}$.

Remark: the VC dimension of $\mathcal{H} = \{$set of linear classifiers in 2 dimensions$\}$ is 3.

❒ **Theorem (Vapnik)** – Let $\mathcal{H}$ be given, with $\text{VC}(\mathcal{H}) = d$ and m the number of training examples. With probability at least $1 - \delta$, we have:

$$\boxed{\epsilon(\widehat{h}) \leqslant \left(\min_{h \in \mathcal{H}} \epsilon(h)\right) + O\left(\sqrt{\frac{d}{m}\log\left(\frac{m}{d}\right) + \frac{1}{m}\log\left(\frac{1}{\delta}\right)}\right)}$$

2 Unsupervised Learning

2.1 Introduction to Unsupervised Learning

❒ **Motivation** – The goal of unsupervised learning is to find hidden patterns in unlabeled data $\{x^{(1)},...,x^{(m)}\}$.

❒ **Jensen's inequality** – Let f be a convex function and X a random variable. We have the following inequality:

$$\boxed{E[f(X)] \geqslant f(E[X])}$$

2.2 Clustering

2.2.1 Expectation-Maximization

❒ **Latent variables** – Latent variables are hidden/unobserved variables that make estimation problems difficult, and are often denoted z. Here are the most common settings where there are latent variables:

Setting	Latent variable z	$x\|z$	Comments
Mixture of k Gaussians	Multinomial(ϕ)	$\mathcal{N}(\mu_j,\Sigma_j)$	$\mu_j \in \mathbb{R}^n, \phi \in \mathbb{R}^k$
Factor analysis	$\mathcal{N}(0,I)$	$\mathcal{N}(\mu+\Lambda z,\psi)$	$\mu_j \in \mathbb{R}^n$

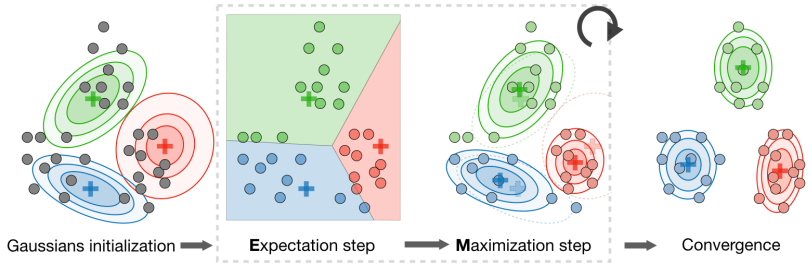
❒ **Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $Q_i(z^{(i)})$ that each data point $x^{(i)}$ came from a particular cluster $z^{(i)}$ as follows:

$$\boxed{Q_i(z^{(i)}) = P(z^{(i)}|x^{(i)};\theta)}$$

- M-step: Use the posterior probabilities $Q_i(z^{(i)})$ as cluster specific weights on data points $x^{(i)}$ to separately re-estimate each cluster model as follows:

$$\boxed{\theta_i = \underset{\theta}{\text{argmax}} \sum_i \int_{z^{(i)}} Q_i(z^{(i)}) \log\left(\frac{P(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}\right) dz^{(i)}}$$

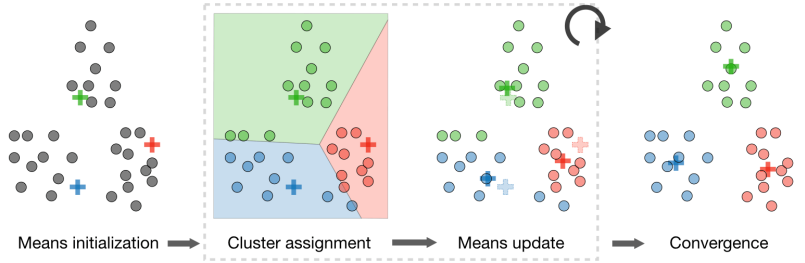


2.2.2 k-means clustering

We note $c^{(i)}$ the cluster of data point i and μ_j the center of cluster j.

❒ **Algorithm** – After randomly initializing the cluster centroids $\mu_1,\mu_2,...,\mu_k \in \mathbb{R}^n$, the k-means algorithm repeats the following step until convergence:

$$\boxed{c^{(i)} = \underset{j}{\arg\min}||x^{(i)}-\mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}}$$



❒ **Distortion function** – In order to see if the algorithm converges, we look at the distortion function defined as follows:

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

2.2.3 Hierarchical clustering

❒ **Algorithm** – It is a clustering algorithm with an agglomerative hierarchical approach that build nested clusters in a successive manner.

❒ **Types** – There are different sorts of hierarchical clustering algorithms that aims at optimizing different objective functions, which is summed up in the table below:

Ward linkage	Average linkage	Complete linkage
Minimize within cluster distance	Minimize average distance between cluster pairs	Minimize maximum distance of between cluster pairs

2.2.4 Clustering assessment metrics

In an unsupervised learning setting, it is often hard to assess the performance of a model since we don't have the ground truth labels as was the case in the supervised learning setting.

❒ **Silhouette coefficient** – By noting a and b the mean distance between a sample and all other points in the same class, and between a sample and all other points in the next nearest cluster, the silhouette coefficient s for a single sample is defined as follows:

$$\boxed{s = \frac{b-a}{\max(a,b)}}$$

❒ **Calinski-Harabaz index** – By noting k the number of clusters, B_k and W_k the between and within-clustering dispersion matrices respectively defined as

$$B_k = \sum_{j=1}^{k} n_{c^{(i)}}(\mu_{c^{(i)}} - \mu)(\mu_{c^{(i)}} - \mu)^T, \qquad W_k = \sum_{i=1}^{m}(x^{(i)} - \mu_{c^{(i)}})(x^{(i)} - \mu_{c^{(i)}})^T$$

the Calinski-Harabaz index $s(k)$ indicates how well a clustering model defines its clusters, such that the higher the score, the more dense and well separated the clusters are. It is defined as follows:

$$\boxed{s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N-k}{k-1}}$$

2.3 Dimension reduction

2.3.1 Principal component analysis

It is a dimension reduction technique that finds the variance maximizing directions onto which to project the data.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n\times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$\boxed{Az = \lambda z}$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n\times n}$. By noting $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, we have:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

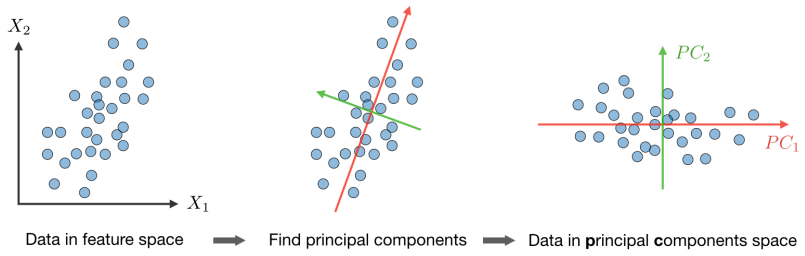
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

❒ **Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:

- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

- Step 2: Compute $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} \in \mathbb{R}^{n\times n}$, which is symmetric with real eigenvalues.
- Step 3: Compute $u_1, ..., u_k \in \mathbb{R}^n$ the k orthogonal principal eigenvectors of Σ, i.e. the orthogonal eigenvectors of the k largest eigenvalues.
- Step 4: Project the data on $\text{span}_{\mathbb{R}}(u_1,...,u_k)$. This procedure maximizes the variance among all k-dimensional spaces.



2.3.2 Independent component analysis

It is a technique meant to find the underlying generating sources.

❒ **Assumptions** – We assume that our data x has been generated by the n-dimensional source vector $s = (s_1,...,s_n)$, where s_i are independent random variables, via a mixing and non-singular matrix A as follows:

$$\boxed{x = As}$$

The goal is to find the unmixing matrix $W = A^{-1}$ by an update rule.

❒ **Bell and Sejnowski ICA algorithm** – This algorithm finds the unmixing matrix W by following the steps below:

- Write the probability of $x = As = W^{-1}s$ as:

$$p(x) = \prod_{i=1}^{n} p_s(w_i^T x) \cdot |W|$$

- Write the log likelihood given our training data $\{x^{(i)}, i \in [\![1,m]\!]\}$ and by noting g the sigmoid function as:

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

Therefore, the stochastic gradient ascent learning rule is such that for each training example $x^{(i)}$, we update W as follows:

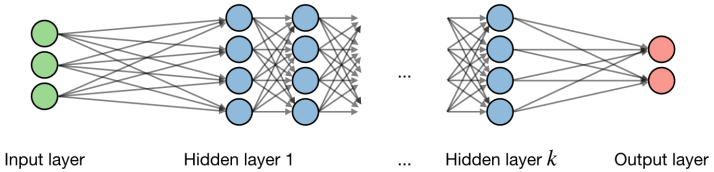
$$\boxed{W \longleftarrow W + \alpha\left(\begin{pmatrix} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{pmatrix} x^{(i)^T} + (W^T)^{-1}\right)}$$

3 Deep Learning

3.1 Neural Networks

Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks.

❒ **Architecture** – The vocabulary around neural networks architectures is described in the figure below:

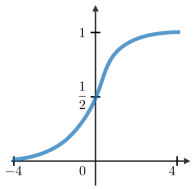
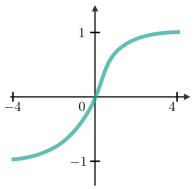
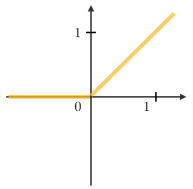
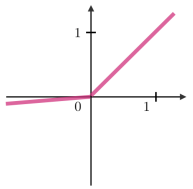


By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

where we note w, b, z the weight, bias and output respectively.

❒ **Activation function** – Activation functions are used at the end of a hidden unit to introduce non-linear complexities to the model. Here are the most common ones:

Sigmoid	Tanh	ReLU	Leaky ReLU
$g(z) = \dfrac{1}{1+e^{-z}}$	$g(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$
			

❒ **Cross-entropy loss** – In the context of neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

❒ **Learning rate** – The learning rate, often noted η, indicates at which pace the weights get updated. This can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to weight w is computed using chain rule and is of the following form:

$$\boxed{\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w}}$$

As a result, the weight is updated as follows:

$$\boxed{w \longleftarrow w - \eta \frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data.
- Step 2: Perform forward propagation to obtain the corresponding loss.
- Step 3: Backpropagate the loss to get the gradients.
- Step 4: Use the gradients to update the weights of the network.

❒ **Dropout** – Dropout is a technique meant at preventing overfitting the training data by dropping out units in a neural network. In practice, neurons are either dropped with probability p or kept with probability $1-p$.

3.2 Convolutional Neural Networks

❒ **Convolutional layer requirement** – By noting W the input volume size, F the size of the convolutional layer neurons, P the amount of zero padding, then the number of neurons N that fit in a given volume is such that:

$$\boxed{N = \frac{W - F + 2P}{S} + 1}$$

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

3.3 Recurrent Neural Networks

❒ **Types of gates** – Here are the different types of gates that we encounter in a typical recurrent neural network:

Input gate	Forget gate	Output gate	Gate
Write to cell or not?	Erase a cell or not?	Reveal a cell or not?	How much writing?

❒ **LSTM** – A long short-term memory (LSTM) network is a type of RNN model that avoids the vanishing gradient problem by adding 'forget' gates.

3.4 Reinforcement Learning and Control

The goal of reinforcement learning is for an agent to learn how to evolve in an environment.

❒ **Markov decision processes** – A Markov decision process (MDP) is a 5-tuple $(S,A,\{P_{sa}\},\gamma,R)$ where:

- $\mathcal{S}$ is the set of states
- $\mathcal{A}$ is the set of actions
- $\{P_{sa}\}$ are the state transition probabilities for $s \in \mathcal{S}$ and $a \in \mathcal{A}$
- $\gamma \in [0,1[$ is the discount factor
- $R : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ or $R : \mathcal{S} \longrightarrow \mathbb{R}$ is the reward function that the algorithm wants to maximize

❒ **Policy** – A policy π is a function $\pi : \mathcal{S} \longrightarrow \mathcal{A}$ that maps states to actions.

Remark: we say that we execute a given policy π if given a state s we take the action $a = \pi(s)$.

❒ **Value function** – For a given policy π and a given state s, we define the value function V^π as follows:

$$\boxed{V^\pi(s) = E\Big[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...|s_0 = s,\pi\Big]}$$

❒ **Bellman equation** – The optimal Bellman equations characterizes the value function V^{π^*} of the optimal policy π^*:

$$\boxed{V^{\pi^*}(s) = R(s) + \max_{a\in\mathcal{A}} \gamma \sum_{s'\in S} P_{sa}(s')V^{\pi^*}(s')}$$

Remark: we note that the optimal policy π^* for a given state s is such that:

$$\boxed{\pi^*(s) = \underset{a\in\mathcal{A}}{\text{argmax}} \sum_{s'\in\mathcal{S}} P_{sa}(s')V^*(s')}$$

❒ **Value iteration algorithm** – The value iteration algorithm is in two steps:

- We initialize the value:

$$\boxed{V_0(s) = 0}$$

- We iterate the value based on the values before:

$$\boxed{V_{i+1}(s) = R(s) + \max_{a\in\mathcal{A}} \left[\sum_{s'\in\mathcal{S}} \gamma P_{sa}(s')V_i(s')\right]}$$

❒ **Maximum likelihood estimate** – The maximum likelihood estimates for the state transition probabilities are as follows:

$$\boxed{P_{sa}(s') = \frac{\#\text{times took action } a \text{ in state } s \text{ and got to } s'}{\#\text{times took action } a \text{ in state } s}}$$

❒ **Q-learning** – Q-learning is a model-free estimation of Q, which is done as follows:

$$\boxed{Q(s,a) \leftarrow Q(s,a) + \alpha\Big[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big]}$$

4 Machine Learning Tips and Tricks

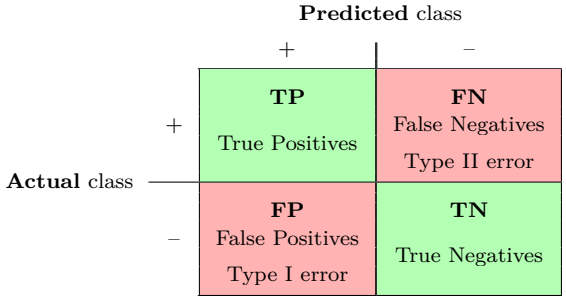
4.1 Metrics

Given a set of data points $\{x^{(1)}, ..., x^{(m)}\}$, where each $x^{(i)}$ has n features, associated to a set of outcomes $\{y^{(1)}, ..., y^{(m)}\}$, we want to assess a given classifier that learns how to predict y from x.

4.1.1 Classification

In a context of a binary classification, here are the main metrics that are important to track to assess the performance of the model.

❒ **Confusion matrix** – The confusion matrix is used to have a more complete picture when assessing the performance of a model. It is defined as follows:



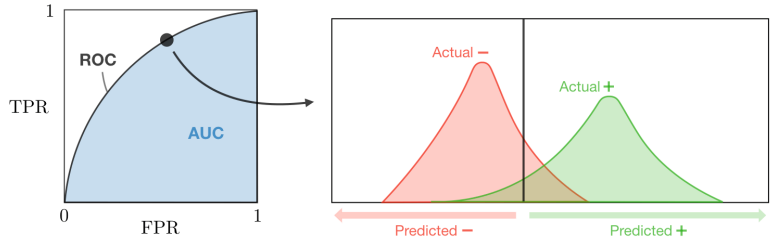
❒ **Main metrics** – The following metrics are commonly used to assess the performance of classification models:

Metric	Formula	Interpretation
Accuracy	$\frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$	Overall performance of model
Precision	$\frac{\text{TP}}{\text{TP}+\text{FP}}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Coverage of actual positive sample
Specificity	$\frac{\text{TN}}{\text{TN}+\text{FP}}$	Coverage of actual negative sample
F1 score	$\frac{2\text{TP}}{2\text{TP}+\text{FP}+\text{FN}}$	Hybrid metric useful for unbalanced classes

❒ **ROC** – The receiver operating curve, also noted ROC, is the plot of TPR versus FPR by varying the threshold. These metrics are are summed up in the table below:

Metric	Formula	Equivalent
True Positive Rate TPR	$\frac{\text{TP}}{\text{TP}+\text{FN}}$	Recall, sensitivity
False Positive Rate FPR	$\frac{\text{FP}}{\text{TN}+\text{FP}}$	1-specificity

❒ **AUC** – The area under the receiving operating curve, also noted AUC or AUROC, is the area below the ROC as shown in the following figure:



4.1.2 Regression

❒ **Basic metrics** – Given a regression model f, the following metrics are commonly used to assess the performance of the model:

Total sum of squares	Explained sum of squares	Residual sum of squares
$\text{SS}_{\text{tot}} = \sum_{i=1}^{m}(y_i - \overline{y})^2$	$\text{SS}_{\text{reg}} = \sum_{i=1}^{m}(f(x_i) - \overline{y})^2$	$\text{SS}_{\text{res}} = \sum_{i=1}^{m}(y_i - f(x_i))^2$

❒ **Coefficient of determination** – The coefficient of determination, often noted R^2 or r^2, provides a measure of how well the observed outcomes are replicated by the model and is defined as follows:

$$\boxed{R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}}$$

❒ **Main metrics** – The following metrics are commonly used to assess the performance of regression models, by taking into account the number of variables n that they take into consideration:

Mallow's Cp	AIC	BIC	Adjusted R^2
$\frac{\text{SS}_{\text{res}} + 2(n+1)\widehat{\sigma}^2}{m}$	$2\big[(n+2) - \log(L)\big]$	$\log(m)(n+2) - 2\log(L)$	$1 - \frac{(1-R^2)(m-1)}{m-n-1}$

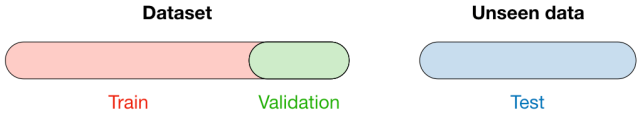
where L is the likelihood and $\widehat{\sigma}^2$ is an estimate of the variance associated with each response.

4.2 Model selection

❒ **Vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80% of the dataset	- Model is assessed - Usually 20% of the dataset - Also called hold-out or development set	- Model gives predictions - Unseen data

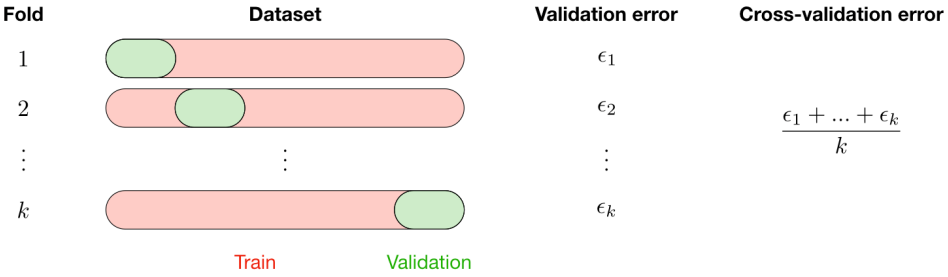
Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



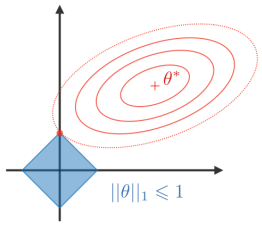
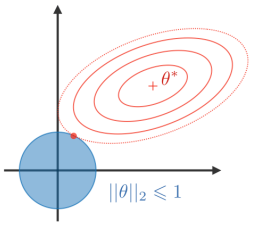
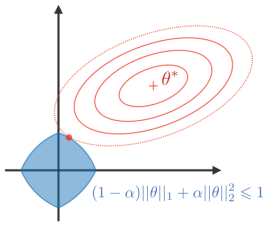
❒ **Cross-validation** – Cross-validation, also noted CV, is a method that is used to select a model that does not rely too much on the initial training set. The different types are summed up in the table below:

k-fold	Leave-p-out
- Training on $k-1$ folds and assessment on the remaining one - Generally $k = 5$ or 10	- Training on $n-p$ observations and assessment on the p remaining ones - Case $p = 1$ is called leave-one-out

The most commonly used method is called k-fold cross-validation and splits the training data into k folds to validate the model on one fold while training the model on the $k-1$ other folds, all of this k times. The error is then averaged over the k folds and is named cross-validation error.



❒ **Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}$, $\alpha \in [0,1]$

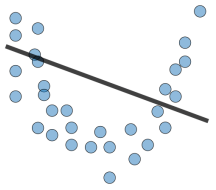
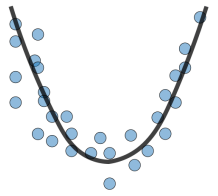
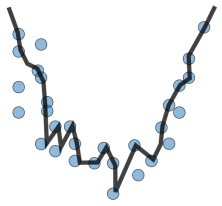
❒ **Model selection** – Train model on training set, then evaluate on the development set, then pick best performance model on the development set, and retrain all of that model on the whole training set.

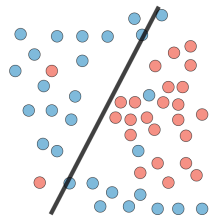
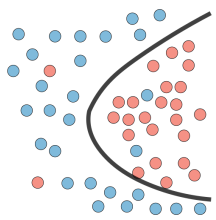
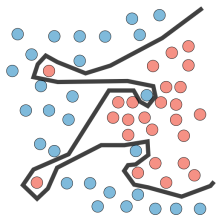
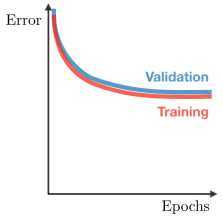
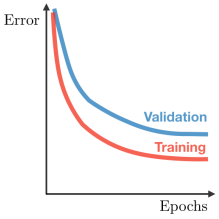
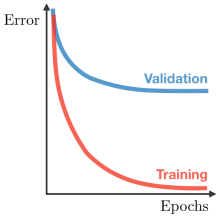
4.3 Diagnostics

❒ **Bias** – The bias of a model is the difference between the expected prediction and the correct model that we try to predict for given data points.

❒ **Variance** – The variance of a model is the variability of the model prediction for given data points.

❒ **Bias/variance tradeoff** – The simpler the model, the higher the bias, and the more complex the model, the higher the variance.

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			

Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data

❒ **Error analysis** – Error analysis is analyzing the root cause of the difference in performance between the current and the perfect models.

❒ **Ablative analysis** – Ablative analysis is analyzing the root cause of the difference in performance between the current and the baseline models.

5 Refreshers

5.1 Probabilities and Statistics

5.1.1 Introduction to Probability and Combinatorics

❒ **Sample space** – The set of all possible outcomes of an experiment is known as the sample space of the experiment and is denoted by S.

❒ **Event** – Any subset E of the sample space is known as an event. That is, an event is a set consisting of possible outcomes of the experiment. If the outcome of the experiment is contained in E, then we say that E has occurred.

❒ **Axioms of probability** – For each event E, we denote $P(E)$ as the probability of event E occuring. By noting $E_1,...,E_n$ mutually exclusive events, we have the 3 following axioms:

$$(1) \quad \boxed{0 \leqslant P(E) \leqslant 1} \quad (2) \quad \boxed{P(S) = 1} \quad (3) \quad \boxed{P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{i=1}^{n} P(E_i)}$$

❒ **Permutation** – A permutation is an arrangement of r objects from a pool of n objects, in a given order. The number of such arrangements is given by $P(n,r)$, defined as:

$$\boxed{P(n,r) = \frac{n!}{(n-r)!}}$$

❒ **Combination** – A combination is an arrangement of r objects from a pool of n objects, where the order does not matter. The number of such arrangements is given by $C(n,r)$, defined as:

$$\boxed{C(n,r) = \frac{P(n,r)}{r!} = \frac{n!}{r!(n-r)!}}$$

Remark: we note that for $0 \leqslant r \leqslant n$, we have $P(n,r) \geqslant C(n,r)$.

5.1.2 Conditional Probability

❒ **Bayes' rule** – For events A and B such that $P(B) > 0$, we have:

$$\boxed{P(A|B) = \frac{P(B|A)P(A)}{P(B)}}$$

Remark: we have $P(A \cap B) = P(A)P(B|A) = P(A|B)P(B)$.

❒ **Partition** – Let $\{A_i, i \in [\![1,n]\!]\}$ be such that for all i, $A_i \neq \varnothing$. We say that $\{A_i\}$ is a partition if we have:

$$\boxed{\forall i \neq j, A_i \cap A_j = \emptyset \quad \text{and} \quad \bigcup_{i=1}^{n} A_i = S}$$

Remark: for any event B in the sample space, we have $P(B) = \sum_{i=1}^{n} P(B|A_i)P(A_i)$.

❒ **Extended form of Bayes' rule** – Let $\{A_i, i \in [\![1,n]\!]\}$ be a partition of the sample space. We have:

$$\boxed{P(A_k|B) = \frac{P(B|A_k)P(A_k)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)}}$$

❒ **Independence** – Two events A and B are independent if and only if we have:

$$\boxed{P(A \cap B) = P(A)P(B)}$$

5.1.3 Random Variables

❒ **Random variable** – A random variable, often noted X, is a function that maps every element in a sample space to a real line.

❒ **Cumulative distribution function (CDF)** – The cumulative distribution function F, which is monotonically non-decreasing and is such that $\lim_{x\to-\infty} F(x) = 0$ and $\lim_{x\to+\infty} F(x) = 1$, is defined as:

$$\boxed{F(x) = P(X \leqslant x)}$$

Remark: we have $P(a < X \leqslant B) = F(b) - F(a)$.

❒ **Probability density function (PDF)** – The probability density function f is the probability that X takes on values between two adjacent realizations of the random variable.

❒ **Relationships involving the PDF and CDF** – Here are the important properties to know in the discrete (D) and the continuous (C) cases.

Case	CDF F	PDF f	Properties of PDF
(D)	$F(x) = \sum_{x_i \leqslant x} P(X = x_i)$	$f(x_j) = P(X = x_j)$	$0 \leqslant f(x_j) \leqslant 1$ and $\sum_j f(x_j) = 1$
(C)	$F(x) = \int_{-\infty}^{x} f(y)dy$	$f(x) = \frac{dF}{dx}$	$f(x) \geqslant 0$ and $\int_{-\infty}^{+\infty} f(x)dx = 1$

❒ **Variance** – The variance of a random variable, often noted $\text{Var}(X)$ or σ^2, is a measure of the spread of its distribution function. It is determined as follows:

$$\boxed{\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2}$$

❒ **Standard deviation** – The standard deviation of a random variable, often noted σ, is a measure of the spread of its distribution function which is compatible with the units of the actual random variable. It is determined as follows:

$$\boxed{\sigma = \sqrt{\text{Var}(X)}}$$

❒ **Expectation and Moments of the Distribution** – Here are the expressions of the expected value $E[X]$, generalized expected value $E[g(X)]$, k^{th} moment $E[X^k]$ and characteristic function $\psi(\omega)$ for the discrete and continuous cases:

Case	$E[X]$	$E[g(X)]$	$E[X^k]$	$\psi(\omega)$
(D)	$\sum_{i=1}^{n} x_i f(x_i)$	$\sum_{i=1}^{n} g(x_i) f(x_i)$	$\sum_{i=1}^{n} x_i^k f(x_i)$	$\sum_{i=1}^{n} f(x_i) e^{i\omega x_i}$
(C)	$\int_{-\infty}^{+\infty} x f(x)dx$	$\int_{-\infty}^{+\infty} g(x) f(x)dx$	$\int_{-\infty}^{+\infty} x^k f(x)dx$	$\int_{-\infty}^{+\infty} f(x) e^{i\omega x}dx$

Remark: we have $e^{i\omega x} = \cos(\omega x) + i\sin(\omega x)$.

❒ **Revisiting the k^{th} moment** – The k^{th} moment can also be computed with the characteristic function as follows:

$$\boxed{E[X^k] = \frac{1}{i^k}\left[\frac{\partial^k \psi}{\partial \omega^k}\right]_{\omega=0}}$$

❒ **Transformation of random variables** – Let the variables X and Y be linked by some function. By noting f_X and f_Y the distribution function of X and Y respectively, we have:

$$\boxed{f_Y(y) = f_X(x)\left|\frac{dx}{dy}\right|}$$

❒ **Leibniz integral rule** – Let g be a function of x and potentially c, and a, b boundaries that may depend on c. We have:

$$\boxed{\frac{\partial}{\partial c}\left(\int_a^b g(x)dx\right) = \frac{\partial b}{\partial c} \cdot g(b) - \frac{\partial a}{\partial c} \cdot g(a) + \int_a^b \frac{\partial g}{\partial c}(x)dx}$$

❒ **Chebyshev's inequality** – Let X be a random variable with expected value μ and standard deviation σ. For $k, \sigma > 0$, we have the following inequality:

$$\boxed{P(|X - \mu| \geqslant k\sigma) \leqslant \frac{1}{k^2}}$$

5.1.4 Jointly Distributed Random Variables

❒ **Conditional density** – The conditional density of X with respect to Y, often noted $f_{X|Y}$, is defined as follows:

$$\boxed{f_{X|Y}(x) = \frac{f_{XY}(x,y)}{f_Y(y)}}$$

❒ **Independence** – Two random variables X and Y are said to be independent if we have:

$$\boxed{f_{XY}(x,y) = f_X(x) f_Y(y)}$$

❒ **Marginal density and cumulative distribution** – From the joint density probability function f_{XY}, we have:

Case	Marginal density	Cumulative function
(D)	$f_X(x_i) = \sum_j f_{XY}(x_i,y_j)$	$F_{XY}(x,y) = \sum_{x_i \leqslant x} \sum_{y_j \leqslant y} f_{XY}(x_i,y_j)$
(C)	$f_X(x) = \int_{-\infty}^{+\infty} f_{XY}(x,y)dy$	$F_{XY}(x,y) = \int_{-\infty}^{x}\int_{-\infty}^{y} f_{XY}(x',y')dx'dy'$

❒ **Distribution of a sum of independent random variables** – Let $Y = X_1 + ... + X_n$ with $X_1, ..., X_n$ independent. We have:

$$\boxed{\psi_Y(\omega) = \prod_{k=1}^{n} \psi_{X_k}(\omega)}$$

❒ **Covariance** – We define the covariance of two random variables X and Y, that we note σ^2_{XY} or more commonly $\text{Cov}(X,Y)$, as follows:

$$\boxed{\text{Cov}(X,Y) \triangleq \sigma^2_{XY} = E[(X-\mu_X)(Y-\mu_Y)] = E[XY] - \mu_X\mu_Y}$$

❒ **Correlation** – By noting σ_X, σ_Y the standard deviations of X and Y, we define the correlation between the random variables X and Y, noted ρ_{XY}, as follows:

$$\boxed{\rho_{XY} = \frac{\sigma^2_{XY}}{\sigma_X\sigma_Y}}$$

Remarks: For any X, Y, we have $\rho_{XY} \in [-1,1]$. If X and Y are independent, then $\rho_{XY} = 0$.

❒ **Main distributions** – Here are the main distributions to have in mind:

Type	Distribution	PDF	$\psi(\omega)$	$E[X]$	$\text{Var}(X)$
(D)	$X \sim \mathcal{B}(n,p)$ Binomial	$P(X=x) = \binom{n}{x} p^x q^{n-x}$ $x \in [\![0,n]\!]$	$(pe^{i\omega}+q)^n$	np	npq
	$X \sim \text{Po}(\mu)$ Poisson	$P(X=x) = \frac{\mu^x}{x!}e^{-\mu}$ $x \in \mathbb{N}$	$e^{\mu(e^{i\omega}-1)}$	μ	μ
(C)	$X \sim \mathcal{U}(a,b)$ Uniform	$f(x) = \frac{1}{b-a}$ $x \in [a,b]$	$\frac{e^{i\omega b}-e^{i\omega a}}{(b-a)i\omega}$	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$
	$X \sim \mathcal{N}(\mu,\sigma)$ Gaussian	$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ $x \in \mathbb{R}$	$e^{i\omega\mu - \frac{1}{2}\omega^2\sigma^2}$	μ	σ^2
	$X \sim \text{Exp}(\lambda)$ Exponential	$f(x) = \lambda e^{-\lambda x}$ $x \in \mathbb{R}_+$	$\frac{1}{1-\frac{i\omega}{\lambda}}$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

5.1.5 Parameter estimation

❒ **Random sample** – A random sample is a collection of n random variables $X_1, ..., X_n$ that are independent and identically distributed with X.

❒ **Estimator** – An estimator $\hat{\theta}$ is a function of the data that is used to infer the value of an unknown parameter θ in a statistical model.

❒ **Bias** – The bias of an estimator $\hat{\theta}$ is defined as being the difference between the expected value of the distribution of $\hat{\theta}$ and the true value, i.e.:

$$\boxed{\text{Bias}(\hat{\theta}) = E[\hat{\theta}] - \theta}$$

Remark: an estimator is said to be unbiased when we have $E[\hat{\theta}] = \theta$.

❒ **Sample mean and variance** – The sample mean and the sample variance of a random sample are used to estimate the true mean μ and the true variance σ^2 of a distribution, are noted $\overline{X}$ and s^2 respectively, and are such that:

$$\boxed{\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i} \quad \text{and} \quad \boxed{s^2 = \hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2}$$

❒ **Central Limit Theorem** – Let us have a random sample $X_1, ..., X_n$ following a given distribution with mean μ and variance σ^2, then we have:

$$\boxed{\overline{X} \underset{n\to+\infty}{\sim} \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{n}}\right)}$$

5.2 Linear Algebra and Calculus

5.2.1 General notations

❒ **Vector** – We note $x \in \mathbb{R}^n$ a vector with n entries, where $x_i \in \mathbb{R}$ is the i^{th} entry:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$$

❒ **Matrix** – We note $A \in \mathbb{R}^{m\times n}$ a matrix with m rows and n columns, where $A_{i,j} \in \mathbb{R}$ is the entry located in the i^{th} row and j^{th} column:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix} \in \mathbb{R}^{m\times n}$$

Remark: the vector x defined above can be viewed as a $n \times 1$ matrix and is more particularly called a column-vector.

❒ **Identity matrix** – The identity matrix $I \in \mathbb{R}^{n\times n}$ is a square matrix with ones in its diagonal and zero everywhere else:

$$I = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix}$$

Remark: for all matrices $A \in \mathbb{R}^{n\times n}$, we have $A \times I = I \times A = A$.

❒ **Diagonal matrix** – A diagonal matrix $D \in \mathbb{R}^{n\times n}$ is a square matrix with nonzero values in its diagonal and zero everywhere else:

$$D = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & d_n \end{pmatrix}$$

Remark: we also note D as $diag(d_1,...,d_n)$.

5.2.2 Matrix operations

❒ **Vector-vector multiplication** – There are two types of vector-vector products:

- inner product: for $x,y \in \mathbb{R}^n$, we have:

$$\boxed{x^T y = \sum_{i=1}^{n} x_i y_i \in \mathbb{R}}$$

- outer product: for $x \in \mathbb{R}^m, y \in \mathbb{R}^n$, we have:

$$\boxed{xy^T = \begin{pmatrix} x_1y_1 & \cdots & x_1y_n \\ \vdots & & \vdots \\ x_my_1 & \cdots & x_my_n \end{pmatrix} \in \mathbb{R}^{m\times n}}$$

❒ **Matrix-vector multiplication** – The product of matrix $A \in \mathbb{R}^{m\times n}$ and vector $x \in \mathbb{R}^n$ is a vector of size $\mathbb{R}^m$, such that:

$$\boxed{Ax = \begin{pmatrix} a_{r,1}^T x \\ \vdots \\ a_{r,m}^T x \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} x_i \in \mathbb{R}^m}$$

where $a_{r,i}^T$ are the vector rows and $a_{c,j}$ are the vector columns of A, and x_i are the entries of x.

❒ **Matrix-matrix multiplication** – The product of matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ is a matrix of size $\mathbb{R}^{n\times p}$, such that:

$$\boxed{AB = \begin{pmatrix} a_{r,1}^T b_{c,1} & \cdots & a_{r,1}^T b_{c,p} \\ \vdots & & \vdots \\ a_{r,m}^T b_{c,1} & \cdots & a_{r,m}^T b_{c,p} \end{pmatrix} = \sum_{i=1}^{n} a_{c,i} b_{r,i}^T \in \mathbb{R}^{n\times p}}$$

where $a_{r,i}^T, b_{r,i}^T$ are the vector rows and $a_{c,j}, b_{c,j}$ are the vector columns of A and B respectively.

❒ **Transpose** – The transpose of a matrix $A \in \mathbb{R}^{m\times n}$, noted A^T, is such that its entries are flipped:

$$\boxed{\forall i,j, \quad A_{i,j}^T = A_{j,i}}$$

Remark: for matrices A,B, we have $(AB)^T = B^T A^T$.

❒ **Inverse** – The inverse of an invertible square matrix A is noted A^{-1} and is the only matrix such that:

$$\boxed{AA^{-1} = A^{-1}A = I}$$

Remark: not all square matrices are invertible. Also, for matrices A,B, we have $(AB)^{-1} = B^{-1}A^{-1}$

❒ **Trace** – The trace of a square matrix A, noted $\mathrm{tr}(A)$, is the sum of its diagonal entries:

$$\boxed{\mathrm{tr}(A) = \sum_{i=1}^{n} A_{i,i}}$$

Remark: for matrices A,B, we have $tr(A^T) = tr(A)$ and $tr(AB) = tr(BA)$

❒ **Determinant** – The determinant of a square matrix $A \in \mathbb{R}^{n\times n}$, noted $|A|$ or $\det(A)$ is expressed recursively in terms of $A_{\backslash i,\backslash j}$, which is the matrix A without its i^{th} row and j^{th} column, as follows:

$$\boxed{\det(A) = |A| = \sum_{j=1}^{n} (-1)^{i+j} A_{i,j} |A_{\backslash i,\backslash j}|}$$

Remark: A is invertible if and only if $|A| \neq 0$. Also, $|AB| = |A||B|$ and $|A^T| = |A|$.

5.2.3 Matrix properties

❒ **Symmetric decomposition** – A given matrix A can be expressed in terms of its symmetric and antisymmetric parts as follows:

$$\boxed{A = \underbrace{\frac{A + A^T}{2}}_{\text{Symmetric}} + \underbrace{\frac{A - A^T}{2}}_{\text{Antisymmetric}}}$$

❒ **Norm** – A norm is a function $N : V \longrightarrow [0, +\infty[$ where V is a vector space, and such that for all $x,y \in V$, we have:

- $N(x + y) \leqslant N(x) + N(y)$
- $N(ax) = |a|N(x)$ for a scalar
- if $N(x) = 0$, then $x = 0$

For $x \in V$, the most commonly used norms are summed up in the table below:

Norm	Notation	Definition	Use case
Manhattan, L^1	$\|\|x\|\|_1$	$\sum_{i=1}^{n} \|x_i\|$	LASSO regularization
Euclidean, L^2	$\|\|x\|\|_2$	$\sqrt{\sum_{i=1}^{n} x_i^2}$	Ridge regularization
p-norm, L^p	$\|\|x\|\|_p$	$\left(\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}}$	Hölder inequality
Infinity, L^∞	$\|\|x\|\|_\infty$	$\max_i \|x_i\|$	Uniform convergence

❒ **Linearly dependence** – A set of vectors is said to be linearly dependent if one of the vectors in the set can be defined as a linear combination of the others.
Remark: if no vector can be written this way, then the vectors are said to be linearly independent.

❒ **Matrix rank** – The rank of a given matrix A is noted $\text{rank}(A)$ and is the dimension of the vector space generated by its columns. This is equivalent to the maximum number of linearly independent columns of A.

❒ **Positive semi-definite matrix** – A matrix $A \in \mathbb{R}^{n\times n}$ is positive semi-definite (PSD) and is noted $A \succeq 0$ if we have:

$$\boxed{A = A^T} \quad \text{and} \quad \boxed{\forall x \in \mathbb{R}^n, \quad x^T A x \geqslant 0}$$

Remark: similarly, a matrix A is said to be positive definite, and is noted $A \succ 0$, if it is a PSD matrix which satisfies for all non-zero vector x, $x^T Ax > 0$.

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n\times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$\boxed{Az = \lambda z}$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n\times n}$. By noting $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, we have:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

❒ **Singular-value decomposition** – For a given matrix A of dimensions $m \times n$, the singular-value decomposition (SVD) is a factorization technique that guarantees the existence of U $m\times m$ unitary, Σ $m \times n$ diagonal and V $n \times n$ unitary matrices, such that:

$$\boxed{A = U\Sigma V^T}$$

5.2.4 Matrix calculus

❒ **Gradient** – Let $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ be a function and $A \in \mathbb{R}^{m\times n}$ be a matrix. The gradient of f with respect to A is a $m \times n$ matrix, noted $\nabla_A f(A)$, such that:

$$\boxed{\Big(\nabla_A f(A)\Big)_{i,j} = \frac{\partial f(A)}{\partial A_{i,j}}}$$

Remark: the gradient of f is only defined when f is a function that returns a scalar.

❒ **Hessian** – Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function and $x \in \mathbb{R}^n$ be a vector. The hessian of f with respect to x is a $n \times n$ symmetric matrix, noted $\nabla_x^2 f(x)$, such that:

$$\boxed{\Big(\nabla_x^2 f(x)\Big)_{i,j} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}}$$

Remark: the hessian of f is only defined when f is a function that returns a scalar.

❒ **Gradient operations** – For matrices A,B,C, the following gradient properties are worth having in mind:

$$\boxed{\nabla_A \text{tr}(AB) = B^T} \qquad \boxed{\nabla_{A^T} f(A) = (\nabla_A f(A))^T}$$

$$\boxed{\nabla_A \text{tr}(ABA^TC) = CAB + C^TAB^T} \qquad \boxed{\nabla_A |A| = |A|(A^{-1})^T}$$

Super VIP Cheatsheet: Deep Learning

Afshine Amidi and Shervine Amidi

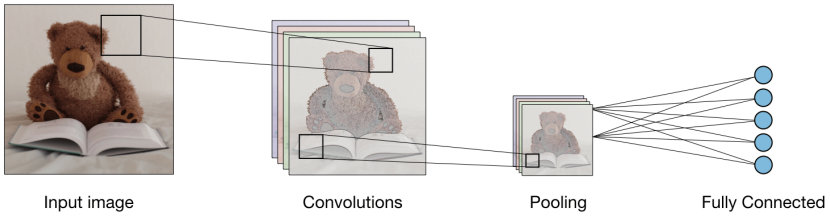
November 25, 2018

Contents

1 Convolutional Neural Networks

1.1 Overview

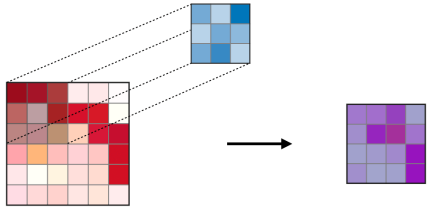
❒ **Architecture of a traditional CNN** – Convolutional neural networks, also known as CNNs, are a specific type of neural networks that are generally composed of the following layers:



The convolution layer and the pooling layer can be fine-tuned with respect to hyperparameters that are described in the next sections.

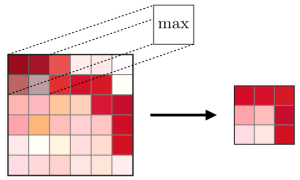
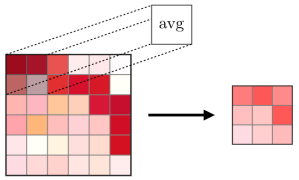
1.2 Types of layer

❒ **Convolutional layer (CONV)** – The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input I with respect to its dimensions. Its hyperparameters include the filter size F and stride S. The resulting output O is called *feature map* or *activation map*.

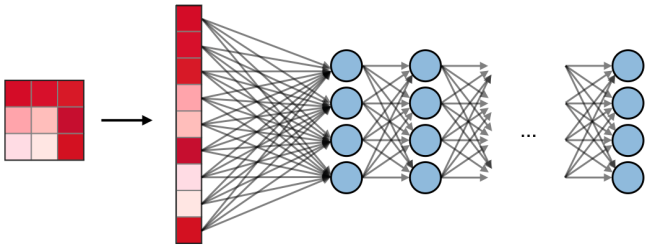


Remark: the convolution step can be generalized to the 1D and 3D cases as well.

❒ **Pooling (POOL)** – The pooling layer (POOL) is a downsampling operation, typically applied after a convolution layer, which does some spatial invariance. In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.

	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration		
Comments	- Preserves detected features - Most commonly used	- Downsamples feature map - Used in LeNet

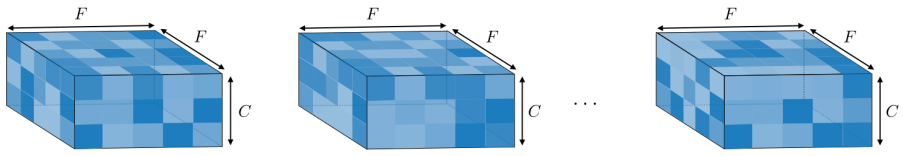
❒ **Fully Connected (FC)** – The fully connected layer (FC) operates on a flattened input where each input is connected to all neurons. If present, FC layers are usually found towards the end of CNN architectures and can be used to optimize objectives such as class scores.



1.3 Filter hyperparameters

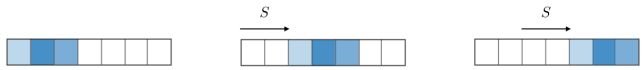
The convolution layer contains filters for which it is important to know the meaning behind its hyperparameters.

❒ **Dimensions of a filter** – A filter of size $F \times F$ applied to an input containing C channels is a $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called activation map) of size $O \times O \times 1$.

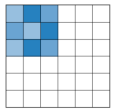
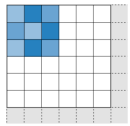
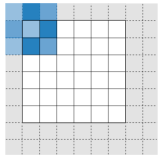


Remark: the application of K filters of size $F \times F$ results in an output feature map of size $O \times O \times K$.

❒ **Stride** – For a convolutional or a pooling operation, the stride S denotes the number of pixels by which the window moves after each operation.



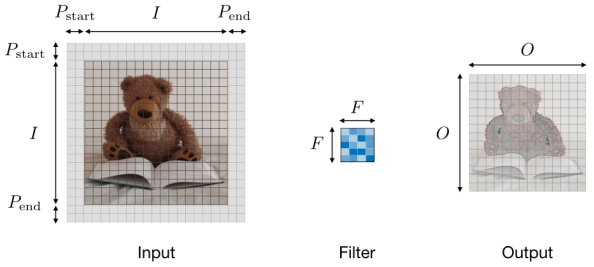
❒ **Zero-padding** – Zero-padding denotes the process of adding P zeroes to each side of the boundaries of the input. This value can either be manually specified or automatically set through one of the three modes detailed below:

	Valid	Same	Full
Value	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in [\![0, F-1]\!]$ $P_{\text{end}} = F - 1$
Illustration			
Purpose	- No padding - Drops last convolution if dimensions do not match	- Padding such that feature map size has size $\lceil \frac{I}{S} \rceil$ - Output size is mathematically convenient - Also called 'half' padding	- Maximum padding such that end convolutions are applied on the limits of the input - Filter 'sees' the input end-to-end

1.4 Tuning hyperparameters

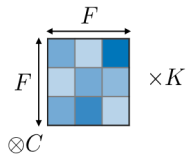
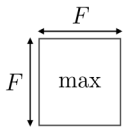
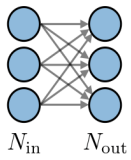
❒ **Parameter compatibility in convolution layer** – By noting I the length of the input volume size, F the length of the filter, P the amount of zero padding, S the stride, then the output size O of the feature map along that dimension is given by:

$$\boxed{O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1}$$



Remark: often times, $P_{start} = P_{end} \triangleq P$, in which case we can replace $P_{start} + P_{end}$ by $2P$ in the formula above.

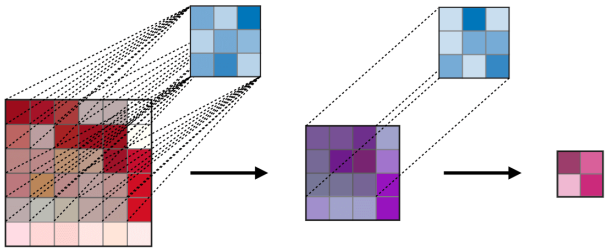
❒ **Understanding the complexity of the model** – In order to assess the complexity of a model, it is often useful to determine the number of parameters that its architecture will have. In a given layer of a convolutional neural network, it is done as follows:

	CONV	POOL	FC
Illustration			
Input size	$I \times I \times C$	$I \times I \times C$	N_{in}
Output size	$O \times O \times K$	$O \times O \times C$	N_{out}
Number of parameters	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarks	- One bias parameter per filter - In most cases, $S < F$ - A common choice for K is $2C$	- Pooling operation done channel-wise - In most cases, $S = F$	- Input is flattened - One bias parameter per neuron - The number of FC neurons is free of structural constraints

❒ **Receptive field** – The receptive field at layer k is the area denoted $R_k \times R_k$ of the input that each pixel of the k-th activation map can 'see'. By calling F_j the filter size of layer j and S_i the stride value of layer i and with the convention $S_0 = 1$, the receptive field at layer k can be computed with the formula:

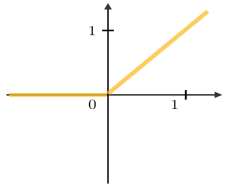
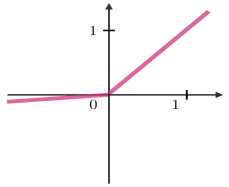
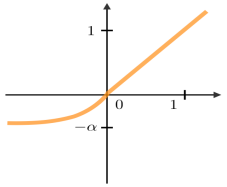
$$R_k = 1 + \sum_{j=1}^{k} (F_j - 1) \prod_{i=0}^{j-1} S_i$$

In the example below, we have $F_1 = F_2 = 3$ and $S_1 = S_2 = 1$, which gives $R_2 = 1+2 \cdot 1+2 \cdot 1 = 5$.



1.5 Commonly used activation functions

❒ **Rectified Linear Unit** – The rectified linear unit layer (ReLU) is an activation function g that is used on all elements of the volume. It aims at introducing non-linearities to the network. Its variants are summarized in the table below:

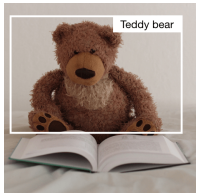
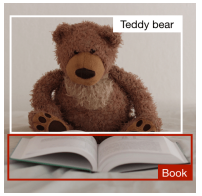
ReLU	Leaky ReLU	ELU
$g(z) = \max(0,z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
Non-linearity complexities biologically interpretable	Addresses dying ReLU issue for negative values	Differentiable everywhere

❒ **Softmax** – The softmax step can be seen as a generalized logistic function that takes as input a vector of scores $x \in \mathbb{R}^n$ and outputs a vector of output probability $p \in \mathbb{R}^n$ through a softmax function at the end of the architecture. It is defined as follows:

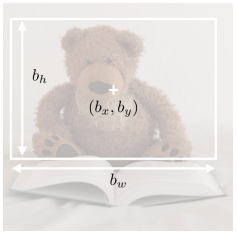
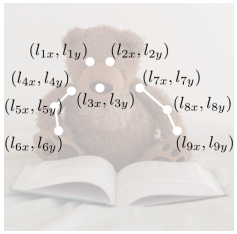
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{where} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

1.6 Object detection

❒ **Types of models** – There are 3 main types of object recognition algorithms, for which the nature of what is predicted is different. They are described in the table below:

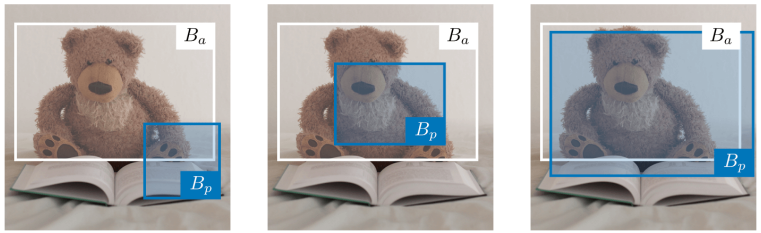
Image classification	Classification w. localization	Detection
		
- Classifies a picture - Predicts probability of object	- Detects object in a picture - Predicts probability of object and where it is located	- Detects up to several objects in a picture - Predicts probabilities of objects and where they are located
Traditional CNN	Simplified YOLO, R-CNN	YOLO, R-CNN

❒ **Detection** – In the context of object detection, different methods are used depending on whether we just want to locate the object or detect a more complex shape in the image. The two main ones are summed up in the table below:

Bounding box detection	Landmark detection
Detects the part of the image where the object is located	- Detects a shape or characteristics of an object (e.g. eyes) - More granular
	
Box of center (b_x,b_y), height b_h and width b_w	Reference points (l_{1x},l_{1y}), ...,(l_{nx},l_{ny})

❒ Intersection over Union – Intersection over Union, also known as IoU, is a function that quantifies how correctly positioned a predicted bounding box B_p is over the actual bounding box B_a. It is defined as:

$$\boxed{\text{IoU}(B_p,B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}}$$



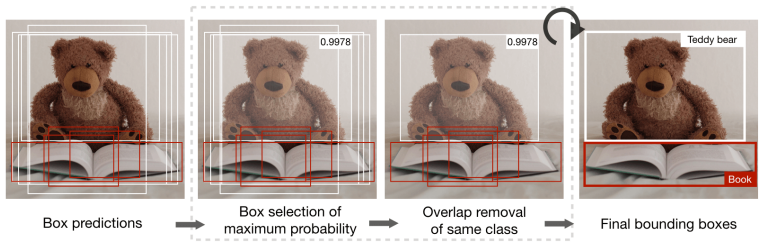
$\text{IoU}(B_p, B_a) = 0.1$ $\text{IoU}(B_p, B_a) = 0.5$ $\text{IoU}(B_p, B_a) = 0.9$

Remark: we always have $IoU \in [0,1]$. By convention, a predicted bounding box B_p is considered as being reasonably good if $IoU(B_p,B_a) \geqslant 0.5$.

❒ Anchor boxes – Anchor boxing is a technique used to predict overlapping bounding boxes. In practice, the network is allowed to predict more than one box simultaneously, where each box prediction is constrained to have a given set of geometrical properties. For instance, the first prediction can potentially be a rectangular box of a given form, while the second will be another rectangular box of a different geometrical form.

❒ Non-max suppression – The non-max suppression technique aims at removing duplicate overlapping bounding boxes of a same object by selecting the most representative ones. After having removed all boxes having a probability prediction lower than 0.6, the following steps are repeated while there are boxes remaining:

- Step 1: Pick the box with the largest prediction probability.
- Step 2: Discard any box having an IoU $\geqslant 0.5$ with the previous box.



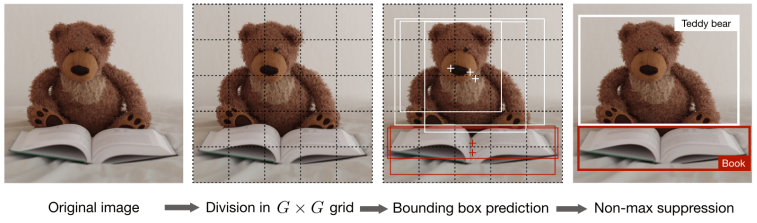
❒ YOLO – You Only Look Once (YOLO) is an object detection algorithm that performs the following steps:

- Step 1: Divide the input image into a $G \times G$ grid.
- Step 2: For each grid cell, run a CNN that predicts y of the following form:

$$\boxed{y = \Big[\underbrace{p_c,b_x,b_y,b_h,b_w,c_1,c_2,...,c_p}_{\text{repeated } k \text{ times}},...\Big]^T \in \mathbb{R}^{G\times G\times k\times(5+p)}}$$

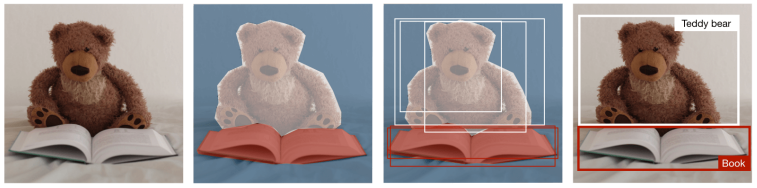
where p_c is the probability of detecting an object, b_x,b_y,b_h,b_w are the properties of the detected bouding box, c_1,...,c_p is a one-hot representation of which of the p classes were detected, and k is the number of anchor boxes.

- Step 3: Run the non-max suppression algorithm to remove any potential duplicate overlapping bounding boxes.



Remark: when $p_c = 0$, then the network does not detect any object. In that case, the corresponding predictions $b_x, ..., c_p$ have to be ignored.

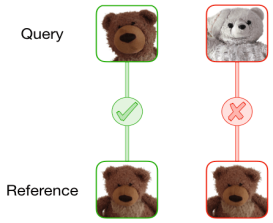
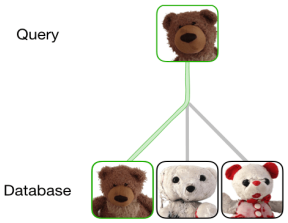
❒ R-CNN – Region with Convolutional Neural Networks (R-CNN) is an object detection algorithm that first segments the image to find potential relevant bounding boxes and then run the detection algorithm to find most probable objects in those bounding boxes.



Remark: although the original algorithm is computationally expensive and slow, newer architectures enabled the algorithm to run faster, such as Fast R-CNN and Faster R-CNN.

1.6.1 Face verification and recognition

❒ **Types of models** – Two main types of model are summed up in table below:

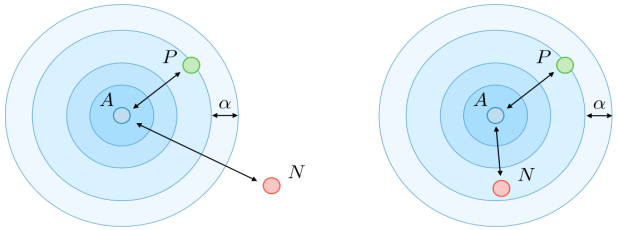
Face verification	Face recognition
- Is this the correct person? - One-to-one lookup	- Is this one of the K persons in the database? - One-to-many lookup
	

❒ **One Shot Learning** – One Shot Learning is a face verification algorithm that uses a limited training set to learn a similarity function that quantifies how different two given images are. The similarity function applied to two images is often noted $d(\text{image 1}, \text{image 2})$.

❒ **Siamese Network** – Siamese Networks aim at learning how to encode images to then quantify how different two images are. For a given input image $x^{(i)}$, the encoded output is often noted as $f(x^{(i)})$.

❒ **Triplet loss** – The triplet loss ℓ is a loss function computed on the embedding representation of a triplet of images A (anchor), P (positive) and N (negative). The anchor and the positive example belong to a same class, while the negative example to another one. By calling $\alpha \in \mathbb{R}^+$ the margin parameter, this loss is defined as follows:

$$\boxed{\ell(A,P,N) = \max\left(d(A,P) - d(A,N) + \alpha, 0\right)}$$



$\ell(A,P,N) = 0$ $\quad$ $\ell(A,P,N) > 0$

1.6.2 Neural style transfer

❒ **Motivation** – The goal of neural style transfer is to generate an image G based on a given content C and a given style S.

Content C $\quad$ Style S $\quad$ Generated image G

❒ **Activation** – In a given layer l, the activation is noted $a^{[l]}$ and is of dimensions $n_H \times n_w \times n_c$

❒ **Content cost function** – The content cost function $J_{\text{content}}(C,G)$ is used to determine how the generated image G differs from the original content image C. It is defined as follows:

$$\boxed{J_{\text{content}}(C,G) = \frac{1}{2}||a^{[l](C)} - a^{[l](G)}||^2}$$

❒ **Style matrix** – The style matrix $G^{[l]}$ of a given layer l is a Gram matrix where each of its elements $G^{[l]}_{kk'}$ quantifies how correlated the channels k and k' are. It is defined with respect to activations $a^{[l]}$ as follows:

$$\boxed{G^{[l]}_{kk'} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_w^{[l]}} a^{[l]}_{ijk} a^{[l]}_{ijk'}}$$

Remark: the style matrix for the style image and the generated image are noted $G^{[l](S)}$ and $G^{[l](G)}$ respectively.

❒ **Style cost function** – The style cost function $J_{\text{style}}(S,G)$ is used to determine how the generated image G differs from the style S. It is defined as follows:

$$\boxed{J^{[l]}_{\text{style}}(S,G) = \frac{1}{(2n_H n_w n_c)^2}||G^{[l](S)} - G^{[l](G)}||^2_F = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left(G^{[l](S)}_{kk'} - G^{[l](G)}_{kk'}\right)^2}$$

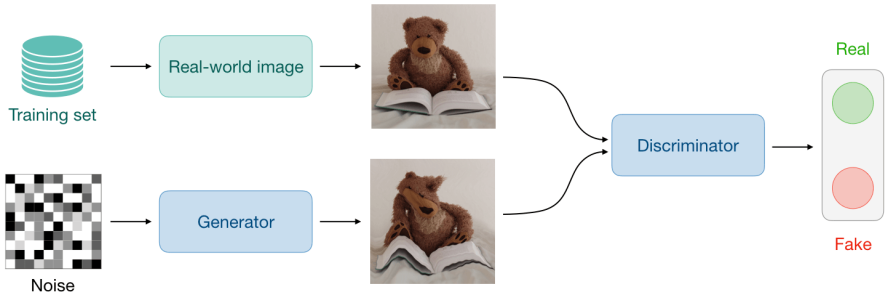
❒ **Overall cost function** – The overall cost function is defined as being a combination of the content and style cost functions, weighted by parameters α,β, as follows:

$$\boxed{J(G) = \alpha J_{\text{content}}(C,G) + \beta J_{\text{style}}(S,G)}$$

Remark: a higher value of α will make the model care more about the content while a higher value of β will make it care more about the style.

1.6.3 Architectures using computational tricks

❒ **Generative Adversarial Network** – Generative adversarial networks, also known as GANs, are composed of a generative and a discriminative model, where the generative model aims at generating the most truthful output that will be fed into the discriminative which aims at differentiating the generated and true image.



Remark: use cases using variants of GANs include text to image, music generation and synthesis.

❒ ResNet – The Residual Network architecture (also called ResNet) uses residual blocks with a high number of layers meant to decrease the training error. The residual block has the following characterizing equation:

$$\boxed{a^{[l+2]} = g(a^{[l]} + z^{[l+2]})}$$

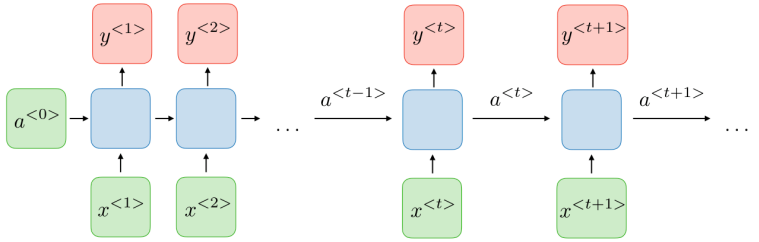
❒ Inception Network – This architecture uses inception modules and aims at giving a try at different convolutions in order to increase its performance. In particular, it uses the 1×1 convolution trick to lower the burden of computation.

⋆ ⋆ ⋆

2 Recurrent Neural Networks

2.1 Overview

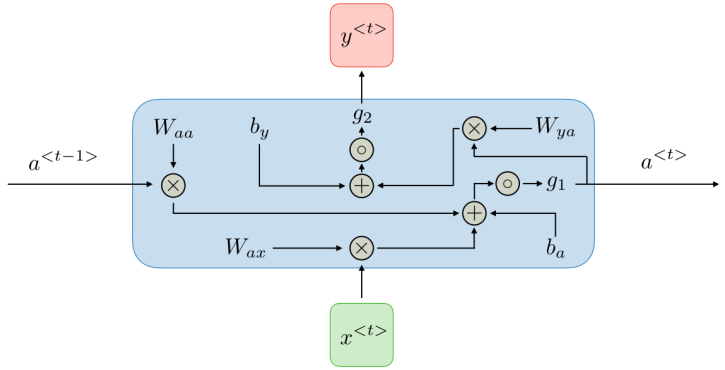
❒ Architecture of a traditional RNN – Recurrent neural networks, also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are typically as follows:



For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as follows:

$$\boxed{a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)} \quad \text{and} \quad \boxed{y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)}$$

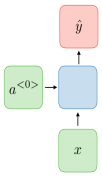
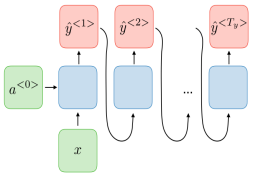
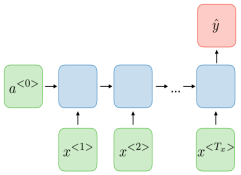
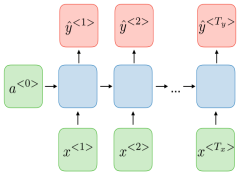
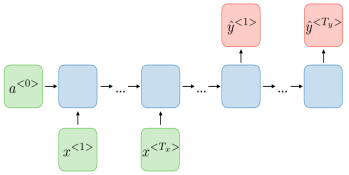
where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions



The pros and cons of a typical RNN architecture are summed up in the table below:

Advantages	Drawbacks
- Possibility of processing input of any length - Model size not increasing with size of input - Computation takes into account historical information - Weights are shared across time	- Computation being slow - Difficulty of accessing information from a long time ago - Cannot consider any future input for the current state

❒ Applications of RNNs – RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

❒ **Loss function** – In the case of a recurrent neural network, the loss function $\mathcal{L}$ of all time steps is defined based on the loss at every time step as follows:

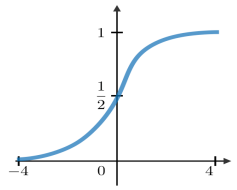
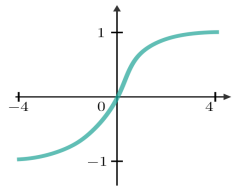
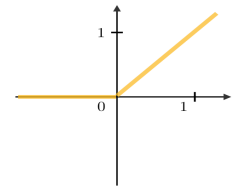
$$\boxed{\mathcal{L}(\widehat{y},y) = \sum_{t=1}^{T_y} \mathcal{L}(\widehat{y}^{<t>},y^{<t>})}$$

❒ **Backpropagation through time** – Backpropagation is done at each point in time. At timestep T, the derivative of the loss $\mathcal{L}$ with respect to weight matrix W is expressed as follows:

$$\boxed{\frac{\partial \mathcal{L}^{(T)}}{\partial W} = \sum_{t=1}^{T} \left.\frac{\partial \mathcal{L}^{(T)}}{\partial W}\right|_{(t)}}$$

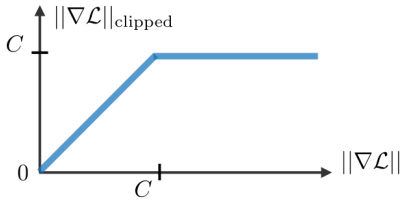
2.2 Handling long term dependencies

❒ **Commonly used activation functions** – The most common activation functions used in RNN modules are described below:

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0,z)$
		

❒ **Vanishing/exploding gradient** – The vanishing and exploding gradient phenomena are often encountered in the context of RNNs. The reason why they happen is that it is difficult to capture long term dependencies because of multiplicative gradient that can be exponentially decreasing/increasing with respect to the number of layers.

❒ **Gradient clipping** – It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



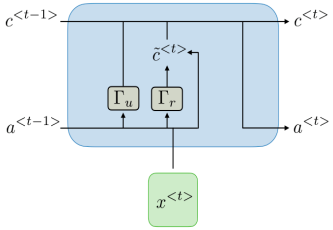
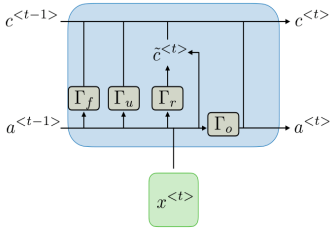
❒ **Types of gates** – In order to remedy the vanishing gradient problem, specific gates are used in some types of RNNs and usually have a well-defined purpose. They are usually noted Γ and are equal to:

$$\boxed{\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)}$$

where W, U, b are coefficients specific to the gate and σ is the sigmoid function. The main ones are summed up in the table below:

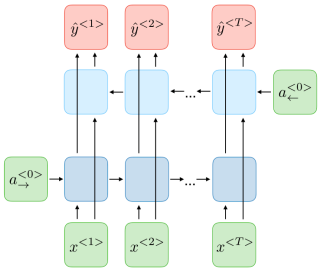
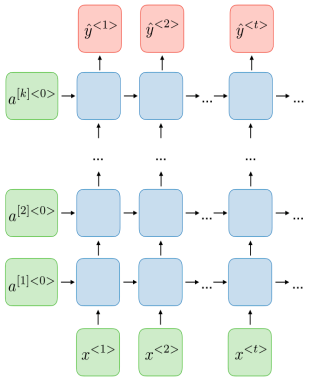
Type of gate	Role	Used in
Update gate Γ_u	How much past should matter now?	GRU, LSTM
Relevance gate Γ_r	Drop previous information?	GRU, LSTM
Forget gate Γ_f	Erase a cell or not?	LSTM
Output gate Γ_o	How much to reveal of a cell?	LSTM

❒ **GRU/LSTM** – Gated Recurrent Unit (GRU) and Long Short-Term Memory units (LSTM) deal with the vanishing gradient problem encountered by traditional RNNs, with LSTM being a generalization of GRU. Below is a table summing up the characterizing equations of each architecture:

	Gated Recurrent Unit (GRU)	**Long Short-Term Memory** (LSTM)
$\tilde{c}^{<t>}$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$	$\tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
$c^{<t>}$	$\Gamma_u \star \tilde{c}^{<t>} + (1-\Gamma_u) \star c^{<t-1>}$	$\Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$
$a^{<t>}$	$c^{<t>}$	$\Gamma_o \star c^{<t>}$
Dependencies		

Remark: the sign $\star$ denotes the element-wise multiplication between two vectors.

❒ **Variants of RNNs** – The table below sums up the other commonly used RNN architectures:

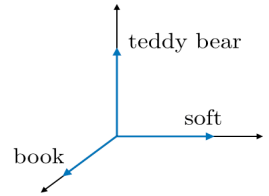
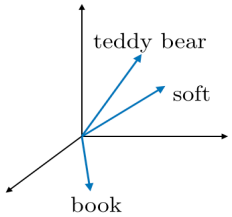
Bidirectional (BRNN)	**Deep** (DRNN)
	

2.3 Learning word representation

In this section, we note V the vocabulary and $|V|$ its size.

2.3.1 Motivation and notations

❒ **Representation techniques** – The two main ways of representing words are summed up in the table below:

1-hot representation	**Word embedding**
	
- Noted o_w - Naive approach, no similarity information	- Noted e_w - Takes into account words similarity

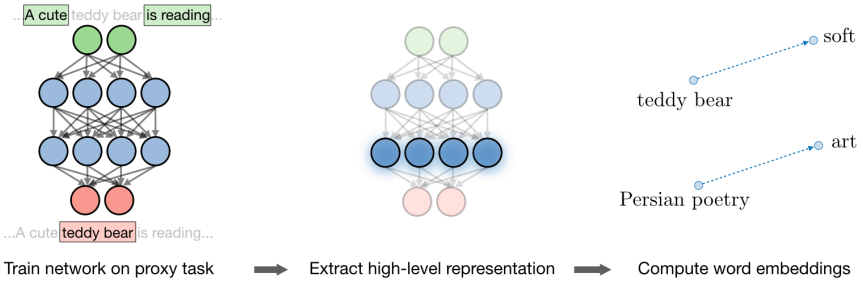
❒ **Embedding matrix** – For a given word w, the embedding matrix E is a matrix that maps its 1-hot representation o_w to its embedding e_w as follows:

$$\boxed{e_w = Eo_w}$$

Remark: learning the embedding matrix can be done using target/context likelihood models.

2.3.2 Word embeddings

❒ **Word2vec** – Word2vec is a framework aimed at learning word embeddings by estimating the likelihood that a given word is surrounded by other words. Popular models include skip-gram, negative sampling and CBOW.



❒ **Skip-gram** – The skip-gram word2vec model is a supervised learning task that learns word embeddings by assessing the likelihood of any given target word t happening with a context word c. By noting θ_t a parameter associated with t, the probability $P(t|c)$ is given by:

$$P(t|c) = \frac{\exp(\theta_t^T e_c)}{\sum_{j=1}^{|V|} \exp(\theta_j^T e_c)}$$

Remark: summing over the whole vocabulary in the denominator of the softmax part makes this model computationally expensive. CBOW is another word2vec model using the surrounding words to predict a given word.

❒ **Negative sampling** – It is a set of binary classifiers using logistic regressions that aim at assessing how a given context and a given target words are likely to appear simultaneously, with the models being trained on sets of k negative examples and 1 positive example. Given a context word c and a target word t, the prediction is expressed by:

$$P(y = 1|c,t) = \sigma(\theta_t^T e_c)$$

Remark: this method is less computationally expensive than the skip-gram model.

❒ **GloVe** – The GloVe model, short for global vectors for word representation, is a word embedding technique that uses a co-occurence matrix X where each $X_{i,j}$ denotes the number of times that a target i occurred with a context j. Its cost function J is as follows:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{|V|} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log(X_{ij}))^2$$

here f is a weighting function such that $X_{i,j} = 0 \Longrightarrow f(X_{i,j}) = 0$.

Given the symmetry that e and θ play in this model, the final word embedding $e_w^{(\text{final})}$ is given by:

$$e_w^{(\text{final})} = \frac{e_w + \theta_w}{2}$$

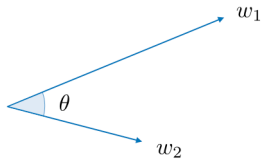
Remark: the individual components of the learned word embeddings are not necessarily interpretable.

2.4 Comparing words

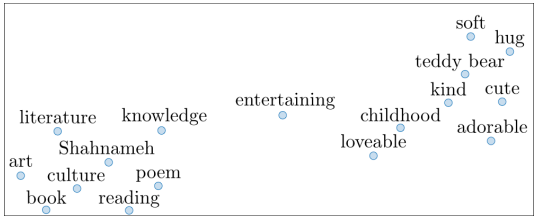
❒ **Cosine similarity** – The cosine similarity between words w_1 and w_2 is expressed as follows:

$$\text{similarity} = \frac{w_1 \cdot w_2}{||w_1||\ ||w_2||} = \cos(\theta)$$

Remark: θ is the angle between words w_1 and w_2.



❒ ***t*-SNE** – t-SNE (t-distributed Stochastic Neighbor Embedding) is a technique aimed at reducing high-dimensional embeddings into a lower dimensional space. In practice, it is commonly used to visualize word vectors in the 2D space.



2.5 Language model

❒ **Overview** – A language model aims at estimating the probability of a sentence $P(y)$.

❒ **n-gram model** – This model is a naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

❒ **Perplexity** – Language models are commonly assessed using the perplexity metric, also known as PP, which can be interpreted as the inverse probability of the dataset normalized by the number of words T. The perplexity is such that the lower, the better and is defined as follows:

$$\text{PP} = \prod_{t=1}^{T} \left(\frac{1}{\sum_{j=1}^{|V|} y_j^{(t)} \cdot \hat{y}_j^{(t)}} \right)^{\frac{1}{T}}$$

Remark: PP is commonly used in t-SNE.

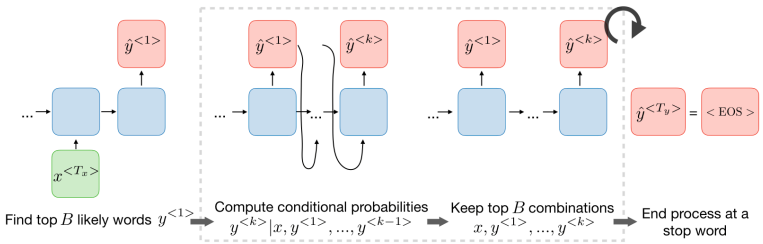
2.6 Machine translation

❒ **Overview** – A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model. The goal is to find a sentence y such that:

$$y = \underset{y^{<1>},\ldots,y^{<T_y>}}{\arg\max}\ P(y^{<1>},\ldots,y^{<T_y>}|x)$$

❒ **Beam search** – It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x.

- Step 1: Find top B likely words $y^{<1>}$
- Step 2: Compute conditional probabilities $y^{<k>}|x,y^{<1>},\ldots,y^{<k-1>}$
- Step 3: Keep top B combinations $x,y^{<1>},\ldots,y^{<k>}$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

❒ **Beam width** – The beam width B is a parameter for beam search. Large values of B yield to better result but with slower performance and increased memory. Small values of B lead to worse results but is less computationally intensive. A standard value for B is around 10.

❒ **Length normalization** – In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective } = \frac{1}{T_y^\alpha}\sum_{t=1}^{T_y}\log\left[p(y^{<t>}|x,y^{<1>},...,y^{<t-1>})\right]$$

Remark: the parameter α can be seen as a softener, and its value is usually between 0.5 and 1.

❒ **Error analysis** – When obtaining a predicted translation $\widehat{y}$ that is bad, one can wonder why we did not get a good translation y^* by performing the following error analysis:

Case	$P(y^*\|x) > P(\widehat{y}\|x)$	$P(y^*\|x) \leqslant P(\widehat{y}\|x)$
Root cause	Beam search faulty	RNN faulty
Remedies	Increase beam width	- Try different architecture - Regularize - Get more data

❒ **Bleu score** – The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp\left(\frac{1}{n}\sum_{k=1}^{n}p_k\right)$$

where p_n is the bleu score on n-gram only defined as follows:

$$p_n = \frac{\sum_{\text{n-gram}\in\widehat{y}}\text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram}\in\widehat{y}}\text{count}(\text{n-gram})}$$

Remark: a brevity penalty may be applied to short predicted translations to prevent an artificially inflated bleu score.

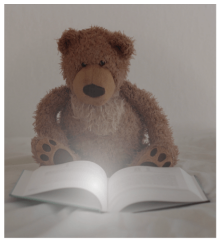
2.7 Attention

❒ **Attention model** – This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting $\alpha^{<t,t'>}$ the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ and $c^{<t>}$ the context at time t, we have:

$$c^{<t>} = \sum_{t'}\alpha^{<t,t'>}a^{<t'>} \quad \text{with} \quad \sum_{t'}\alpha^{<t,t'>} = 1$$

Remark: the attention scores are commonly used in image captioning and machine translation.

A cute teddy bear is reading Persian literature



A cute teddy bear is reading Persian literature

❒ **Attention weight** – The amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ is given by $\alpha^{<t,t'>}$ computed as follows:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x}\exp(e^{<t,t''>})}$$

Remark: computation complexity is quadratic with respect to T_x.

⋆ ⋆ ⋆

3 Deep Learning Tips and Tricks

3.1 Data processing

❒ **Data augmentation** – Deep learning models usually need a lot of data to be properly trained. It is often useful to get more data from the existing ones using data augmentation techniques. The main ones are summed up in the table below. More precisely, given the following input image, here are the techniques that we can apply:

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

❒ **Batch normalization** – It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

3.2 Training a neural network

3.2.1 Definitions

❒ **Epoch** – In the context of training a model, epoch is a term used to refer to one iteration where the model sees the whole training set to update its weights.

❒ **Mini-batch gradient descent** – During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues. Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

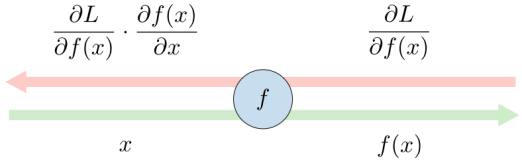
❒ **Loss function** – In order to quantify how a given model performs, the loss function L is usually used to evaluate to what extent the actual outputs y are correctly predicted by the model outputs z.

❒ **Cross-entropy loss** – In the context of binary classification in neural networks, the cross-entropy loss $L(z,y)$ is commonly used and is defined as follows:

$$\boxed{L(z,y) = -\Big[y\log(z) + (1-y)\log(1-z)\Big]}$$

3.2.2 Finding optimal weights

❒ **Backpropagation** – Backpropagation is a method to update the weights in the neural network by taking into account the actual output and the desired output. The derivative with respect to each weight w is computed using the chain rule.

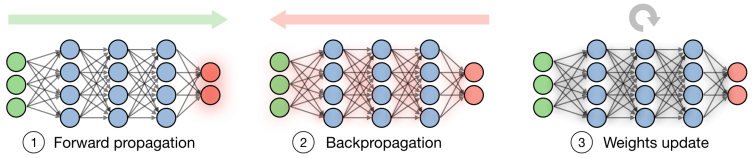


Using this method, each weight is updated with the rule:

$$\boxed{w \longleftarrow w - \alpha \frac{\partial L(z,y)}{\partial w}}$$

❒ **Updating weights** – In a neural network, weights are updated as follows:

- Step 1: Take a batch of training data and perform forward propagation to compute the loss.
- Step 2: Backpropagate the loss to get the gradient of the loss with respect to each weight.
- Step 3: Use the gradients to update the weights of the network.

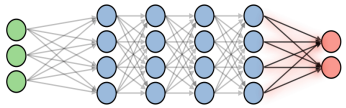
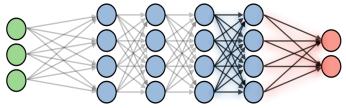
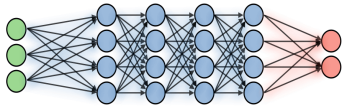


3.3 Parameter tuning

3.3.1 Weights initialization

❒ **Xavier initialization** – Instead of initializing the weights in a purely random manner, Xavier initialization enables to have initial weights that take into account characteristics that are unique to the architecture.

❒ **Transfer learning** – Training a deep learning model requires a lot of data and more importantly a lot of time. It is often useful to take advantage of pre-trained weights on huge datasets that took days/weeks to train, and leverage it towards our use case. Depending on how much data we have at hand, here are the different ways to leverage this:

Training size	Illustration	Explanation
Small		Freezes all layers, trains weights on softmax
Medium		Freezes most layers, trains weights on last layers and softmax
Large		Trains weights on layers and softmax by initializing weights on pre-trained ones

3.3.2 Optimizing convergence

❒ **Learning rate** – The learning rate, often noted α or sometimes η, indicates at which pace the weights get updated. It can be fixed or adaptively changed. The current most popular method is called Adam, which is a method that adapts the learning rate.

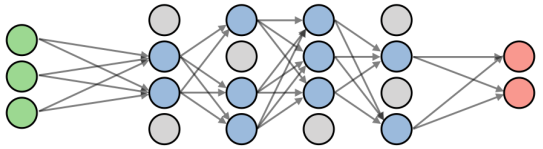
❒ **Adaptive learning rates** – Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution. While Adam optimizer is the most commonly used technique, others can also be useful. They are summed up in the table below:

Method	Explanation	Update of w	Update of b
Momentum	- Dampens oscillations - Improvement to SGD - 2 parameters to tune	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- Root Mean Square propagation - Speeds up learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \longleftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - Most popular method - 4 parameters to tune	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \longleftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

Remark: other methods include Adadelta, Adagrad and SGD.

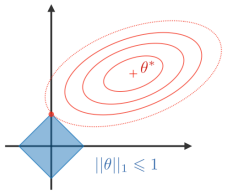
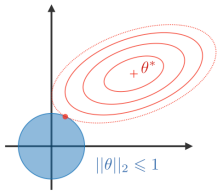
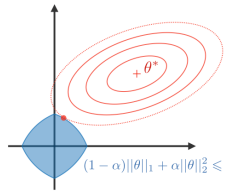
3.4 Regularization

❒ **Dropout** – Dropout is a technique used in neural networks to prevent overfitting the training data by dropping out neurons with probability $p > 0$. It forces the model to avoid relying too much on particular sets of features.

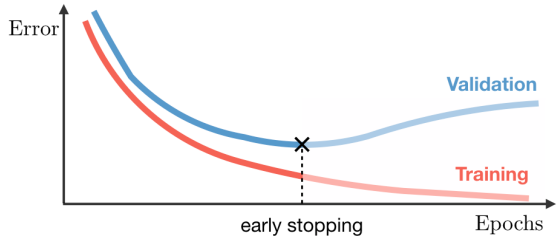


Remark: most deep learning frameworks parametrize dropout through the 'keep' parameter $1-p$.

❒ **Weight regularization** – In order to make sure that the weights are not too large and that the model is not overfitting the training set, regularization techniques are usually performed on the model weights. The main ones are summed up in the table below:

LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$... + \lambda \|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda \|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda \left[(1-\alpha) \|\|\theta\|\|_1 + \alpha \|\|\theta\|\|_2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0,1]$

❒ **Early stopping** – This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



3.5 Good practices

❒ **Overfitting small batch** – When debugging a model, it is often useful to make quick tests to see if there is any major issue with the architecture of the model itself. In particular, in order to make sure that the model can be properly trained, a mini-batch is passed inside the network to see if it can overfit on it. If it cannot, it means that the model is either too complex or not complex enough to even overfit on a small batch, let alone a normal-sized training set.

❒ **Gradient checking** – Gradient checking is a method used during the implementation of the backward pass of a neural network. It compares the value of the analytical gradient to the numerical gradient at given points and plays the role of a sanity-check for correctness.

	Numerical gradient	**Analytical gradient**
Formula	$\frac{df}{dx}(x) \approx \frac{f(x+h)-f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
Comments	- Expensive; loss has to be computed two times per dimension - Used to verify correctness of analytical implementation -Trade-off in choosing h not too small (numerical instability) nor too large (poor gradient approx.)	- 'Exact' result - Direct computation - Used in the final implementation

⋆ ⋆ ⋆

Super VIP Cheatsheet: Artificial Intelligence

Afshine Amidi and Shervine Amidi

September 8, 2019

Contents

1 Reflex-based models

1.1 Linear predictors

In this section, we will go through reflex-based models that can improve with experience, by going through samples that have input-output pairs.

❒ **Feature vector** – The feature vector of an input x is noted $\phi(x)$ and is such that:

$$\boxed{\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_d(x) \end{bmatrix} \in \mathbb{R}^d}$$

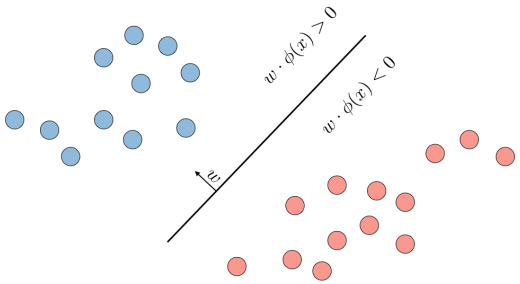
❒ **Score** – The score $s(x,w)$ of an example $(\phi(x),y) \in \mathbb{R}^d \times \mathbb{R}$ associated to a linear model of weights $w \in \mathbb{R}^d$ is given by the inner product:

$$\boxed{s(x,w) = w \cdot \phi(x)}$$

1.1.1 Classification

❒ **Linear classifier** – Given a weight vector $w \in \mathbb{R}^d$ and a feature vector $\phi(x) \in \mathbb{R}^d$, the binary linear classifier f_w is given by:

$$\boxed{f_w(x) = \text{sign}(s(x,w)) = \begin{cases} +1 & \text{if } w \cdot \phi(x) > 0 \\ -1 & \text{if } w \cdot \phi(x) < 0 \\ ? & \text{if } w \cdot \phi(x) = 0 \end{cases}}$$



❒ **Margin** – The margin $m(x,y,w) \in \mathbb{R}$ of an example $(\phi(x),y) \in \mathbb{R}^d \times \{-1,+1\}$ associated to a linear model of weights $w \in \mathbb{R}^d$ quantifies the confidence of the prediction: larger values are better. It is given by:

$$\boxed{m(x,y,w) = s(x,w) \times y}$$

1.1.2 Regression

❒ **Linear regression** – Given a weight vector $w \in \mathbb{R}^d$ and a feature vector $\phi(x) \in \mathbb{R}^d$, the output of a linear regression of weights w denoted as f_w is given by:

$$\boxed{f_w(x) = s(x,w)}$$

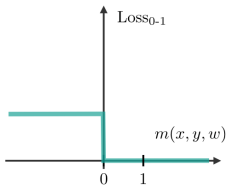
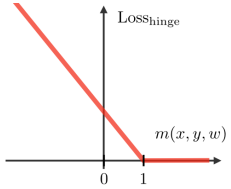
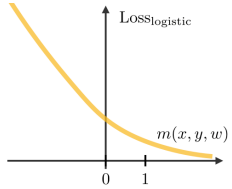
❒ **Residual** – The residual $\text{res}(x,y,w) \in \mathbb{R}$ is defined as being the amount by which the prediction $f_w(x)$ overshoots the target y:

$$\boxed{\text{res}(x,y,w) = f_w(x) - y}$$

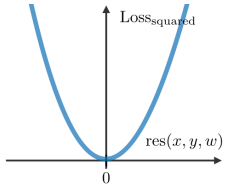
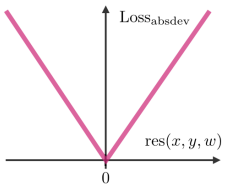
1.2 Loss minimization

❒ **Loss function** – A loss function $\text{Loss}(x,y,w)$ quantifies how unhappy we are with the weights w of the model in the prediction task of output y from input x. It is a quantity we want to minimize during the training process.

❒ **Classification case** – The classification of a sample x of true label $y \in \{-1,+1\}$ with a linear model of weights w can be done with the predictor $f_w(x) \triangleq \text{sign}(s(x,w))$. In this situation, a metric of interest quantifying the quality of the classification is given by the margin $m(x,y,w)$, and can be used with the following loss functions:

Name	Zero-one loss	Hinge loss	Logistic loss
$\text{Loss}(x,y,w)$	$1_{\{m(x,y,w) \leqslant 0\}}$	$\max(1 - m(x,y,w), 0)$	$\log(1 + e^{-m(x,y,w)})$
Illustration			

❒ **Regression case** – The prediction of a sample x of true label $y \in \mathbb{R}$ with a linear model of weights w can be done with the predictor $f_w(x) \triangleq s(x,w)$. In this situation, a metric of interest quantifying the quality of the regression is given by the margin $\text{res}(x,y,w)$ and can be used with the following loss functions:

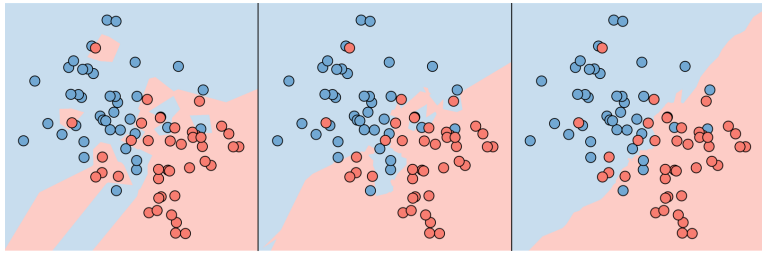
Name	Squared loss	Absolute deviation loss
$\text{Loss}(x,y,w)$	$(\text{res}(x,y,w))^2$	$\lvert\text{res}(x,y,w)\rvert$
Illustration		

❒ **Loss minimization framework** – In order to train a model, we want to minimize the training loss is defined as follows:

$$\boxed{\text{TrainLoss}(w) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x,y,w)}$$

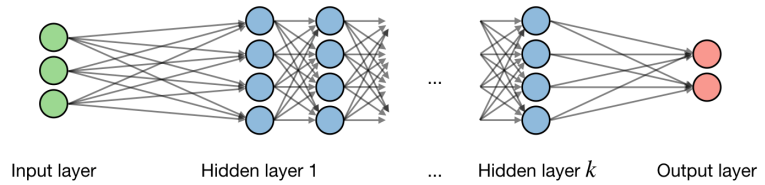
1.3 Non-linear predictors

❒ **k-nearest neighbors** – The k-nearest neighbors algorithm, commonly known as k-NN, is a non-parametric approach where the response of a data point is determined by the nature of its k neighbors from the training set. It can be used in both classification and regression settings.



Remark: the higher the parameter k, the higher the bias, and the lower the parameter k, the higher the variance.

❒ **Neural networks** – Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks. The vocabulary around neural networks architectures is described in the figure below:



By noting i the i^{th} layer of the network and j the j^{th} hidden unit of the layer, we have:

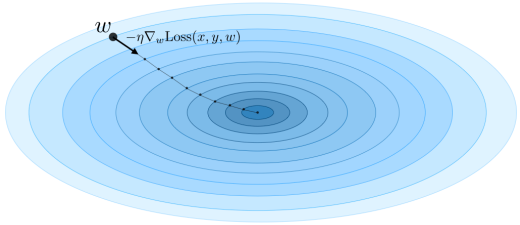
$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

where we note w, b, x, z the weight, bias, input and non-activated output of the neuron respectively.

1.4 Stochastic gradient descent

❒ **Gradient descent** – By noting $\eta \in \mathbb{R}$ the learning rate (also called step size), the update rule for gradient descent is expressed with the learning rate and the loss function $\text{Loss}(x,y,w)$ as follows:

$$\boxed{w \longleftarrow w - \eta\nabla_w \text{Loss}(x,y,w)}$$



❒ **Stochastic updates** – Stochastic gradient descent (SGD) updates the parameters of the model one training example $(\phi(x),y) \in \mathcal{D}_{\text{train}}$ at a time. This method leads to sometimes noisy, but fast updates.

❒ **Batch updates** – Batch gradient descent (BGD) updates the parameters of the model one batch of examples (e.g. the entire training set) at a time. This method computes stable update directions, at a greater computational cost.

1.5 Fine-tuning models

❒ **Hypothesis class** – A hypothesis class $\mathcal{F}$ is the set of possible predictors with a fixed $\phi(x)$ and varying w:

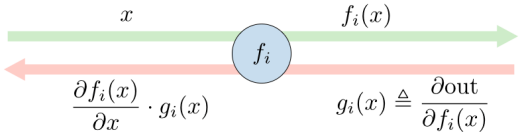
$$\boxed{\mathcal{F} = \left\{f_w : w \in \mathbb{R}^d\right\}}$$

❒ **Logistic function** – The logistic function σ, also called the sigmoid function, is defined as:

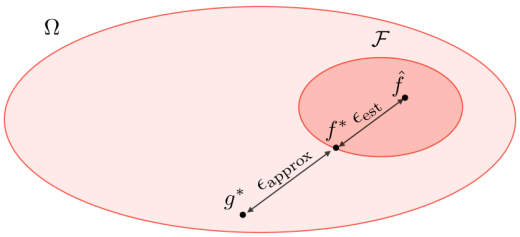
$$\boxed{\forall z \in]-\infty,+\infty[, \quad \sigma(z) = \frac{1}{1+e^{-z}}}$$

Remark: we have $\sigma'(z) = \sigma(z)(1-\sigma(z))$.

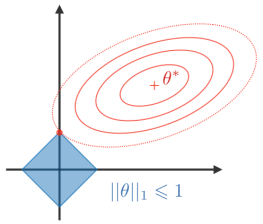
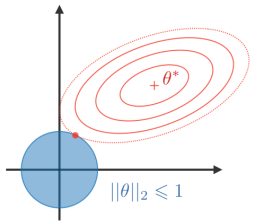
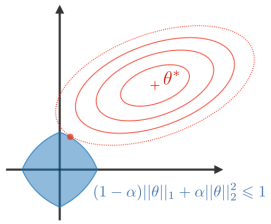
❒ **Backpropagation** – The forward pass is done through f_i, which is the value for the subexpression rooted at i, while the backward pass is done through $g_i = \frac{\partial \text{out}}{\partial f_i}$ and represents how f_i influences the output.



❒ **Approximation and estimation error** – The approximation error ϵ_{approx} represents how far the entire hypothesis class $\mathcal{F}$ is from the target predictor g^*, while the estimation error ϵ_{est} quantifies how good the predictor $\hat{f}$ is with respect to the best predictor f^* of the hypothesis class $\mathcal{F}$.



❒ **Regularization** – The regularization procedure aims at avoiding the model to overfit the data and thus deals with high variance issues. The following table sums up the different types of commonly used regularization techniques:

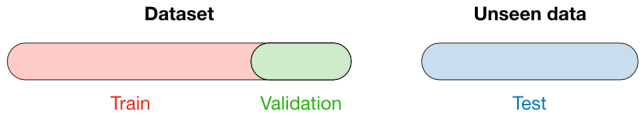
LASSO	Ridge	Elastic Net
- Shrinks coefficients to 0 - Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}$, $\alpha \in [0,1]$

❒ **Hyperparameters** – Hyperparameters are the properties of the learning algorithm, and include features, regularization parameter λ, number of iterations T, step size η, etc.

❒ **Sets vocabulary** – When selecting a model, we distinguish 3 different parts of the data that we have as follows:

Training set	Validation set	Testing set
- Model is trained - Usually 80 of the dataset	- Model is assessed - Usually 20 of the dataset - Also called hold-out	- Model gives predictions - Unseen data or development set

Once the model has been chosen, it is trained on the entire dataset and tested on the unseen test set. These are represented in the figure below:



1.6 Unsupervised Learning

The class of unsupervised learning methods aims at discovering the structure of the data, which may have of rich latent structures.

1.6.1 k-means

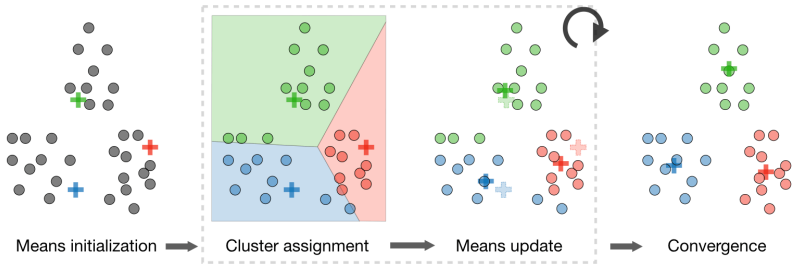
❒ **Clustering** – Given a training set of input points $\mathcal{D}_{\text{train}}$, the goal of a clustering algorithm is to assign each point $\phi(x_i)$ to a cluster $z_i \in \{1,\ldots,k\}$.

❒ **Objective function** – The loss function for one of the main clustering algorithms, k-means, is given by:

$$\boxed{\text{Loss}_{\text{k-means}}(x,\mu) = \sum_{i=1}^{n} ||\phi(x_i) - \mu_{z_i}||^2}$$

❒ **Algorithm** – After randomly initializing the cluster centroids $\mu_1,\mu_2,\ldots,\mu_k \in \mathbb{R}^n$, the k-means algorithm repeats the following step until convergence:

$$\boxed{z_i = \arg\min_j ||\phi(x_i) - \mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^{m} 1_{\{z_i=j\}}\phi(x_i)}{\sum_{i=1}^{m} 1_{\{z_i=j\}}}}$$



1.6.2 Principal Component Analysis

❒ **Eigenvalue, eigenvector** – Given a matrix $A \in \mathbb{R}^{n\times n}$, λ is said to be an eigenvalue of A if there exists a vector $z \in \mathbb{R}^n \backslash \{0\}$, called eigenvector, such that we have:

$$\boxed{Az = \lambda z}$$

❒ **Spectral theorem** – Let $A \in \mathbb{R}^{n\times n}$. If A is symmetric, then A is diagonalizable by a real orthogonal matrix $U \in \mathbb{R}^{n\times n}$. By noting $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, we have:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

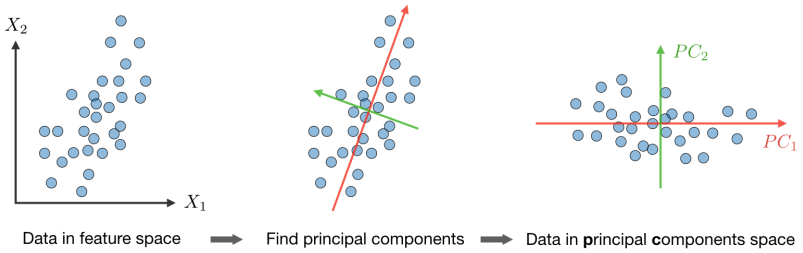
Remark: the eigenvector associated with the largest eigenvalue is called principal eigenvector of matrix A.

❒ **Algorithm** – The Principal Component Analysis (PCA) procedure is a dimension reduction technique that projects the data on k dimensions by maximizing the variance of the data as follows:

- Step 1: Normalize the data to have a mean of 0 and standard deviation of 1.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

- Step 2: Compute $\Sigma = \dfrac{1}{m}\displaystyle\sum_{i=1}^{m} x^{(i)}x^{(i)T} \in \mathbb{R}^{n\times n}$, which is symmetric with real eigenvalues.
- Step 3: Compute $u_1,...,u_k \in \mathbb{R}^n$ the k orthogonal principal eigenvectors of Σ, i.e. the orthogonal eigenvectors of the k largest eigenvalues.
- Step 4: Project the data on $\text{span}_{\mathbb{R}}(u_1,...,u_k)$. This procedure maximizes the variance among all k-dimensional spaces.



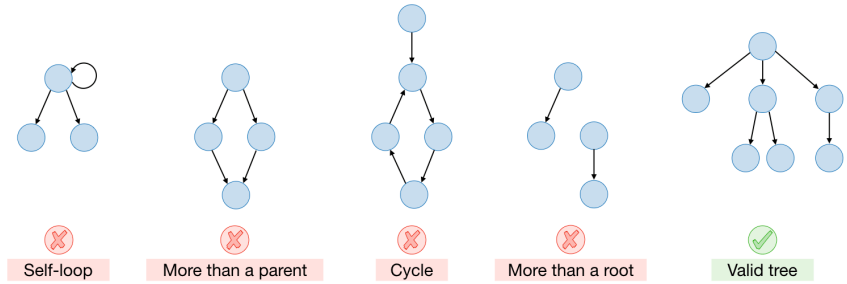
2 States-based models

2.1 Search optimization

In this section, we assume that by accomplishing action a from state s, we deterministically arrive in state Succ(s,a). The goal here is to determine a sequence of actions (a_1,a_2,a_3,a_4,...) that starts from an initial state and leads to an end state. In order to solve this kind of problem, our objective will be to find the minimum cost path by using states-based models.

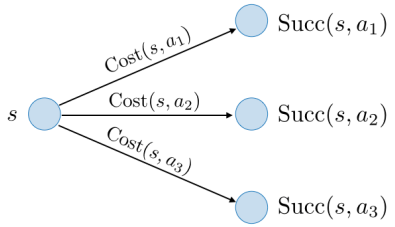
2.1.1 Tree search

This category of states-based algorithms explores all possible states and actions. It is quite memory efficient, and is suitable for huge state spaces but the runtime can become exponential in the worst cases.



❒ **Search problem** – A search problem is defined with:

- a starting state s_{start}
- possible actions Actions(s) from state s
- action cost Cost(s,a) from state s with action a
- successor Succ(s,a) of state s after action a
- whether an end state was reached IsEnd(s)

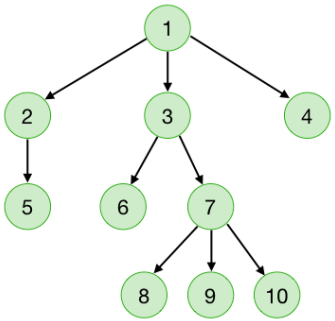


The objective is to find a path that minimizes the cost.

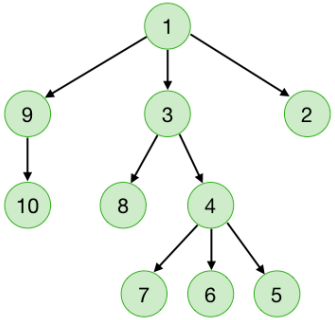
❒ **Backtracking search** – Backtracking search is a naive recursive algorithm that tries all possibilities to find the minimum cost path. Here, action costs can be either positive or negative.

❒ **Breadth-first search (BFS)** – Breadth-first search is a graph search algorithm that does a level-by-level traversal. We can implement it iteratively with the help of a queue that stores at

each step future nodes to be visited. For this algorithm, we can assume action costs to be equal to a constant $c \geqslant 0$.



❒ **Depth-first search (DFS)** – Depth-first search is a search algorithm that traverses a graph by following each path as deep as it can. We can implement it recursively, or iteratively with the help of a stack that stores at each step future nodes to be visited. For this algorithm, action costs are assumed to be equal to 0.



❒ **Iterative deepening** – The iterative deepening trick is a modification of the depth-first search algorithm so that it stops after reaching a certain depth, which guarantees optimality when all action costs are equal. Here, we assume that action costs are equal to a constant $c \geqslant 0$.

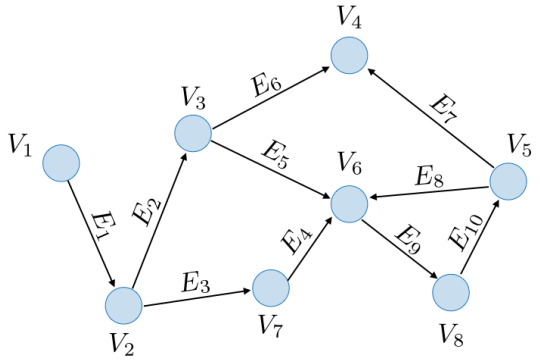
❒ **Tree search algorithms summary** – By noting b the number of actions per state, d the solution depth, and D the maximum depth, we have:

Algorithm	Action costs	Space	Time
Backtracking search	any	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
Breadth-first search	$c \geqslant 0$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$
Depth-first search	0	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
DFS-Iterative deepening	$c \geqslant 0$	$\mathcal{O}(d)$	$\mathcal{O}(b^d)$

2.1.2 Graph search

This category of states-based algorithms aims at constructing optimal paths, enabling exponential savings. In this section, we will focus on dynamic programming and uniform cost search.

❒ **Graph** – A graph is comprised of a set of vertices V (also called nodes) as well as a set of edges E (also called links).

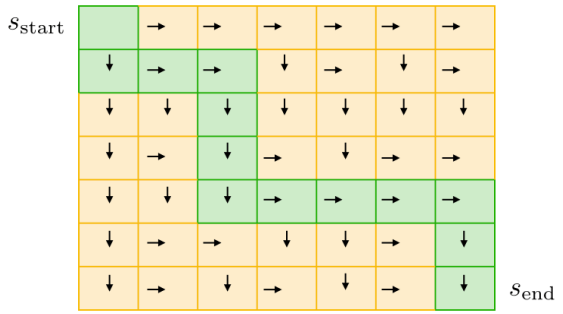


Remark: a graph is said to be acylic when there is no cycle.

❒ **State** – A state is a summary of all past actions sufficient to choose future actions optimally.

❒ **Dynamic programming** – Dynamic programming (DP) is a backtracking search algorithm with memoization (i.e. partial results are saved) whose goal is to find a minimum cost path from state s to an end state s_{end}. It can potentially have exponential savings compared to traditional graph search algorithms, and has the property to only work for acyclic graphs. For any given state s, the future cost is computed as follows:

$$\text{FutureCost}(s) = \begin{cases} 0 & \text{if IsEnd}(s) \\ \min\limits_{a \in \text{Actions}(s)} \big[\text{Cost}(s,a) + \text{FutureCost}(\text{Succ}(s,a))\big] & \text{otherwise} \end{cases}$$

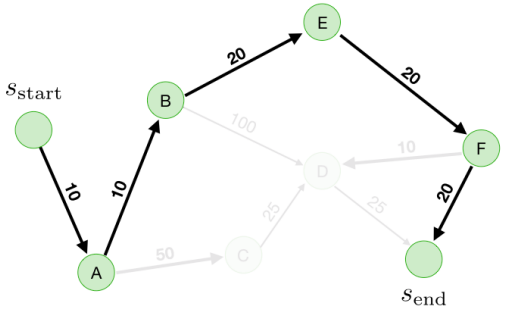


Remark: the figure above illustrates a bottom-to-top approach whereas the formula provides the intuition of a top-to-bottom problem resolution.

❒ **Types of states** – The table below presents the terminology when it comes to states in the context of uniform cost search:

State	Explanation
Explored $\mathcal{E}$	States for which the optimal path has already been found
Frontier $\mathcal{F}$	States seen for which we are still figuring out how to get there with the cheapest cost
Unexplored $\mathcal{U}$	States not seen yet

❒ **Uniform cost search** – Uniform cost search (UCS) is a search algorithm that aims at finding the shortest path from a state s_{start} to an end state s_{end}. It explores states s in increasing order of PastCost(s) and relies on the fact that all action costs are non-negative.



Remark 1: the UCS algorithm is logically equivalent to Djikstra's algorithm.
Remark 2: the algorithm would not work for a problem with negative action costs, and adding a positive constant to make them non-negative would not solve the problem since this would end up being a different problem.

❒ **Correctness theorem** – When a state s is popped from the frontier $\mathcal{F}$ and moved to explored set $\mathcal{E}$, its priority is equal to PastCost(s) which is the minimum cost path from s_{start} to s.

❒ **Graph search algorithms summary** – By noting N the number of total states, n of which are explored before the end state s_{end}, we have:

Algorithm	Acyclicity	Costs	Time/space
Dynamic programming	yes	any	$\mathcal{O}(N)$
Uniform cost search	no	$c \geqslant 0$	$\mathcal{O}(n\log(n))$

Remark: the complexity countdown supposes the number of possible actions per state to be constant.

2.1.3 Learning costs

Suppose we are not given the values of Cost(s,a), we want to estimate these quantities from a training set of minimizing-cost-path sequence of actions $(a_1, a_2, ..., a_k)$.

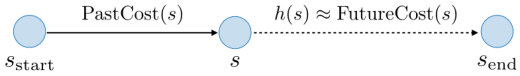
❒ **Structured perceptron** – The structured perceptron is an algorithm aiming at iteratively learning the cost of each state-action pair. At each step, it:

- decreases the estimated cost of each state-action of the true minimizing path y given by the training data,
- increases the estimated cost of each state-action of the current predicted path y' inferred from the learned weights.

Remark: there are several versions of the algorithm, one of which simplifies the problem to only learning the cost of each action a, and the other parametrizes Cost(s,a) to a feature vector of learnable weights.

2.1.4 A* search

❒ **Heuristic function** – A heuristic is a function h over states s, where each $h(s)$ aims at estimating FutureCost(s), the cost of the path from s to s_{end}.



❒ **Algorithm** – A^* is a search algorithm that aims at finding the shortest path from a state s to an end state s_{end}. It explores states s in increasing order of PastCost$(s) + h(s)$. It is equivalent to a uniform cost search with edge costs Cost$'(s,a)$ given by:

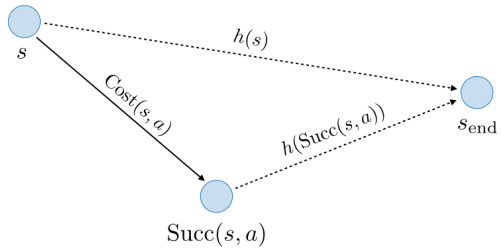
$$\boxed{\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)}$$

Remark: this algorithm can be seen as a biased version of UCS exploring states estimated to be closer to the end state.

❒ **Consistency** – A heuristic h is said to be consistent if it satisfies the two following properties:

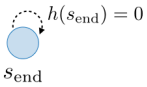
- For all states s and actions a,

$$\boxed{h(s) \leqslant \text{Cost}(s,a) + h(\text{Succ}(s,a))}$$



- The end state verifies the following:

$$\boxed{h(s_{\text{end}}) = 0}$$



❒ **Correctness** – If h is consistent, then A^* returns the minimum cost path.

❒ **Admissibility** – A heuristic h is said to be admissible if we have:

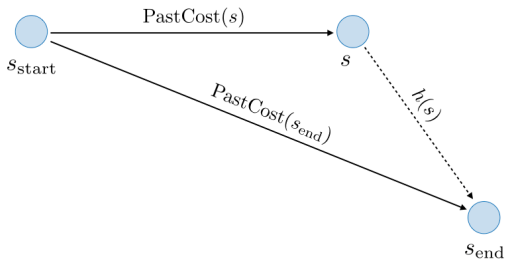
$$\boxed{h(s) \leqslant \text{FutureCost}(s)}$$

❒ **Theorem** – Let $h(s)$ be a given heuristic. We have:

$$\boxed{h(s) \text{ consistent} \implies h(s) \text{ admissible}}$$

❒ **Efficiency** – A^* explores all states s satisfying the following equation:

$$\boxed{\text{PastCost}(s) \leqslant \text{PastCost}(s_{\text{end}}) - h(s)}$$



Remark: larger values of $h(s)$ is better as this equation shows it will restrict the set of states s going to be explored.

2.1.5 Relaxation

It is a framework for producing consistent heuristics. The idea is to find closed-form reduced costs by removing constraints and use them as heuristics.

❒ **Relaxed search problem** – The relaxation of search problem P with costs Cost is noted P_{rel} with costs Cost_{rel}, and satisfies the identity:

$$\boxed{\text{Cost}_{\text{rel}}(s,a) \leqslant \text{Cost}(s,a)}$$

❒ **Relaxed heuristic** – Given a relaxed search problem P_{rel}, we define the relaxed heuristic $h(s) = \text{FutureCost}_{\text{rel}}(s)$ as the minimum cost path from s to an end state in the graph of costs $\text{Cost}_{\text{rel}}(s,a)$.

❒ **Consistency of relaxed heuristics** – Let P_{rel} be a given relaxed problem. By theorem, we have:

$$\boxed{h(s) = \text{FutureCost}_{\text{rel}}(s) \implies h(s) \text{ consistent}}$$

❒ **Tradeoff when choosing heuristic** – We have to balance two aspects in choosing a heuristic:

- Computational efficiency: $h(s) = \text{FutureCost}_{\text{rel}}(s)$ must be easy to compute. It has to produce a closed form, easier search and independent subproblems.
- Good enough approximation: the heuristic $h(s)$ should be close to $\text{FutureCost}(s)$ and we have thus to not remove too many constraints.

❒ **Max heuristic** – Let $h_1(s)$, $h_2(s)$ be two heuristics. We have the following property:

$$\boxed{h_1(s),\, h_2(s) \text{ consistent} \implies h(s) = \max\{h_1(s),\, h_2(s)\} \text{ consistent}}$$

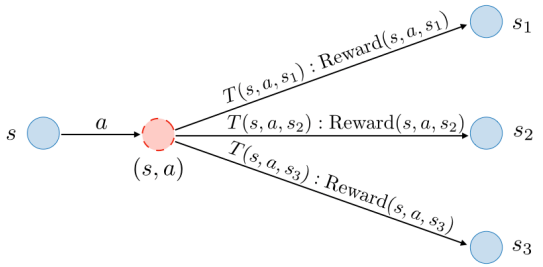
2.2 Markov decision processes

In this section, we assume that performing action a from state s can lead to several states s'_1,s'_2,... in a probabilistic manner. In order to find our way between an initial state and an end state, our objective will be to find the maximum value policy by using Markov decision processes that help us cope with randomness and uncertainty.

2.2.1 Notations

❒ **Definition** – The objective of a Markov decision process is to maximize rewards. It is defined with:

- a starting state s_{start}
- possible actions $\text{Actions}(s)$ from state s
- transition probabilities $T(s,a,s')$ from s to s' with action a
- rewards $\text{Reward}(s,a,s')$ from s to s' with action a
- whether an end state was reached $\text{IsEnd}(s)$
- a discount factor $0 \leqslant \gamma \leqslant 1$



❒ **Transition probabilities** – The transition probability $T(s,a,s')$ specifies the probability of going to state s' after action a is taken in state s. Each $s' \mapsto T(s,a,s')$ is a probability distribution, which means that:

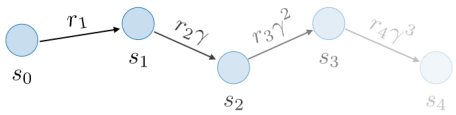
$$\forall s,a, \quad \boxed{\sum_{s' \in \text{ States}} T(s,a,s') = 1}$$

❒ **Policy** – A policy π is a function that maps each state s to an action a, i.e.

$$\boxed{\pi : s \mapsto a}$$

❒ **Utility** – The utility of a path $(s_0, ..., s_k)$ is the discounted sum of the rewards on that path. In other words,

$$\boxed{u(s_0,...,s_k) = \sum_{i=1}^{k} r_i \gamma^{i-1}}$$



Remark: the figure above is an illustration of the case $k = 4$.

❒ **Q-value** – The Q-value of a policy π by taking action a from state s, also noted $Q_\pi(s,a)$, is the expected utility of taking action a from state s and then following policy π. It is defined as follows:

$$\boxed{Q_\pi(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_\pi(s')\right]}$$

❒ **Value of a policy** – The value of a policy π from state s, also noted $V_\pi(s)$, is the expected utility by following policy π from state s over random paths. It is defined as follows:

$$\boxed{V_\pi(s) = Q_\pi(s,\pi(s))}$$

Remark: $V_\pi(s)$ is equal to 0 if s is an end state.

2.2.2 Applications

❒ **Policy evaluation** – Given a policy π, policy evaluation is an iterative algorithm that computes V_π. It is done as follows:

- Initialization: for all states s, we have

$$\boxed{V_\pi^{(0)}(s) \longleftarrow 0}$$

- Iteration: for t from 1 to T_{PE}, we have

$$\forall s, \quad \boxed{V_\pi^{(t)}(s) \longleftarrow Q_\pi^{(t-1)}(s,\pi(s))}$$

with

$$\boxed{Q_\pi^{(t-1)}(s,\pi(s)) = \sum_{s' \in \text{ States}} T(s,\pi(s),s') \left[\text{Reward}(s,\pi(s),s') + \gamma V_\pi^{(t-1)}(s')\right]}$$

Remark: by noting S the number of states, A the number of actions per state, S' the number of successors and T the number of iterations, then the time complexity is of $\mathcal{O}(T_{PE}SS')$.

❒ **Optimal Q-value** – The optimal Q-value $Q_{\text{opt}}(s,a)$ of state s with action a is defined to be the maximum Q-value attained by any policy starting. It is computed as follows:

$$\boxed{Q_{\text{opt}}(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')\right]}$$

❒ **Optimal value** – The optimal value $V_{\text{opt}}(s)$ of state s is defined as being the maximum value attained by any policy. It is computed as follows:

$$\boxed{V_{\text{opt}}(s) = \max_{a \in \text{ Actions}(s)} Q_{\text{opt}}(s,a)}$$

❒ **Optimal policy** – The optimal policy π_{opt} is defined as being the policy that leads to the optimal values. It is defined by:

$$\forall s, \quad \boxed{\pi_{\text{opt}}(s) = \operatorname*{argmax}_{a \in \text{ Actions}(s)} Q_{\text{opt}}(s,a)}$$

❒ **Value iteration** – Value iteration is an algorithm that finds the optimal value V_{opt} as well as the optimal policy π_{opt}. It is done as follows:

- Initialization: for all states s, we have

$$\boxed{V_{\text{opt}}^{(0)}(s) \longleftarrow 0}$$

- Iteration: for t from 1 to T_{VI}, we have

$$\forall s, \quad \boxed{V_{\text{opt}}^{(t)}(s) \longleftarrow \max_{a \in \text{ Actions}(s)} Q_{\text{opt}}^{(t-1)}(s,a)}$$

with

$$\boxed{Q_{\text{opt}}^{(t-1)}(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')\right]}$$

Remark: if we have either $\gamma < 1$ or the MDP graph being acyclic, then the value iteration algorithm is guaranteed to converge to the correct answer.

2.2.3 When unknown transitions and rewards

Now, let's assume that the transition probabilities and the rewards are unknown.

❒ **Model-based Monte Carlo** – The model-based Monte Carlo method aims at estimating $T(s,a,s')$ and $\text{Reward}(s,a,s')$ using Monte Carlo simulation with:

$$\boxed{\widehat{T}(s,a,s') = \frac{\#\text{ times } (s,a,s') \text{ occurs}}{\#\text{ times } (s,a) \text{ occurs}}}$$

and

$$\boxed{\widehat{\text{Reward}}(s,a,s') = r \text{ in } (s,a,r,s')}$$

These estimations will be then used to deduce Q-values, including Q_π and Q_{opt}.

Remark: model-based Monte Carlo is said to be off-policy, because the estimation does not depend on the exact policy.

❒ **Model-free Monte Carlo** – The model-free Monte Carlo method aims at directly estimating Q_π, as follows:

$$\boxed{\widehat{Q}_\pi(s,a) = \text{average of } u_t \text{ where } s_{t-1} = s, a_t = a}$$

where u_t denotes the utility starting at step t of a given episode.

Remark: model-free Monte Carlo is said to be on-policy, because the estimated value is dependent on the policy π used to generate the data.

❒ **Equivalent formulation** – By introducing the constant $\eta = \frac{1}{1+(\#\text{updates to } (s,a))}$ and for each (s,a,u) of the training set, the update rule of model-free Monte Carlo has a convex combination formulation:

$$\boxed{\widehat{Q}_\pi(s,a) \leftarrow (1-\eta)\widehat{Q}_\pi(s,a) + \eta u}$$

as well as a stochastic gradient formulation:

$$\boxed{\widehat{Q}_\pi(s,a) \leftarrow \widehat{Q}_\pi(s,a) - \eta(\widehat{Q}_\pi(s,a) - u)}$$

❒ **SARSA** – State-action-reward-state-action (SARSA) is a boostrapping method estimating Q_π by using both raw data and estimates as part of the update rule. For each (s,a,r,s',a'), we have:

$$\boxed{\widehat{Q}_\pi(s,a) \longleftarrow (1-\eta)\widehat{Q}_\pi(s,a) + \eta\Big[r + \gamma\widehat{Q}_\pi(s',a')\Big]}$$

Remark: the SARSA estimate is updated on the fly as opposed to the model-free Monte Carlo one where the estimate can only be updated at the end of the episode.

❒ **Q-learning** – Q-learning is an off-policy algorithm that produces an estimate for Q_{opt}. On each (s,a,r,s',a'), we have:

$$\boxed{\widehat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\widehat{Q}_{\text{opt}}(s,a) + \eta\Big[r + \gamma \max_{a' \in \text{ Actions}(s')} \widehat{Q}_{\text{opt}}(s',a')\Big]}$$

❒ **Epsilon-greedy** – The epsilon-greedy policy is an algorithm that balances exploration with probability ϵ and exploitation with probability $1-\epsilon$. For a given state s, the policy π_{act} is computed as follows:

$$\boxed{\pi_{\text{act}}(s) = \begin{cases} \underset{a \in \text{ Actions}}{\text{argmax}}\ \widehat{Q}_{\text{opt}}(s,a) & \text{with proba } 1-\epsilon \\ \text{random from Actions}(s) & \text{with proba } \epsilon \end{cases}}$$

2.3 Game playing

In games (e.g. chess, backgammon, Go), other agents are present and need to be taken into account when constructing our policy.

❒ **Game tree** – A game tree is a tree that describes the possibilities of a game. In particular, each node is a decision point for a player and each root-to-leaf path is a possible outcome of the game.

❒ **Two-player zero-sum game** – It is a game where each state is fully observed and such that players take turns. It is defined with:

- a starting state s_{start}
- possible actions $\text{Actions}(s)$ from state s
- successors $\text{Succ}(s,a)$ from states s with actions a
- whether an end state was reached $\text{IsEnd}(s)$
- the agent's utility $\text{Utility}(s)$ at end state s
- the player $\text{Player}(s)$ who controls state s

Remark: we will assume that the utility of the agent has the opposite sign of the one of the opponent.

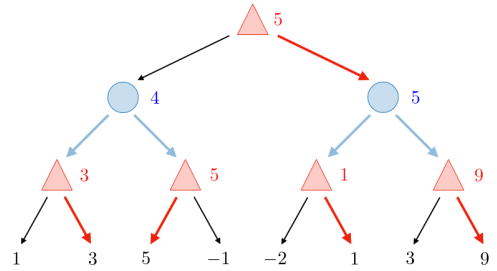
❒ **Types of policies** – There are two types of policies:

- Deterministic policies, noted $\pi_p(s)$, which are actions that player p takes in state s.
- Stochastic policies, noted $\pi_p(s,a) \in [0,1]$, which are probabilities that player p takes action a in state s.

❒ **Expectimax** – For a given state s, the expectimax value $V_{\text{exptmax}}(s)$ is the maximum expected utility of any agent policy when playing with respect to a fixed and known opponent policy π_{opp}. It is computed as follows:

$$\boxed{V_{\text{exptmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum\limits_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}}$$

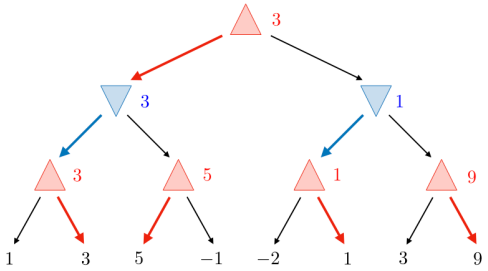
Remark: expectimax is the analog of value iteration for MDPs.



❒ **Minimax** – The goal of minimax policies is to find an optimal policy against an adversary by assuming the worst case, i.e. that the opponent is doing everything to minimize the agent's utility. It is done as follows:

$$\boxed{V_{\text{minimax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min\limits_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}}$$

Remark: we can extract π_{max} and π_{min} from the minimax value $V_{minimax}$.



❒ **Minimax properties** – By noting V the value function, there are 3 properties around minimax to have in mind:

- *Property 1*: if the agent were to change its policy to any π_{agent}, then the agent would be no better off.

$$\boxed{\forall \pi_{\text{agent}}, \quad V(\pi_{\max},\pi_{\min}) \geqslant V(\pi_{\text{agent}},\pi_{\min})}$$

- *Property 2*: if the opponent changes its policy from $\pi_{\min}$ to π_{opp}, then he will be no better off.

$$\boxed{\forall \pi_{\text{opp}}, \quad V(\pi_{\max},\pi_{\min}) \leqslant V(\pi_{\max},\pi_{\text{opp}})}$$

- *Property 3*: if the opponent is known to be not playing the adversarial policy, then the minimax policy might not be optimal for the agent.

$$\boxed{\forall \pi, \quad V(\pi_{\max},\pi) \leqslant V(\pi_{\text{exptmax}},\pi)}$$

In the end, we have the following relationship:

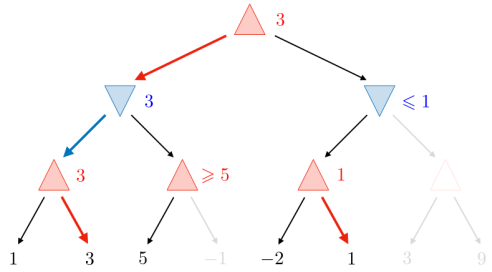
$$\boxed{V(\pi_{\text{exptmax}},\pi_{\min}) \leqslant V(\pi_{\max},\pi_{\min}) \leqslant V(\pi_{\max},\pi) \leqslant V(\pi_{\text{exptmax}},\pi)}$$

2.3.1 Speeding up minimax

❒ **Evaluation function** – An evaluation function is a domain-specific and approximate estimate of the value $V_{\text{minimax}}(s)$. It is noted $\text{Eval}(s)$.

Remark: $\text{FutureCost}(s)$ is an analogy for search problems.

❒ **Alpha-beta pruning** – Alpha-beta pruning is a domain-general exact method optimizing the minimax algorithm by avoiding the unnecessary exploration of parts of the game tree. To do so, each player keeps track of the best value they can hope for (stored in α for the maximizing player and in β for the minimizing player). At a given step, the condition $\beta < \alpha$ means that the optimal path is not going to be in the current branch as the earlier player had a better option at their disposal.



❒ **TD learning** – Temporal difference (TD) learning is used when we don't know the transitions/rewards. The value is based on exploration policy. To be able to use it, we need to know rules of the game $\text{Succ}(s,a)$. For each (s,a,r,s'), the update is done as follows:

$$\boxed{w \longleftarrow w - \eta\Big[V(s,w) - (r + \gamma V(s',w))\Big]\nabla_w V(s,w)}$$

2.3.2 Simultaneous games

This is the contrary of turn-based games, where there is no ordering on the player's moves.

❒ **Single-move simultaneous game** – Let there be two players A and B, with given possible actions. We note $V(a,b)$ to be A's utility if A chooses action a, B chooses action b. V is called the payoff matrix.

❒ **Strategies** – There are two main types of strategies:

- A pure strategy is a single action:

$$\boxed{a \in \text{Actions}}$$

- A mixed strategy is a probability distribution over actions:

$$\forall a \in \text{Actions}, \quad \boxed{0 \leqslant \pi(a) \leqslant 1}$$

❒ **Game evaluation** – The value of the game $V(\pi_A,\pi_B)$ when player A follows π_A and player B follows π_B is such that:

$$\boxed{V(\pi_A,\pi_B) = \sum_{a,b} \pi_A(a)\pi_B(b)V(a,b)}$$

❒ **Minimax theorem** – By noting π_A,π_B ranging over mixed strategies, for every simultaneous two-player zero-sum game with a finite number of actions, we have:

$$\boxed{\max_{\pi_A}\min_{\pi_B} V(\pi_A,\pi_B) = \min_{\pi_B}\max_{\pi_A} V(\pi_A,\pi_B)}$$

2.3.3 Non-zero-sum games

❒ **Payoff matrix** – We define $V_p(\pi_A,\pi_B)$ to be the utility for player p.

❒ **Nash equilibrium** – A Nash equilibrium is (π_A^*,π_B^*) such that no player has an incentive to change its strategy. We have:

$$\boxed{\forall \pi_A, V_A(\pi_A^*,\pi_B^*) \geqslant V_A(\pi_A,\pi_B^*)} \quad \text{and} \quad \boxed{\forall \pi_B, V_B(\pi_A^*,\pi_B^*) \geqslant V_B(\pi_A^*,\pi_B)}$$

Remark: in any finite-player game with finite number of actions, there exists at least one Nash equilibrium.

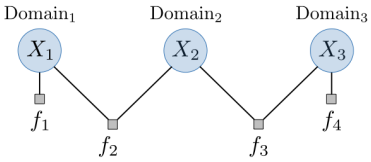
3 Variables-based models

3.1 Constraint satisfaction problems

In this section, our objective is to find maximum weight assignments of variable-based models. One advantage compared to states-based models is that these algorithms are more convenient to encode problem-specific constraints.

3.1.1 Factor graphs

❒ **Definition** – A factor graph, also referred to as a Markov random field, is a set of variables $X = (X_1,...,X_n)$ where $X_i \in \text{Domain}_i$ and m factors $f_1,...,f_m$ with each $f_j(X) \geqslant 0$.



❒ **Scope and arity** – The scope of a factor f_j is the set of variables it depends on. The size of this set is called the arity.

Remark: factors of arity 1 and 2 are called unary and binary respectively.

❒ **Assignment weight** – Each assignment $x = (x_1,...,x_n)$ yields a weight $\text{Weight}(x)$ defined as being the product of all factors f_j applied to that assignment. Its expression is given by:

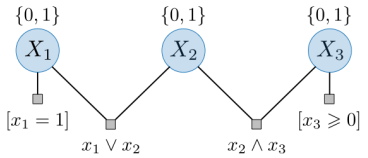
$$\boxed{\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)}$$

❒ **Constraint satisfaction problem** – A constraint satisfaction problem (CSP) is a factor graph where all factors are binary; we call them to be constraints:

$$\boxed{\forall j \in [\![1,m]\!], \quad f_j(x) \in \{0,1\}}$$

Here, the constraint j with assignment x is said to be satisfied if and only if $f_j(x) = 1$.

❒ **Consistent assignment** – An assignment x of a CSP is said to be consistent if and only if $\text{Weight}(x) = 1$, i.e. all constraints are satisfied.



3.1.2 Dynamic ordering

❒ **Dependent factors** – The set of dependent factors of variable X_i with partial assignment x is called $D(x,X_i)$, and denotes the set of factors that link X_i to already assigned variables.

❒ **Backtracking search** – Backtracking search is an algorithm used to find maximum weight assignments of a factor graph. At each step, it chooses an unassigned variable and explores its values by recursion. Dynamic ordering (*i.e.* choice of variables and values) and lookahead (*i.e.* early elimination of inconsistent options) can be used to explore the graph more efficiently, although the worst-case runtime stays exponential: $O(|\text{Domain}|^n)$.

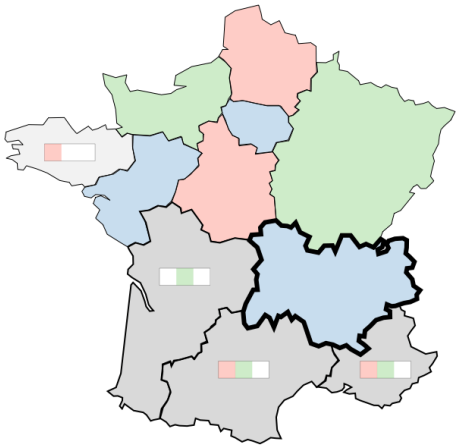
❒ **Forward checking** – It is a one-step lookahead heuristic that preemptively removes inconsistent values from the domains of neighboring variables. It has the following characteristics:

- After assigning a variable X_i, it eliminates inconsistent values from the domains of all its neighbors.
- If any of these domains becomes empty, we stop the local backtracking search.
- If we un-assign a variable X_i, we have to restore the domain of its neighbors.

❒ **Most constrained variable** – It is a variable-level ordering heuristic that selects the next unassigned variable that has the fewest consistent values. This has the effect of making inconsistent assignments to fail earlier in the search, which enables more efficient pruning.

❒ **Least constrained value** – It is a value-level ordering heuristic that assigns the next value that yields the highest number of consistent values of neighboring variables. Intuitively, this procedure chooses first the values that are most likely to work.

Remark: in practice, this heuristic is useful when all factors are constraints.



The example above is an illustration of the 3-color problem with backtracking search coupled with most constrained variable exploration and least constrained value heuristic, as well as forward checking at each step.

❒ **Arc consistency** – We say that arc consistency of variable X_l with respect to X_k is enforced when for each $x_l \in \text{Domain}_l$:

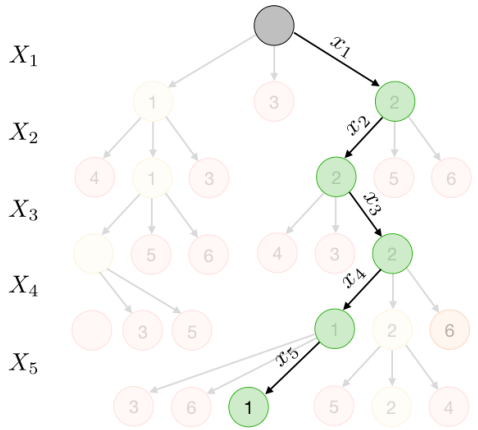
- unary factors of X_l are non-zero,
- there exists at least one $x_k \in \text{Domain}_k$ such that any factor between X_l and X_k is non-zero.

❒ **AC-3** – The AC-3 algorithm is a multi-step lookahead heuristic that applies forward checking to all relevant variables. After a given assignment, it performs forward checking and then successively enforces arc consistency with respect to the neighbors of variables for which the domain change during the process.

Remark: AC-3 can be implemented both iteratively and recursively.

3.1.3 Approximate methods

❒ **Beam search** – Beam search is an approximate algorithm that extends partial assignments of n variables of branching factor $b = |\text{Domain}|$ by exploring the K top paths at each step. The beam size $K \in \{1,...,b^n\}$ controls the tradeoff between efficiency and accuracy. This algorithm has a time complexity of $O(n \cdot Kb\log(Kb))$.
The example below illustrates a possible beam search of parameters $K = 2$, $b = 3$ and $n = 5$.



Remark: $K = 1$ corresponds to greedy search whereas $K \to +\infty$ is equivalent to BFS tree search.

❒ **Iterated conditional modes** – Iterated conditional modes (ICM) is an iterative approximate algorithm that modifies the assignment of a factor graph one variable at a time until convergence. At step i, we assign to X_i the value v that maximizes the product of all factors connected to that variable.

Remark: ICM may get stuck in local minima.

❒ **Gibbs sampling** – Gibbs sampling is an iterative approximate method that modifies the assignment of a factor graph one variable at a time until convergence. At step i:

- we assign to each element $u \in \text{Domain}_i$ a weight $w(u)$ that is the product of all factors connected to that variable,
- we sample v from the probability distribution induced by w and assign it to X_i.

Remark: Gibbs sampling can be seen as the probabilistic counterpart of ICM. It has the advantage to be able to escape local minima in most cases.

3.1.4 Factor graph transformations

❒ **Independence** – Let A,B be a partitioning of the variables X. We say that A and B are independent if there are no edges between A and B and we write:

$$\boxed{A,B \text{ independent} \iff A \perp\!\!\!\perp B}$$

Remark: independence is the key property that allows us to solve subproblems in parallel.

❒ **Conditional independence** – We say that A and B are conditionally independent given C if conditioning on C produces a graph in which A and B are independent. In this case, it is written:

$$\boxed{A \text{ and } B \text{ cond. indep. given } C \iff A \perp\!\!\!\perp B|C}$$

❒ **Conditioning** – Conditioning is a transformation aiming at making variables independent that breaks up a factor graph into smaller pieces that can be solved in parallel and can use backtracking. In order to condition on a variable $X_i = v$, we do as follows:

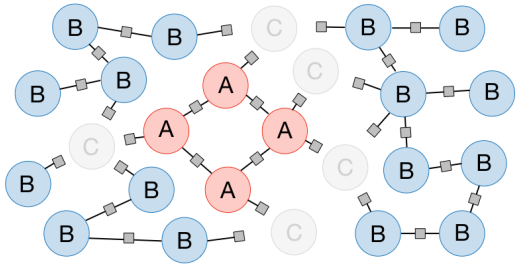
- Consider all factors $f_1,...,f_k$ that depend on X_i
- Remove X_i and $f_1,...,f_k$
- Add $g_j(x)$ for $j \in \{1,...,k\}$ defined as:

$$\boxed{g_j(x) = f_j(x \cup \{X_i : v\})}$$

❒ **Markov blanket** – Let $A \subseteq X$ be a subset of variables. We define MarkovBlanket(A) to be the neighbors of A that are not in A.

❒ **Proposition** – Let $C = \text{MarkovBlanket}(A)$ and $B = X \backslash (A \cup C)$. Then we have:

$$\boxed{A \perp\!\!\!\perp B|C}$$



❒ **Elimination** – Elimination is a factor graph transformation that removes X_i from the graph and solves a small subproblem conditioned on its Markov blanket as follows:

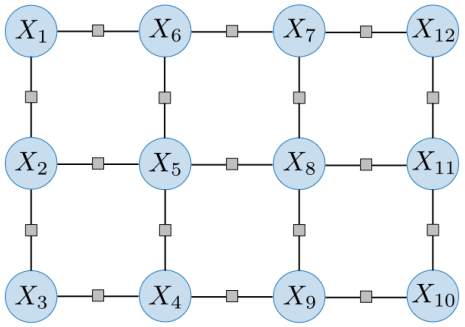
- Consider all factors $f_{i,1},...,f_{i,k}$ that depend on X_i
- Remove X_i and $f_{i,1},...,f_{i,k}$
- Add $f_{\text{new},i}(x)$ defined as:

$$\boxed{f_{\text{new},i}(x) = \max_{x_i} \prod_{l=1}^{k} f_{i,l}(x)}$$

❒ **Treewidth** – The treewidth of a factor graph is the maximum arity of any factor created by variable elimination with the best variable ordering. In other words,

$$\boxed{\text{Treewidth} = \min_{\text{orderings}} \max_{i \in \{1,...,n\}} \text{arity}(f_{\text{new},i})}$$

The example below illustrates the case of a factor graph of treewidth 3.



Remark: finding the best variable ordering is a NP-hard problem.

3.2 Bayesian networks

In this section, our goal will be to compute conditional probabilities. What is the probability of a query given evidence?

3.2.1 Introduction

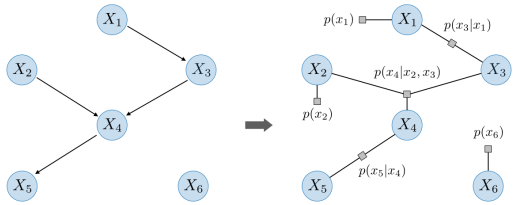
❒ **Explaining away** – Suppose causes C_1 and C_2 influence an effect E. Conditioning on the effect E and on one of the causes (say C_1) changes the probability of the other cause (say C_2). In this case, we say that C_1 has explained away C_2.

❒ **Directed acyclic graph** – A directed acyclic graph (DAG) is a finite directed graph with no directed cycles.

❒ **Bayesian network** – A Bayesian network is a directed acyclic graph (DAG) that specifies a joint distribution over random variables $X = (X_1,...,X_n)$ as a product of local conditional distributions, one for each node:

$$\boxed{P(X_1 = x_1,...,X_n = x_n) \triangleq \prod_{i=1}^{n} p(x_i|x_{\text{Parents}(i)})}$$

Remark: Bayesian networks are factor graphs imbued with the language of probability.



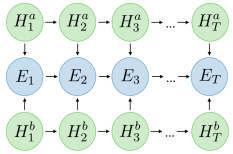
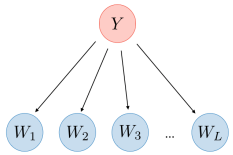
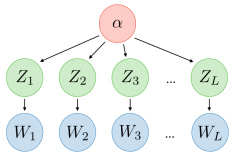
❒ **Locally normalized** – For each $x_{\text{Parents}(i)}$, all factors are local conditional distributions. Hence they have to satisfy:

$$\boxed{\sum_{x_i} p(x_i|x_{\text{Parents}(i)}) = 1}$$

As a result, sub-Bayesian networks and conditional distributions are consistent.

Remark: local conditional distributions are the true conditional distributions.

❒ **Marginalization** – The marginalization of a leaf node yields a Bayesian network without that node.

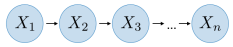
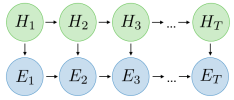
Factorial HMM	$H_t^o \underset{o\in\{a,b\}}{\sim} p(H_t^o\|H_{t-1}^o)$ $E_t \sim p(E_t\|H_t^a, H_t^b)$		Multiple object tracking
Naive Bayes	$Y \sim p(Y)$ $W_i \sim p(W_i\|Y)$		Document classification
Latent Dirichlet Allocation (LDA)	$\alpha \in \mathbb{R}^K$ distribution $Z_i \sim p(Z_i\|\alpha)$ $W_i \sim p(W_i\|Z_i)$		Topic modeling

3.2.2 Probabilistic programs

❒ **Concept** – A probabilistic program randomizes variables assignment. That way, we can write down complex Bayesian networks that generate assignments without us having to explicitly specify associated probabilities.

Remark: examples of probabilistic programs include Hidden Markov model (HMM), factorial HMM, naive Bayes, latent Dirichlet allocation, diseases and symptoms and stochastic block models.

❒ **Summary** – The table below summarizes the common probabilistic programs as well as their applications:

Program	Algorithm	Illustration	Example
Markov Model	$X_i \sim p(X_i\|X_{i-1})$		Language modeling
Hidden Markov Model (HMM)	$H_t \sim p(H_t\|H_{t-1})$ $E_t \sim p(E_t\|H_t)$		Object tracking

3.2.3 Inference

❒ **General probabilistic inference strategy** – The strategy to compute the probability $P(Q|E=e)$ of query Q given evidence $E=e$ is as follows:

- Step 1: Remove variables that are not ancestors of the query Q or the evidence E by marginalization
- Step 2: Convert Bayesian network to factor graph
- Step 3: Condition on the evidence $E=e$
- Step 4: Remove nodes disconnected from the query Q by marginalization
- Step 5: Run probabilistic inference algorithm (manual, variable elimination, Gibbs sampling, particle filtering)

❒ **Forward-backward algorithm** – This algorithm computes the exact value of $P(H=h_k|E=e)$ (smoothing query) for any $k \in \{1,...,L\}$ in the case of an HMM of size L. To do so, we proceed in 3 steps:

- Step 1: for $i \in \{1,...,L\}$, compute $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})p(h_i|h_{i-1})p(e_i|h_i)$
- Step 2: for $i \in \{L,...,1\}$, compute $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})p(h_{i+1}|h_i)p(e_{i+1}|h_{i+1})$
- Step 3: for $i \in \{1,...,L\}$, compute $S_i(h_i) = \frac{F_i(h_i)B_i(h_i)}{\sum_{h_i} F_i(h_i)B_i(h_i)}$

with the convention $F_0 = B_{L+1} = 1$. From this procedure and these notations, we get that

$$\boxed{P(H = h_k|E = e) = S_k(h_k)}$$

Remark: this algorithm interprets each assignment to be a path where each edge $h_{i-1} \to h_i$ is of weight $p(h_i|h_{i-1})p(e_i|h_i)$.

❒ **Gibbs sampling** – This algorithm is an iterative approximate method that uses a small set of assignments (particles) to represent a large probability distribution. From a random assignment x, Gibbs sampling performs the following steps for $i \in \{1,...,n\}$ until convergence:

- For all $u \in \text{Domain}_i$, compute the weight $w(u)$ of assignment x where $X_i = u$
- Sample v from the probability distribution induced by w: $v \sim P(X_i = v|X_{-i} = x_{-i})$
- Set $X_i = v$

Remark: X_{-i} denotes $X\backslash\{X_i\}$ and x_{-i} represents the corresponding assignment.

❒ **Particle filtering** – This algorithm approximates the posterior density of state variables given the evidence of observation variables by keeping track of K particles at a time. Starting from a set of particles C of size K, we run the following 3 steps iteratively:

- Step 1: proposal - For each old particle $x_{t-1} \in C$, sample x from the transition probability distribution $p(x|x_{t-1})$ and add x to a set C'.
- Step 2: weighting - Weigh each x of the set C' by $w(x) = p(e_t|x)$, where e_t is the evidence observed at time t.
- Step 3: resampling - Sample K elements from the set C' using the probability distribution induced by w and store them in C: these are the current particles x_t.

Remark: a more expensive version of this algorithm also keeps track of past particles in the proposal step.

❒ **Maximum likelihood** – If we don't know the local conditional distributions, we can learn them using maximum likelihood.

$$\boxed{\max_\theta \prod_{x \in \mathcal{D}_{\text{train}}} p(X = x; \theta)}$$

❒ **Laplace smoothing** – For each distribution d and partial assignment $(x_{\text{Parents}(i)}, x_i)$, add λ to $\text{count}_d(x_{\text{Parents}(i)}, x_i)$, then normalize to get probability estimates.

❒ **Algorithm** – The Expectation-Maximization (EM) algorithm gives an efficient method at estimating the parameter θ through maximum likelihood estimation by repeatedly constructing a lower-bound on the likelihood (E-step) and optimizing that lower bound (M-step) as follows:

- E-step: Evaluate the posterior probability $q(h)$ that each data point e came from a particular cluster h as follows:

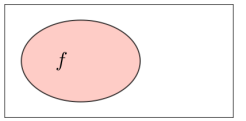
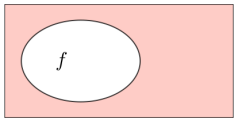
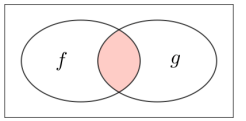
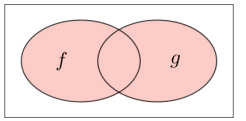
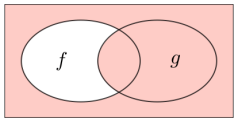
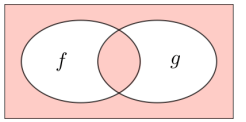
$$\boxed{q(h) = P(H = h|E = e; \theta)}$$

- M-step: Use the posterior probabilities $q(h)$ as cluster specific weights on data points e to determine θ through maximum likelihood.

4 Logic-based models

4.1 Basics

❒ **Syntax of propositional logic** – By noting f,g formulas, and $\neg, \wedge, \vee, \to, \leftrightarrow$ connectives, we can write the following logical expressions:

Name	Symbol	Meaning	Illustration
Affirmation	f	f	
Negation	$\neg f$	not f	
Conjunction	$f \wedge g$	f and g	
Disjunction	$f \vee g$	f or g	
Implication	$f \to g$	if f then g	
Biconditional	$f \leftrightarrow g$	f, that is to say g	

Remark: formulas can be built up recursively out of these connectives.

❒ **Model** – A model w denotes an assignment of binary weights to propositional symbols.

Example: the set of truth values $w = \{A : 0, B : 1, C : 0\}$ is one possible model to the propositional symbols A, B and C.

❒ **Interpretation function** – The interpretation function $\mathcal{I}(f,w)$ outputs whether model w satisfies formula f:

$$\boxed{\mathcal{I}(f,w) \in \{0,1\}}$$

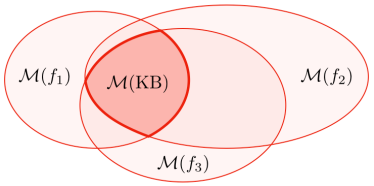
❒ **Set of models** – $\mathcal{M}(f)$ denotes the set of models w that satisfy formula f. Mathematically speaking, we define it as follows:

$$\boxed{\forall w \in \mathcal{M}(f), \quad \mathcal{I}(f,w) = 1}$$

4.2 Knowledge base

❒ **Definition** – The knowledge base KB is the conjunction of all formulas that have been considered so far. The set of models of the knowledge base is the intersection of the set of models that satisfy each formula. In other words:

$$\boxed{\mathcal{M}(\text{KB}) = \bigcap_{f \in \text{KB}} \mathcal{M}(f)}$$



❒ **Probabilistic interpretation** – The probability that query f is evaluated to 1 can be seen as the proportion of models w of the knowledge base KB that satisfy f, i.e.:

$$\boxed{P(f|\text{KB}) = \frac{\displaystyle\sum_{w \in \mathcal{M}(\text{KB}) \cap \mathcal{M}(f)} P(W = w)}{\displaystyle\sum_{w \in \mathcal{M}(\text{KB})} P(W = w)}}$$

❒ **Satisfiability** – The knowledge base KB is said to be satisfiable if at least one model w satisfies all its constraints. In other words:

$$\boxed{\text{KB satisfiable} \iff \mathcal{M}(\text{KB}) \neq \varnothing}$$

Remark: $\mathcal{M}(KB)$ denotes the set of models compatible with all the constraints of the knowledge base.

❒ **Relation between formulas and knowledge base** – We define the following properties between the knowledge base KB and a new formula f:

Name	Mathematical formulation	Illustration	Notes
KB entails f	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \mathcal{M}(\text{KB})$		- f does not bring any new information - Also written KB $\models f$
KB contradicts f	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \varnothing$		- No model satisfies the constraints after adding f Equivalent to KB $\models \neg f$
f contingent to KB	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \varnothing$ and $\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \mathcal{M}(\text{KB})$		- f does not contradict KB - f adds a non-trivial amount of information to KB

❒ **Model checking** – A model checking algorithm takes as input a knowledge base KB and outputs whether it is satisfiable or not.

Remark: popular model checking algorithms include DPLL and WalkSat.

❒ **Inference rule** – An inference rule of premises $f_1,...,f_k$ and conclusion g is written:

$$\boxed{\frac{f_1,...,f_k}{g}}$$

❒ **Forward inference algorithm** – From a set of inference rules Rules, this algorithm goes through all possible $f_1,...,f_k$ and adds g to the knowledge base KB if a matching rule exists. This process is repeated until no more additions can be made to KB.

❒ **Derivation** – We say that KB derives f (written KB $\vdash f$) with rules Rules if f already is in KB or gets added during the forward inference algorithm using the set of rules Rules.

❒ **Properties of inference rules** – A set of inference rules Rules can have the following properties:

Name	Mathematical formulation	Notes
Soundness	$\{f : \text{KB} \vdash f\} \subseteq \{f : \text{KB} \models f\}$	- Inferred formulas are entailed by KB - Can be checked one rule at a time - "*Nothing but the truth*"
Completeness	$\{f : \text{KB} \vdash f\} \supseteq \{f : \text{KB} \models f\}$	- Formulas entailing KB are either already in the knowledge base or inferred from it - "*The whole truth*"

4.3 Propositional logic

In this section, we will go through logic-based models that use logical formulas and inference rules. The idea here is to balance expressivity and computational efficiency.

❒ **Horn clause** – By noting p_1,...,p_k and q propositional symbols, a Horn clause has the form:

$$\boxed{(p_1 \wedge ... \wedge p_k) \longrightarrow q}$$

Remark: when $q = false$, it is called a "goal clause", otherwise we denote it as a "definite clause".

❒ **Modus ponens inference rule** – For propositional symbols f_1,...,f_k and p, the modus ponens rule is written:

$$\boxed{\frac{f_1,...,f_k, \quad (f_1 \wedge ... \wedge f_k) \longrightarrow p}{p}}$$

Remark: it takes linear time to apply this rule, as each application generate a clause that contains a single propositional symbol.

❒ **Completeness** – Modus ponens is complete with respect to Horn clauses if we suppose that KB contains only Horn clauses and p is an entailed propositional symbol. Applying modus ponens will then derive p.

❒ **Conjunctive normal form** – A conjunctive normal form (CNF) formula is a conjunction of clauses, where each clause is a disjunction of atomic formulas.

Remark: in other words, CNFs are $\wedge$ of $\vee$.

❒ **Equivalent representation** – Every formula in propositional logic can be written into an equivalent CNF formula. The table below presents general conversion properties:

Rule name		Initial	Converted
Eliminate	$\leftrightarrow$	$f \leftrightarrow g$	$(f \rightarrow g) \wedge (g \rightarrow f)$
	$\rightarrow$	$f \rightarrow g$	$\neg f \vee g$
	$\neg\neg$	$\neg\neg f$	f
Distribute	$\neg$ over $\wedge$	$\neg(f \wedge g)$	$\neg f \vee \neg g$
	$\neg$ over $\vee$	$\neg(f \vee g)$	$\neg f \wedge \neg g$
	$\vee$ over $\wedge$	$f \vee (g \wedge h)$	$(f \vee g) \wedge (f \vee h)$

❒ **Resolution inference rule** – For propositional symbols f_1,...,f_n, and g_1,...,g_m as well as p, the resolution rule is written:

$$\boxed{\frac{f_1 \vee ... \vee f_n \vee p, \quad \neg p \vee g_1 \vee ... \vee g_m}{f_1 \vee ... \vee f_n \vee g_1 \vee ... \vee g_m}}$$

Remark: it can take exponential time to apply this rule, as each application generates a clause that has a subset of the propositional symbols.

❒ **Resolution-based inference** – The resolution-based inference algorithm follows the following steps:

- Step 1: Convert all formulas into CNF
- Step 2: Repeatedly apply resolution rule
- Step 3: Return unsatisfiable if and only if False is derived

4.4 First-order logic

The idea here is that variables yield compact knowledge representations.

❒ **Model** – A model w in first-order logic maps:

- constant symbols to objects
- predicate symbols to tuple of objects

❒ **Horn clause** – By noting x_1,...,x_n variables and a_1,...,a_k,b atomic formulas, the first-order logic version of a horn clause has the form:

$$\boxed{\forall x_1,...,\forall x_n, \quad (a_1 \wedge ... \wedge a_k) \rightarrow b}$$

❒ **Substitution** – A substitution θ maps variables to terms and Subst(θ,f) denotes the result of substitution θ on f.

❒ **Unification** – Unification takes two formulas f and g and returns the most general substitution θ that makes them equal:

$$\boxed{\text{Unify}[f,g] = \theta} \quad \text{s.t.} \quad \boxed{\text{Subst}[\theta,f] = \text{Subst}[\theta,g]}$$

Note: Unify[f,g] returns Fail if no such θ exists.

❒ **Modus ponens** – By noting x_1,...,x_n variables, a_1,...,a_k and a'_1,...,a'_k atomic formulas and by calling $\theta = \text{Unify}(a'_1 \wedge ... \wedge a'_k, a_1 \wedge ... \wedge a_k)$ the first-order logic version of modus ponens can be written:

$$\boxed{\frac{a'_1,...,a'_k \quad \forall x_1,...,\forall x_n (a_1 \wedge ... \wedge a_k) \rightarrow b}{\text{Subst}[\theta, b]}}$$

❒ **Completeness** – Modus ponens is complete for first-order logic with only Horn clauses.

❒ **Resolution rule** – By noting $f_1, ..., f_n, g_1, ..., g_m, p, q$ formulas and by calling $\theta = \text{Unify}(p,q)$, the first-order logic version of the resolution rule can be written:

$$\boxed{\frac{f_1 \vee ... \vee f_n \vee p, \quad \neg q \vee g_1 \vee ... \vee g_m}{\text{Subst}[\theta, f_1 \vee ... \vee f_n \vee g_1 \vee ... \vee g_m]}}$$

❒ **Semi-decidability** – First-order logic, even restricted to only Horn clauses, is semi-decidable.

- if KB $\models f$, forward inference on complete inference rules will prove f in finite time
- if KB $\not\models f$, no algorithm can show this in finite time

Machine Learning Interview Cheat sheets

Aqeel Anwar

Last Updated: March 2021

This document contains cheat sheets on various topics asked during a Machine Learning/Data science interview. This document is constantly updated to include more topics.

Click here to get the updated version

Table of Contents

Cheat Sheet – Bias-Variance Tradeoff

What is Bias?

$$bias = \mathbb{E}[f'(x)] - f(x)$$

- Error between average model prediction and ground truth
- The bias of the estimated function tells us the capacity of the underlying model to predict the values

What is Variance?

$$variance = \mathbb{E}\left[\left(f'(x) - \mathbb{E}[f'(x)]\right)^2\right]$$

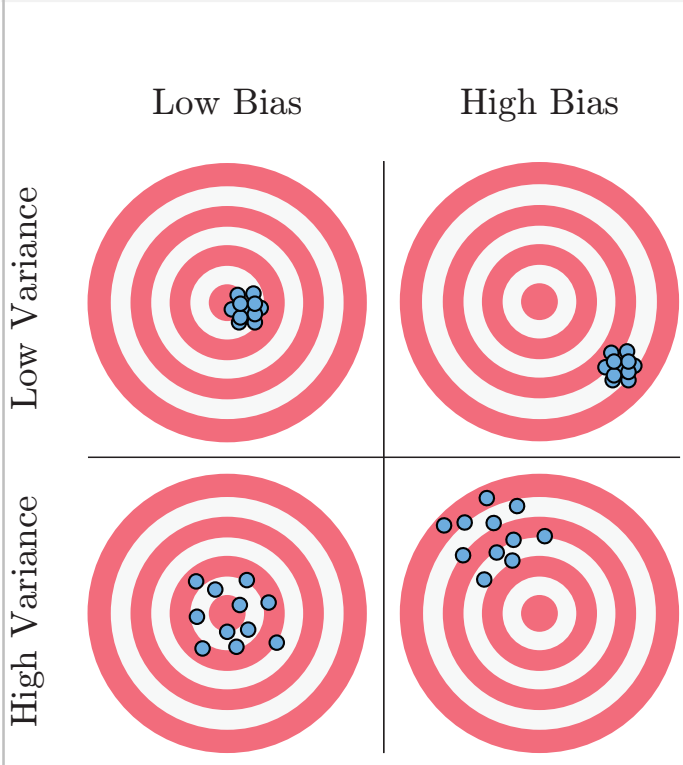
- Average variability in the model prediction for the given dataset
- The variance of the estimated function tells you how much the function can adjust to the change in the dataset

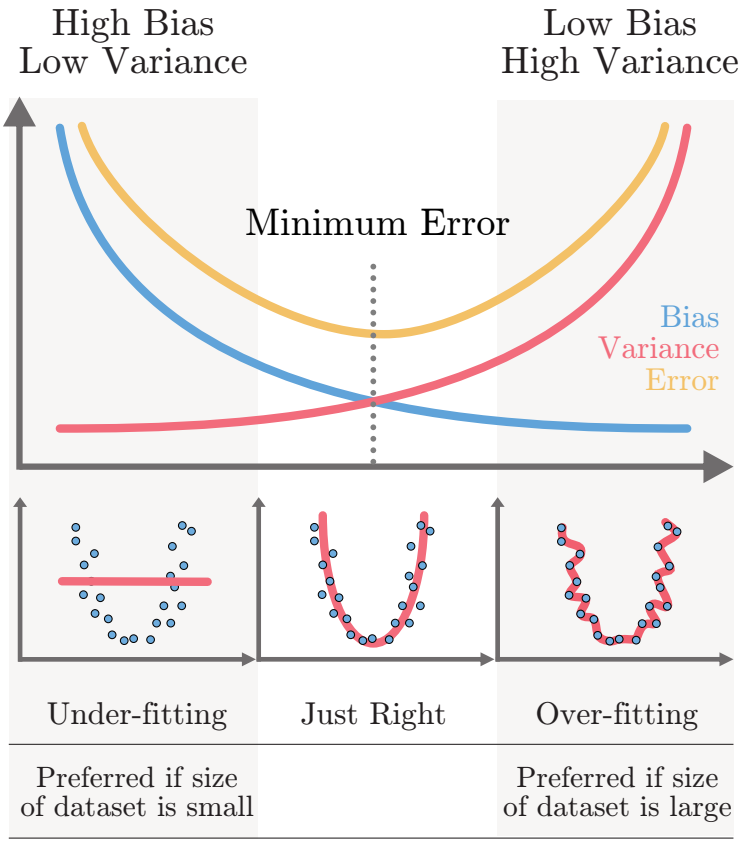
High Bias

→ Overly-simplified Model
→ Under-fitting
→ High error on both test and train data

High Variance

→ Overly-complex Model
→ Over-fitting
→ Low error on train data and high on test
→ Starts modelling the noise in the input





Bias variance Trade-off

- Increasing bias reduces variance and vice-versa
- Error = bias^2 + variance +irreducible error
- The best model is where the error is reduced.
- Compromise between bias and variance

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Imbalanced Data in Classification

Classifier that always predicts label blue yields prediction accuracy of 90%

Accuracy doesn't always give the correct insight about your trained model

Accuracy: %age correct prediction	Correct prediction over total predictions	One value for entire network
Precision: Exactness of model	From the detected cats, how many were actually cats	Each class/label has a value
Recall: Completeness of model	Correctly detected cats over total cats	Each class/label has a value
F1 Score: Combines Precision/Recall	Harmonic mean of Precision and Recall	Each class/label has a value

Performance metrics associated with Class 1

Predicted Labels \ Actual Labels	1	0
1	True Positive	False Positive
0	False Negative	True Negative

(Is your prediction correct?) (What did you predict)

True Negative

(Your prediction is correct) (You predicted 0)

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F1 score} = 2x\frac{(\text{Prec} \times \text{Rec})}{(\text{Prec} + \text{Rec})}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

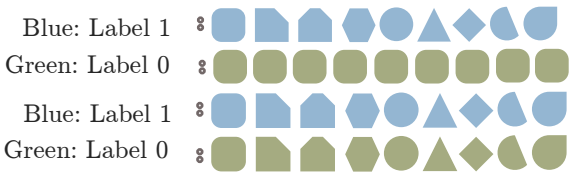
$$\text{False +ve rate} = \frac{FP}{TN + FP}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{Recall, Sensitivity} = \text{True +ve rate} = \frac{TP}{TP + FN}$$

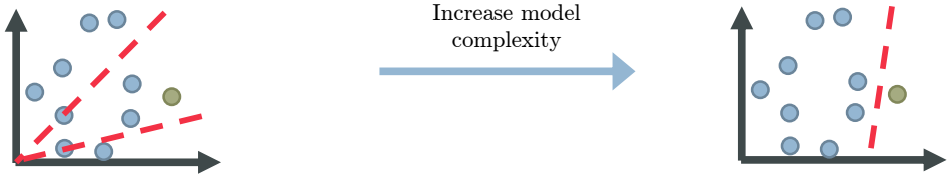
Possible solutions

1. **Data Replication:** Replicate the available data until the number of samples are comparable
2. **Synthetic Data:** Images: Rotate, dilate, crop, add noise to existing input images and create new data
3. **Modified Loss:** Modify the loss to reflect greater error when misclassifying smaller sample set



$$loss = a * loss_{green} + b * loss_{blue} \qquad a > b$$

4. **Change the algorithm:** Increase the model/algorithm complexity so that the two classes are perfectly separable (Con: Overfitting)



No straight line (y=ax) passing through origin can perfectly separate data. **Best solution:** line y=0, predict all labels blue

Straight line (y=ax+**b**) can perfectly separate data. Green class will no longer be predicted as blue

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – PCA Dimensionality Reduction

What is PCA?

- Based on the dataset find a new set of orthogonal feature vectors in such a way that the data spread is maximum in the direction of the feature vector (or dimension)
- Rates the feature vector in the decreasing order of data spread (or variance)
- The datapoints have maximum variance in the first feature vector, and minimum variance in the last feature vector
- The variance of the datapoints in the direction of feature vector can be termed as a measure of information in that direction.

Steps

1. Standardize the datapoints
2. Find the covariance matrix from the given datapoints
3. Carry out eigen-value decomposition of the covariance matrix
4. Sort the eigenvalues and eigenvectors

$$X_{new} = \frac{X - mean(X)}{std(X)}$$

$$C[i,j] = cov(x_i, x_j)$$

$$C = V\Sigma V^{-1}$$

$$\Sigma_{sort} = sort(\Sigma) \quad V_{sort} = sort(V, \Sigma_{sort})$$

Dimensionality Reduction with PCA

- Keep the first m out of n feature vectors rated by PCA. These m vectors will be the best m vectors preserving the maximum information that could have been preserved with m vectors on the given dataset

Steps:

1. Carry out steps 1-4 from above
2. Keep first m feature vectors from the sorted eigenvector matrix
3. Transform the data for the new basis (feature vectors)
4. The importance of the feature vector is proportional to the magnitude of the eigen value

$$V_{reduced} = V[:, 0:m]$$

$$X_{reduced} = X_{new} \times V_{reduced}$$

Figure 1

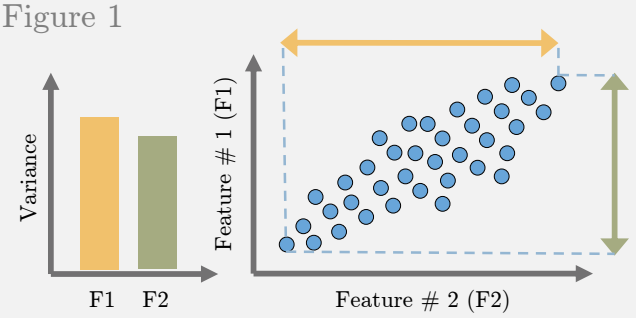


Figure 2

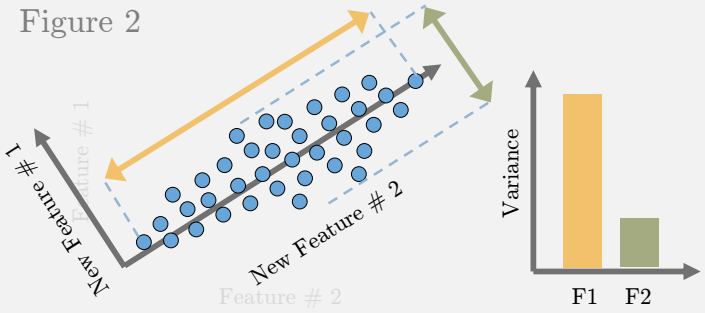


Figure 3

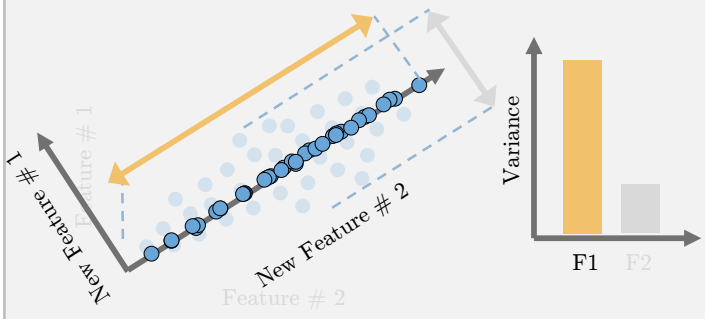


Figure 1: Datapoints with feature vectors as x and y-axis

Figure 2: The cartesian coordinate system is rotated to maximize the standard deviation along any one axis (new feature # 2)

Figure 3: Remove the feature vector with minimum standard deviation of datapoints (new feature # 1) and project the data on new feature # 2

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Bayes Theorem and Classifier

What is Bayes' Theorem?

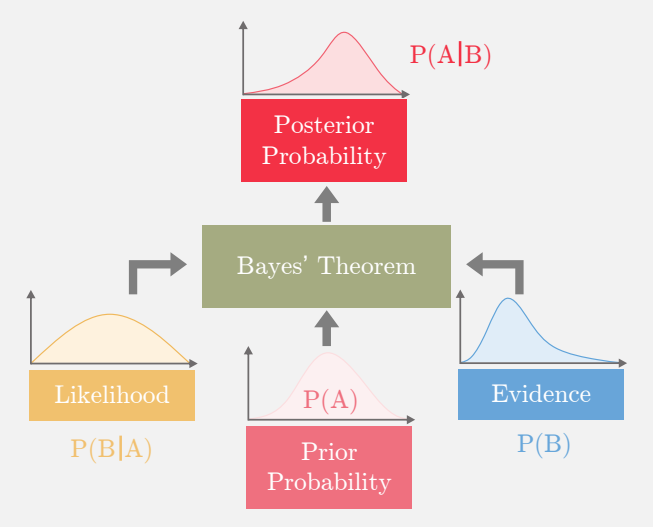
- Describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(A|B) = \frac{P(B|A)(likelihood) \times P(A)(prior)}{P(B)(prior)}$$

- How the probability of an event changes when we have knowledge of another event

$$P(A) \longrightarrow P(A|B)$$

Usually a better estimate than P(A)



Example

- Probability of fire P(F) = 1%
- Probability of smoke P(S) = 10%
- Prob of smoke given there is a fire P(S|F) = 90%
- What is the probability that there is a fire given we see a smoke P(F|S)?

$$P(F|S) = \frac{P(S|F) \times P(F)}{P(S)} = \frac{0.9 \times 0.01}{0.1} = 9\%$$

Maximum Aposteriori Probability (MAP) Estimation

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We try to accommodate our prior knowledge when estimating.

$$\hat{y}_{MAP} = argmax_y \;\; P(y) \prod_i P(x_i|y)$$

y that maximizes the product of prior and likelihood

Maximum Likelihood Estimation (MLE)

The MAP estimate of the random variable y, given that we have observed iid (x_1, x_2, x_3, …), is given by. We assume we don't have any prior knowledge of the quantity being estimated.

$$\hat{y}_{MLE} = argmax_y \prod_i P(x_i|y)$$

y that maximizes only the likelihood

MLE is a special case of MAP where our prior is uniform (all values are equally likely)

Naïve Bayes' Classifier (Instantiation of MAP as classifier)

Suppose we have two classes, $y=y_1$ and $y=y_2$. Say we have more than one evidence/features (x_1, x_2, x_3, …), using Bayes' theorem

$$P(y|x_1, x_2, x_3, \ldots) = \frac{P(x_1, x_2, x_3, \ldots|y) \times P(y)}{P(x_1, x_2, x_3, \ldots)}$$

Bayes' theorem assumes the features (x_1, x_2, x_3, …) are i.i.d. i.e $P(x_1, x_2, x_3, \ldots|y) = \prod_i P(x_i|y)$

$$P(y|x_1, x_2, x_3, \ldots) = \prod_i P(x_i|y) \frac{P(y)}{P(x_1, x_2, x_3, \ldots)}$$

$$\hat{y} = y_1 \; if \; \frac{P(y_1|x_1, x_2, x_3, \ldots)}{P(y_2|x_1, x_2, x_3, \ldots)} > 1 \;\; else \; \hat{y} = y_2$$

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Regression Analysis

What is Regression Analysis?
Fitting a function f(.) to datapoints y_i=f(x_i) under some error function. Based on the estimated function and error, we have the following types of regression

1. Linear Regression:
Fits a line minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_i \|y_i - f_\beta^{linear}(x_i)\|^2$$
$$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$$

2. Polynomial Regression:
Fits a polynomial of order k (k+1 unknowns) minimizing the sum of mean-squared error for each datapoint.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta^{poly}(x_i)\|^2$$
$$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_k x_i^k$$

3. Bayesian Regression:
For each datapoint, fits a gaussian distribution by minimizing the mean-squared error. As the number of data points x_i increases, it converges to point estimates i.e. $n \to \infty, \sigma^2 \to 0$

$$min_\beta \sum_i \|y_i - \mathcal{N}\left(f_\beta(x_i), \sigma^2\right)\|^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\mathcal{N}\left(\mu, \sigma^2\right) \to \text{Gaussian with mean } \mu \text{ and variance } \sigma^2$$

4. Ridge Regression:
Can fit either a line, or polynomial minimizing the sum of mean-squared error for each datapoint and the weighted L2 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} \beta_j^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

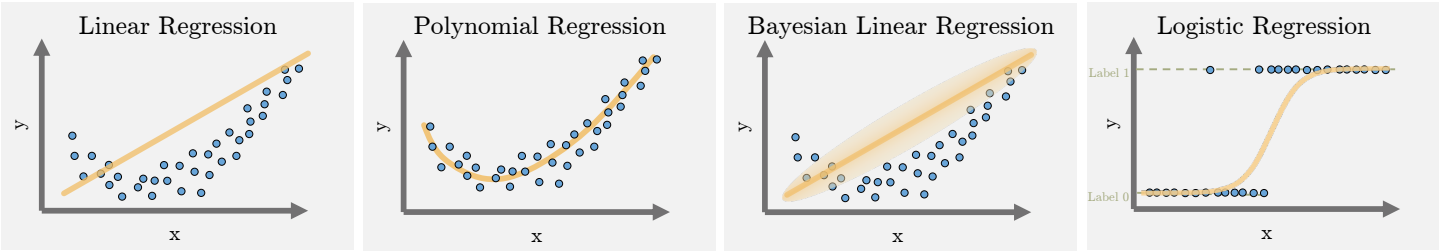
5. LASSO Regression:
Can fit either a line, or polynomial minimizing the the sum of mean-squared error for each datapoint and the weighted L1 norm of the function parameters beta.

$$min_\beta \sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{k} |\beta_j|$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$

6. Logistic Regression (NOT regression, but classification):
Can fit either a line, or polynomial with sigmoid activation minimizing the sum of mean-squared error for each datapoint. The labels y are binary class labels.

$$min_\beta \sum_i \|y_i - \sigma(f_\beta(x_i))\|^2$$
$$f_\beta(x_i) = f_\beta^{poly}(x_i) \text{ or } f_\beta^{linear}(x_i)$$
$$\sigma(t) = \frac{1}{1+e^{-t}}$$

Visual Representation:



Summary:

	What does it fit?	Estimated function	Error Function
Linear	A line in n dimensions	$f_\beta^{linear}(x_i) = \beta_0 + \beta_1 x_i$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Polynomial	A polynomial of order k	$f_\beta^{poly}(x_i) = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$
Bayesian Linear	Gaussian distribution for each point	$\mathcal{N}\left(f_\beta(x_i), \sigma^2\right)$	$\sum_i \|y_i - \mathcal{N}\left(f_\beta(x_i), \sigma^2\right)\|^2$
Ridge	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{n} \beta_j^2$
LASSO	Linear/polynomial	$f_\beta^{poly}(x_i)$ or $f_\beta^{linear}(x_i)$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2 + \sum_{j=0}^{n} \|\beta_j\|$
Logistic	Linear/polynomial with sigmoid	$\sigma(f_\beta(x_i))$	$\sum_{i=0}^{m} \|y_i - f_\beta(x_i)\|^2$

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Regularization in ML

What is Regularization in ML?

- Regularization is an approach to address over-fitting in ML.
- Overfitted model fails to generalize estimations on test data
- When the underlying model to be learned is low bias/high variance, or when we have small amount of data, the estimated model is prone to over-fitting.
- Regularization reduces the variance of the model

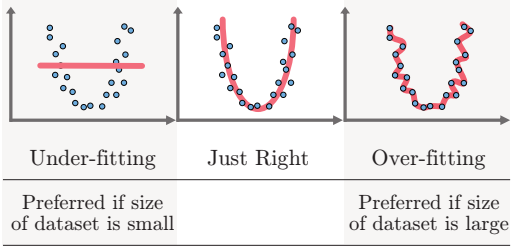


Figure 1. Overfitting

Types of Regularization:

1. Modify the loss function:

- **L2 Regularization:** Prevents the weights from getting too large (defined by L2 norm). Larger the weights, more complex the model is, more chances of overfitting.

$$loss = error(y, \hat{y}) + \lambda \sum_j \beta_j^2 \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **L1 Regularization:** Prevents the weights from getting too large (defined by L1 norm). Larger the weights, more complex the model is, more chances of overfitting. L1 regularization introduces sparsity in the weights. It forces more weights to be zero, than reducing the the average magnitude of all weights

$$loss = error(y, \hat{y}) + \lambda \sum_j |\beta_j| \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

- **Entropy:** Used for the models that output probability. Forces the probability distribution towards uniform distribution.

$$loss = error(p, \hat{p}) - \lambda \sum_i \hat{p}_i log(\hat{p}_i) \quad \lambda \geq 0,\ \lambda \propto model\ bias,\quad \lambda \propto \frac{1}{model\ variance}$$

2. Modify data sampling:

- **Data augmentation:** Create more data from available data by randomly cropping, dilating, rotating, adding small amount of noise etc.
- **K-fold Cross-validation:** Divide the data into k groups. Train on (k-1) groups and test on 1 group. Try all k possible combinations.

3. Change training approach:

- **Injecting noise:** Add random noise to the weights when they are being learned. It pushes the model to be relatively insensitive to small variations in the weights, hence regularization
- **Dropout:** Generally used for neural networks. Connections between consecutive layers are randomly dropped based on a dropout-ratio and the remaining network is trained in the current iteration. In the next iteration, another set of random connections are dropped.

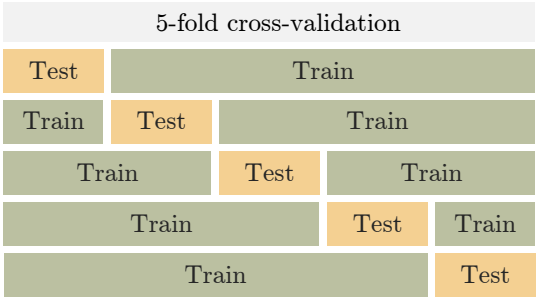


Figure 2. K-fold CV

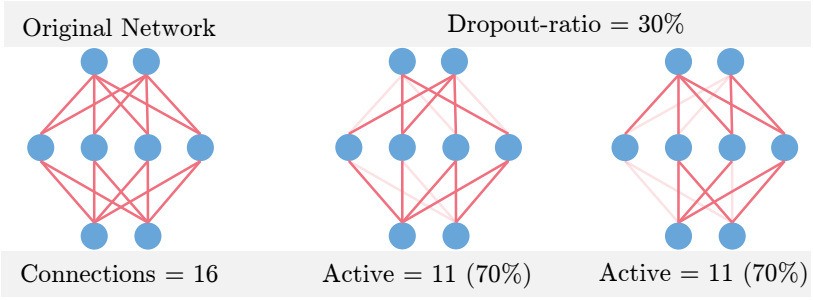


Figure 3. Drop-out

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Famous CNNs

AlexNet – 2012

Why: AlexNet was born out of the need to improve the results of the ImageNet challenge.

What: The network consists of 5 Convolutional (CONV) layers and 3 Fully Connected (FC) layers. The activation used is the Rectified Linear Unit (ReLU).

How: Data augmentation is carried out to reduce over-fitting, Uses Local response localization.

AlexNet Network - Structural Details

Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							62,378,344

VGGNet – 2014

Why: VGGNet was born out of the need to reduce the # of parameters in the CONV layers and improve on training time

What: There are multiple variants of VGGNet (VGG16, VGG19, etc.)

How: The important point to note here is that all the conv kernels are of size 3x3 and maxpool kernels are of size 2x2 with a stride of two.

VGG16 - Structural Details

#	Input Image			output			Layer	Stride	Kernel		in	out	Param
1	224	224	3	224	224	64	conv3-64	1	3	3	3	64	1792
2	224	224	64	224	224	64	conv3064	1	3	3	64	64	36928
	224	224	64	112	112	64	maxpool	2	2	2	64	64	0
3	112	112	64	112	112	128	conv3-128	1	3	3	64	128	73856
4	112	112	128	112	112	128	conv3-128	1	3	3	128	128	147584
	112	112	128	56	56	128	maxpool	2	2	2	128	128	65664
5	56	56	128	56	56	256	conv3-256	1	3	3	128	256	295168
6	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
7	56	56	256	56	56	256	conv3-256	1	3	3	256	256	590080
	56	56	256	28	28	256	maxpool	2	2	2	256	256	0
8	28	28	256	28	28	512	conv3-512	1	3	3	256	512	1180160
9	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
10	28	28	512	28	28	512	conv3-512	1	3	3	512	512	2359808
	28	28	512	14	14	512	maxpool	2	2	2	512	512	0
11	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
12	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
13	14	14	512	14	14	512	conv3-512	1	3	3	512	512	2359808
	14	14	512	7	7	512	maxpool	2	2	2	512	512	0
14	1	1	25088	1	1	4096	fc		1	1	25088	4096	102764544
15	1	1	4096	1	1	4096	fc		1	1	4096	4096	16781312
16	1	1	4096	1	1	1000	fc		1	1	4096	1000	4097000
						Total							138,423,208

ResNet – 2015

Why: Neural Networks are notorious for not being able to find a simpler mapping when it exists. ResNet solves that.

What: There are multiple versions of ResNetXX architectures where 'XX' denotes the number of layers. The most used ones are ResNet50 and ResNet101. Since the vanishing gradient problem was taken care of (more about it in the How part), CNN started to get deeper and deeper

How: ResNet architecture makes use of shortcut connections do solve the vanishing gradient problem. The basic building block of ResNet is a Residual block that is repeated throughout the network.

ResNet18 - Structural Details

#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
						Total								11,511,784

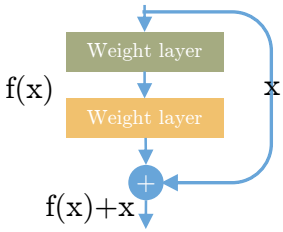


Figure 1 ResNet Block

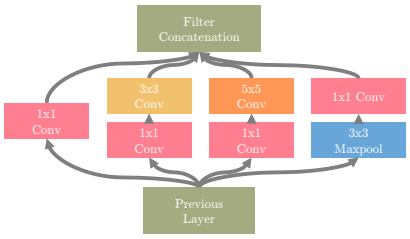


Figure 2 Inception Block

Inception – 2014

Why: Lager kernels are preferred for more global features, on the other hand, smaller kernels provide good results in detecting area-specific features. For effective recognition of such a variable-sized feature, we need kernels of different sizes. That is what Inception does.

What: The Inception network architecture consists of several inception modules of the following structure. Each inception module consists of four operations in parallel, 1x1 conv layer, 3x3 conv layer, 5x5 conv layer, max pooling

How: Inception increases the network space from which the best network is to be chosen via training. Each inception module can capture salient features at different levels.

GoogLeNet - Structural Details

Input Image			output			Layer	Input Layer	Stride	Pad	Kernel		in	out	Param
227	227	3	112	112	64	conv1	input	2	1	7	7	3	64	9472
112	112	64	56	56	64	maxpool1	conv1	2	0.5	3	3	64	64	0
56	56	64	56	56	64	conv1x1	maxpool1	1	0	1	1	64	64	4160
56	56	64	56	56	192	conv2-1		1	1	3	3	64	192	110784
56	56	192	28	28	192	maxpool2		2	0.5	3	3	192	192	0
28	28	192	28	28	96	conv1x1a	maxpool2	1	0	1	1	192	96	18528
28	28	96	28	28	16	conv1x1b	maxpool2	1	0	1	1	192	16	3088
28	28	192	28	28	192	maxpool-a	maxpool2	1	1	3	3	192	192	0
28	28	192	28	28	64	conv1x1c	maxpool2	1	0	1	1	192	64	12352
28	28	96	28	28	128	conv3-3	conv1x1a	1	1	3	3	96	128	110720
28	28	16	28	28	32	conv5x5	conv1x1b	1	2	5	5	16	32	12832
28	28	192	28	28	32	conv1x1d	maxpool-a	1	0	1	1	192	32	6176
			28	28	256	depth-concat	conv1x1c, conv3x3, conv5x5, conv1x1d							
28	28	256	28	28	128	conv1x1a	depth-concat	1	0	1	1	256	128	32896
28	28	128	28	28	32	conv1x1b	depth-concat	1	0	1	1	256	32	8224
28	28	192	28	28	256	maxpool-a	depth-concat	1	1	3	3	256	256	0
28	28	192	28	28	128	conv1x1c	depth-concat	1	0	1	1	256	128	32896
28	28	96	28	28	192	conv3-3	conv1x1a	1	1	3	3	128	192	221376
28	28	16	28	28	96	conv5x5	conv1x1b	1	2	5	5	32	96	76896
28	28	192	28	28	64	conv1x1d	maxpool-a	1	0	1	1	256	64	16448
			28	28	480	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
28	28	480	14	14	480	maxpool3	depth-concat	2	0.5	3	3	480	480	0
14	14	480	14	14	96	conv1x1a	maxpool3	1	0	1	1	480	96	46176
14	14	480	14	14	16	conv1x1b	maxpool3	1	0	1	1	480	16	7696
14	14	480	14	14	480	maxpool-a	maxpool3	1	1	3	3	480	480	0
14	14	480	14	14	192	conv1x1c	maxpool3	1	0	1	1	480	192	92352
14	14	96	14	14	208	conv3-3	conv1x1a	1	1	3	3	96	208	179920
14	14	16	14	14	48	conv5x5	conv1x1b	1	2	5	5	16	48	19248
14	14	192	14	14	64	conv1x1d	maxpool-a	1	0	1	1	480	64	30784
			14	14	512	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
14	14	512	14	14	112	conv1x1a	depth-concat	1	0	1	1	512	112	57456
14	14	512	14	14	24	conv1x1b	depth-concat	1	0	1	1	64	24	1560
14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
14	14	512	14	14	160	conv1x1c	depth-concat	1	0	1	1	64	160	10400
14	14	96	14	14	224	conv3-3	conv1x1a	1	1	3	3	112	224	226016
14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	24	64	38464
14	14	160	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
			14	14	512	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
14	14	512	14	14	128	conv1x1a	depth-concat	1	0	1	1	512	128	65664
14	14	512	14	14	24	conv1x1b	depth-concat	1	0	1	1	64	24	1560
14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
14	14	512	14	14	128	conv1x1c	depth-concat	1	0	1	1	64	128	8320
14	14	96	14	14	256	conv3-3	conv1x1a	1	1	3	3	128	256	295168
14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	24	64	38464
14	14	128	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
			14	14	512	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
14	14	512	14	14	144	conv1x1a	depth-concat	1	0	1	1	512	144	73872
14	14	512	14	14	32	conv1x1b	depth-concat	1	0	1	1	64	32	2080
14	14	512	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
14	14	512	14	14	112	conv1x1c	depth-concat	1	0	1	1	64	112	7280
14	14	96	14	14	288	conv3-3	conv1x1a	1	1	3	3	144	288	373536
14	14	16	14	14	64	conv5x5	conv1x1b	1	2	5	5	32	64	51264
14	14	112	14	14	64	conv1x1d	maxpool-a	1	0	1	1	64	64	4160
			14	14	528	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
14	14	528	14	14	160	conv1x1a	depth-concat	1	0	1	1	528	160	84640
14	14	528	14	14	32	conv1x1b	depth-concat	1	0	1	1	64	32	2080
14	14	528	14	14	64	maxpool-a	depth-concat	1	1	3	3	64	64	0
14	14	528	14	14	256	conv1x1c	depth-concat	1	0	1	1	64	256	16640
14	14	96	14	14	320	conv3-3	conv1x1a	1	1	3	3	160	320	461120
14	14	16	14	14	128	conv5x5	conv1x1b	1	2	5	5	32	128	102528
14	14	256	14	14	128	conv1x1d	maxpool-a	1	0	1	1	64	128	8320
			14	14	832	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
14	14	832	7	7	832	maxpool4	depth-concat	2	0.5	3	3	832	832	0
7	7	832	7	7	160	conv1x1a	maxpool4	1	0	1	1	832	160	133280
7	7	832	7	7	32	conv1x1b	maxpool4	1	0	1	1	832	32	26656
7	7	832	7	7	832	maxpool-a	maxpool4	1	1	3	3	832	832	0
7	7	832	7	7	256	conv1x1c	maxpool4	1	0	1	1	832	256	213248
7	7	96	7	7	320	conv3-3	conv1x1a	1	1	3	3	160	320	461120
7	7	16	7	7	128	conv5x5	conv1x1b	1	2	5	5	32	128	102528
7	7	256	7	7	128	conv1x1d	maxpool-a	1	0	1	1	832	128	106624
			7	7	832	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
7	7	832	7	7	192	conv1x1a	depth-concat	1	0	1	1	832	192	159936
7	7	832	7	7	48	conv1x1b	depth-concat	1	0	1	1	832	48	39984
7	7	832	7	7	832	maxpool-a	depth-concat	1	1	3	3	832	832	0
7	7	832	7	7	384	conv1x1c	depth-concat	1	0	1	1	832	384	319872
7	7	96	7	7	384	conv3-3	conv1x1a	1	1	3	3	192	384	663936
7	7	16	7	7	128	conv5x5	conv1x1b	1	2	5	5	48	128	153728
7	7	384	7	7	128	conv1x1d	maxpool-a	1	0	1	1	128	128	16512
			7	7	1024	depth-concat	conv1x1a, conv3x3, conv5x5, conv1x1d							
7	7	1024	1	1	1024	avgpool	depth-concat	1	0	7	7	1024	1024	0
1	1	1024	1	1	1000	fc	depth-concat	1	0	1	1	1024	1000	1025000
										Total				6,414,360

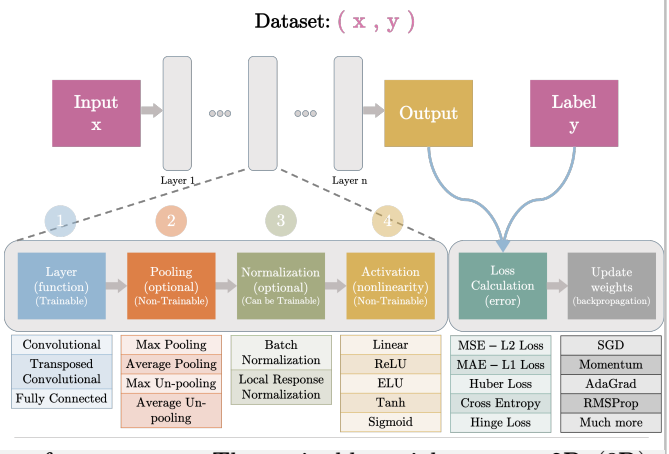
Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Convolutional Neural Network

Convolutional Neural Network:

The data gets into the CNN through the input layer and passes through various hidden layers before getting to the output layer. The output of the network is compared to the actual labels in terms of loss or error. The partial derivatives of this loss w.r.t the trainable weights are calculated, and the weights are updated through one of the various methods using backpropagation.



CNN Template:

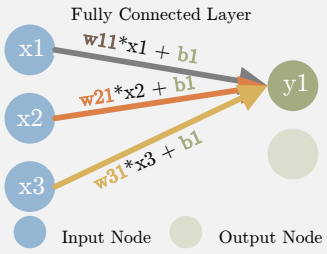
Most of the commonly used hidden layers (not all) follow a pattern

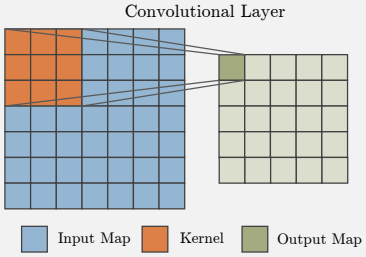
1. **Layer function:** Basic transforming function such as convolutional or fully connected layer.

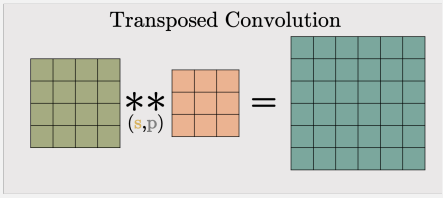
a. **Fully Connected:** Linear functions between the input and the Outputs

a. **Convolutional Layers:** These layers are applied to 2D (3D) input feature maps. The trainable weights are a 2D (3D) kernel/filter that moves across the input feature map, generating dot products with the overlapping region of the input feature map.

b. **Transposed Convolutional (DeConvolutional) Layer:** Usually used to increase the size of the output feature map (Upsampling) The idea behind the transposed convolutional layer is to undo (not exactly) the convolutional layer



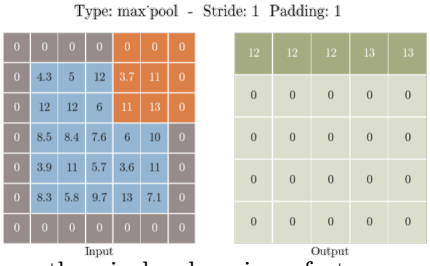




2. **Pooling:** Non-trainable layer to change the size of the feature map

a. **Max/Average Pooling:** Decrease the spatial size of the input layer based on selecting the maximum/average value in receptive field defined by the kernel

b. **UnPooling:** A non-trainable layer used to increase the spatial size of the input layer based on placing the input pixel at a certain index in the receptive field of the output defined by the kernel.



3. **Normalization:** Usually used just before the activation functions to limit the unbounded activation from increasing the output layer values too high

a. **Local Response Normalization LRN:** A **non-trainable layer** that square-normalizes the pixel values in a feature map within a local neighborhood.

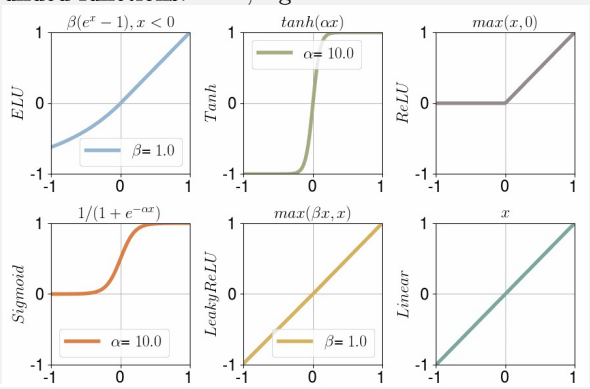
b. **Batch Normalization:** A trainable approach to normalizing the data by learning scale and shift variable during training.

3. **Activation:** Introduce non-linearity so CNN can efficiently map non-linear complex mapping.

a. **Non-parametric/Static functions:** Linear, ReLU

b. **Parametric functions:** ELU, tanh, sigmoid, Leaky ReLU

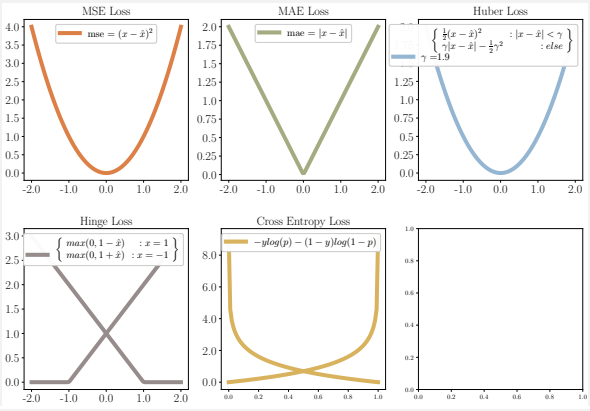
c. **Bounded functions:** tanh, sigmoid



5. **Loss function:** Quantifies how far off the CNN prediction is from the actual labels.

a. Regression Loss Functions: MAE, MSE, Huber loss

b. Classification Loss Functions: Cross entropy, Hinge loss



Source: https://www.cheatsheets.aqeel-anwar.com

Cheat Sheet – Ensemble Learning in ML

What is Ensemble Learning? Wisdom of the crowd

Combine multiple weak models/learners into one predictive model to reduce bias, variance and/or improve accuracy.

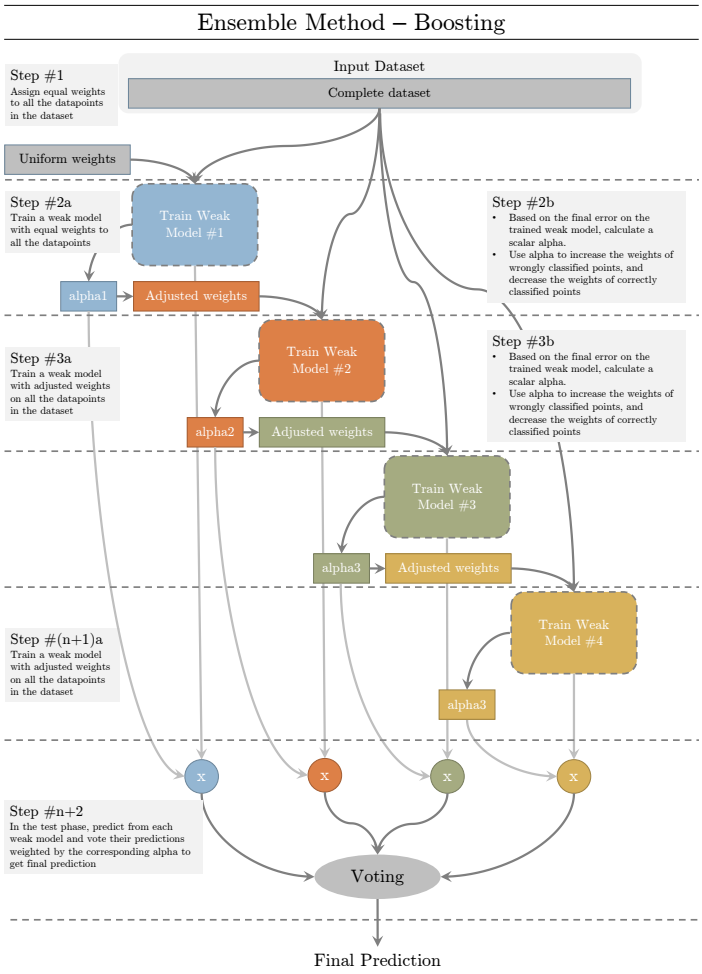
Types of Ensemble Learning: N number of weak learners

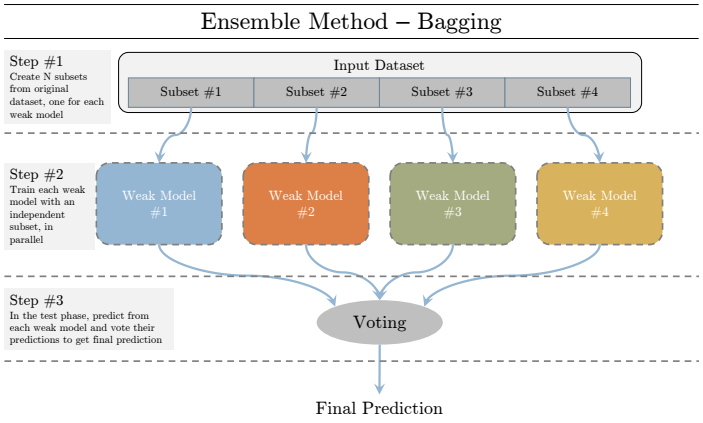
1.Bagging: Trains N different weak models (usually of same types – homogenous) with N non-overlapping subset of the input dataset in parallel. In the test phase, each model is evaluated. The label with the greatest number of predictions is selected as the prediction. Bagging methods reduces variance of the prediction

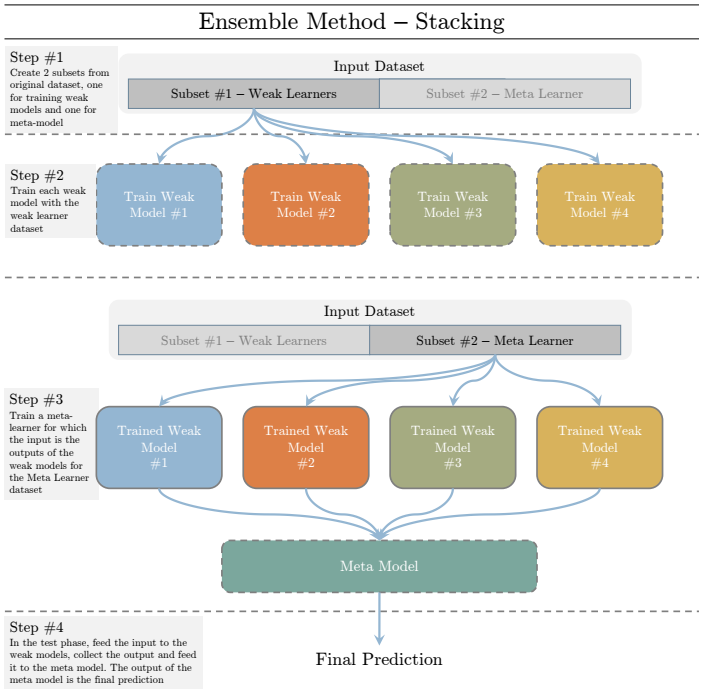
2.Boosting: Trains N different weak models (usually of same types – homogenous) with the complete dataset in a sequential order. The datapoints wrongly classified with previous weak model is provided more weights to that they can be classified by the next weak leaner properly. In the test phase, each model is evaluated and based on the test error of each weak model, the prediction is weighted for voting. Boosting methods decreases the bias of the prediction.

3.Stacking: Trains N different weak models (usually of different types – heterogenous) with one of the two subsets of the dataset in parallel. Once the weak learners are trained, they are used to trained a meta learner to combine their predictions and carry out final prediction using the other subset. In test phase, each model predicts its label, these set of labels are fed to the meta learner which generates the final prediction.

The block diagrams, and comparison table for each of these three methods can be seen below.







Parameter	Bagging	Boosting	Stacking
Focuses on	Reducing variance	Reducing bias	Improving accuracy
Nature of weak learners is	Homogenous	Homogenous	Heterogenous
Weak learners are aggregated by	Simple voting	Weighted voting	Learned voting (meta-learner)

Source: https://www.cheatsheets.aqeel-anwar.com

1/4 How to prepare for behavioral interview?

Collect stories, assign keywords, practice the STAR format

Keywords

List important keywords that will be populated with your personal stories. Most common keywords are given in the table below

Conflict Resolution	Negotiation	Compromise to achieve goal	Creativity	Flexibility	Convincing
Handling Crisis	Challenging Situation	Working with difficult people	Another team priorities not aligned	Adjust to a colleague style	Take Stand
Handling –ve feedback	Coworker view of you	Working with a deadline	Your strength	Your weakness	Influence Others
Handling failure	Handling unexpected situation	Converting challenge to opportunity	Decision without enough data	Conflict Resolution	Mentorship/ Leadership

Stories

1. List all the organizations you have been a part of. For example
    1. Academia: BSc, MSc, PhD
    2. Industry: Jobs, Internship
    3. Societies: Cultural, Technical, Sports
2. Think of stories from step 1 that can fall into one of the keywords categories. The more stories the better. You should have at least 10-15 stories.
3. Create a summary table by assigning multiple keywords to each stories. This will help you filter out the stories when the question asked in the interview. An example can be seen below

Story 1:	[Convincing] [Take Stand] [influence other]
Story 2:	[Mentorship] [Leadership]
Story 3:	[Conflict resolution] [Negotiation]
Story 4:	[decision-without-enough-data]

STAR Format

Write down the stories in the STAR format as explained in the 2/4 part of this cheat sheet. This will help you practice the organization of story in a meaningful way.

Icon Source: www.flaticon.com

2/4 How to prepare for behavioral interview?

Direct*, meaningful*, personalized*, logical*

*(Respective colors are used to identify these characteristics in the example)

Example: "Tell us about a time when you had to convince senior executives"

Situation

Explain the situation and provide necessary context for your story.

S

"I worked as an intern in XYZ company in the summer of 2019. The project details provided to me was elaborative. After some initial brainstorming, and research I realized that the project approach can be modified to make it more efficient in terms of the underlying KPIs. I decided to talk to my manager about it."

Task

Explain the task and your responsibility in the situation

T

"I had an hour-long call with my manager and explained him in detail the proposed approach and how it could improve the KPIs. I was able to convince him. He asked me if I will be able to present my proposed approach for approval in front of the higher executives. I agreed to it. I was working out of the ABC(city) office and the executives need to fly in from XYZ(city) office."

Action

Walk through the steps and actions you took to address the issue

A

"I did a quick background check on the executives to know better about their area of expertise so that I can convince them accordingly. I prepared an elaborative 15 slide presentation starting with explaining their approach, moving onto my proposed approach and finally comparing them on preliminary results.

Result

State the outcome of the result of your actions

R

"After some active discussion we were able to establish that the proposed approach was better than the initial one. The executives proposed a few small changes to my approach and really appreciated my stand. At the end of my internship, I was selected among the 3 out of 68 interns who got to meet the senior vice president of the company over lunch."

Icon Source: www.flaticon.com

Source: https://www.cheatsheets.aqeel-anwar.com

3/4 How to answer a behavioral question?

Understand, Extract, Map, Select and Apply

Example: "Tell us about a time when you had to convince senior executives"

Understand

Understand the question

Example: A story where I was able to convince my seniors. Maybe they had something in mind, and I had a better approach and tried to convince them

Extract

Extract keywords and tags

Extract useful keywords that encapsulates the gist of the question

Example:

[Convincing], [Creative], [Leadership]

Map

Map the keyword to your stories

Shortlist all the stories that fall under the keywords extracted from previous step

Example:

Story1, Story2, Story3, Story4, … , Story N

Select

Select the best story

From the shortlisted stories, pick the one that best describes the question and has not been used so far in the interview

Example: Story3

Apply

Apply the STAR method

Apply the STAR method on the selected story to answer the question

Example: See Cheat Sheet 2/3 for details

Icon Source: www.flaticon.com

Source: https://www.cheatsheets.aqeel-anwar.com

4/4

Behavioral Interview Cheat Sheet

Summarizing the behavioral interview

How to prepare for the interview

1. **Gather important topics as keywords**
   Understand and collect all the important topics commonly asked in the interview
2. **Collect your stories**
   Based on all the organizations you have been a part of, think of all the stories that fall under the keywords above
3. **Practice stories in STAR format**
   Practice each story using the STAR format. You will have to answer the question following this format.
4. **Assign keywords to stories**
   Assign each of your story one or more keywords. This will help you recall them quickly
5. **Create a summary table**
   Create a summary table mapping stories to their associated keywords. This will be used during the behavioral question

How to answer a question during interview

- **U** — **Understand the question**
  Understand the question and clarify any confusions that you have
- **E** — **Extract the keywords**
  Try to extract one or more of the keywords from the question
- **M** — **Map the keywords to stories**
  Based on the keywords extracted, find the stories using the summary table created during preparation (Step 4)
- **S** — **Select a story**
  Since each keyword maybe assigned to multiple stories, select the one that is most relevant and has not been used.
- **A** — **Apply the START format**
  Once the story has been shortlisted, apply STAR format on the story to answer the question.

Icon Source: www.flaticon.com

Source: https://www.cheatsheets.aqeel-anwar.com

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science Interactively at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

matplotlib Cheat sheet

Version 3.5.0

Quick start

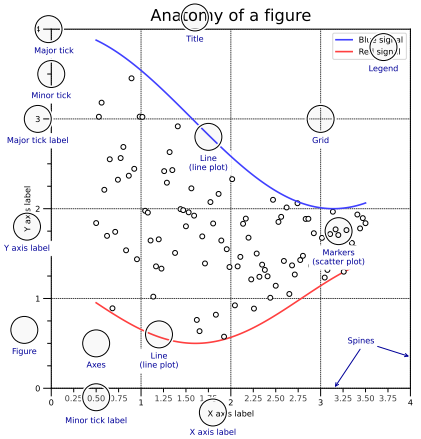
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure



Subplots layout

`subplot[s](rows,cols,…)` API
`fig, axs = plt.subplots(3, 3)`

`G = gridspec(rows,cols,…)` API
`ax = G[0,:]`

`ax.inset_axes(extent)` API

`d=make_axes_locatable(ax)` API
`ax = d.new_horizontal('10%')`

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)` API
X, **Y**, fmt, color, marker, linestyle

`scatter(X,Y,…)` API
X, **Y**, [s]izes, [c]olors, marker, cmap

`bar[h](x,height,…)` API
x, **height**, width, bottom, align, color

`imshow(Z,…)` API
Z, cmap, interpolation, extent, origin

`contour[f]([X],[Y],Z,…)` API
X, Y, **Z**, levels, colors, extent, origin

`pcolormesh([X],[Y],Z,…)` API
X, Y, **Z**, vmin, vmax, cmap

`quiver([X],[Y],U,V,…)` API
X, Y, **U**, **V**, C, units, angles

`pie(X,…)` API
Z, explode, labels, colors, radius

`text(x,y,text,…)` API
x, **y**, **text**, va, ha, size, weight, transform

`fill[_between][x](…)` API
X, Y1, Y2, color, where

Advanced plots

`step(X,Y,[fmt],…)` API
X, **Y**, fmt, color, marker, where

`boxplot(X,…)` API
X, notch, sym, bootstrap, widths

`errorbar(X,Y,xerr,yerr,…)` API
X, **Y**, xerr, yerr, fmt

`hist(X, bins, …)` API
X, bins, range, density, weights

`violinplot(D,…)` API
D, positions, widths, vert

`barbs([X],[Y], U, V, …)` API
X, Y, **U**, **V**, C, length, pivot, sizes

`eventplot(positions,…)` API
positions, orientation, lineoffsets

`hexbin(X,Y,C,…)` API
X, **Y**, C, gridsize, bins

Scales

API

`ax.set_[xy]scale(scale,…)`

linear any values

log values > 0

symlog any values

logit 0 < values < 1

Projections

API

`subplot(…,projection=p)`

p='polar'

p='3d'

`p=Orthographic()` API
`from cartopy.crs import Cartographic`

Lines

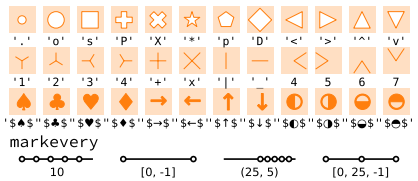
API

`linestyle or ls`

`capstyle or dash_capstyle`

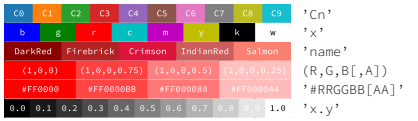
Markers

API



Colors

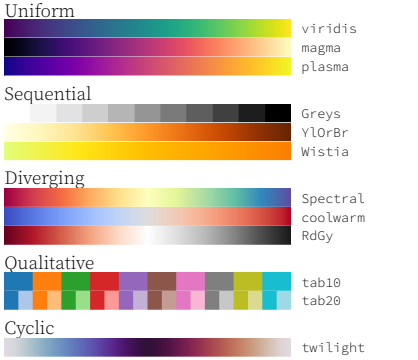
API



Colormaps

API

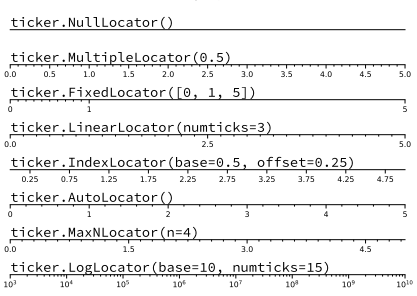
`plt.get_cmap(name)`



Tick locators

API

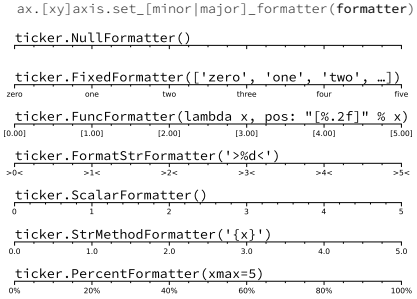
```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



Tick formatters

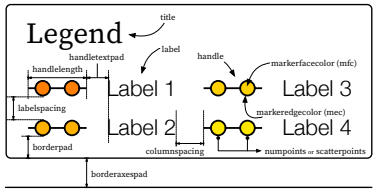
API

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

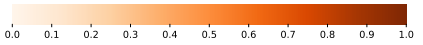


Ornaments

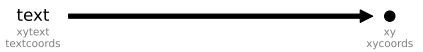
`ax.legend(…)` API
handles, labels, loc, title, frameon



`ax.colorbar(…)` API
mappable, ax, cax, orientation



`ax.annotate(…)` API
text, **xy**, **xytext**, xycoords, textcoords, arrowprops



Event handling

API

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

API

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

API

`plt.style.use(style)`

Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

API

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten simple rules

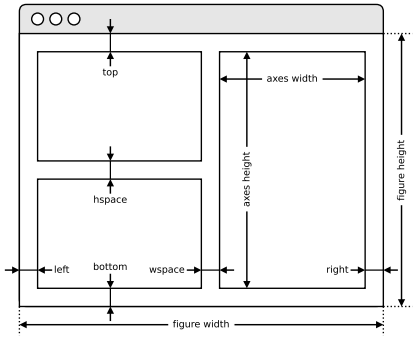
READ

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

Axes adjustments

API

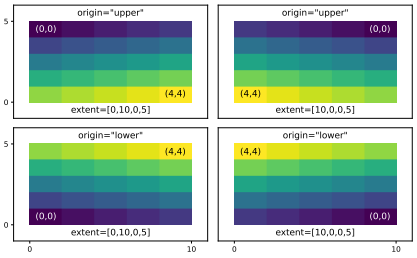
plt.**subplots_adjust**(...)



Extent & origin

API

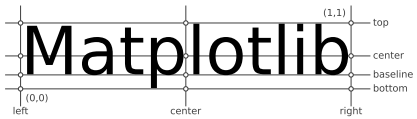
ax.**imshow**(extent=..., origin=...)



Text alignments

API

ax.**text**(..., ha=... , va=..., ...)



Text parameters

API

ax.**text**(..., family=..., size=..., weight=...)
ax.**text**(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

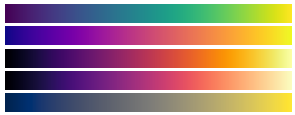
The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

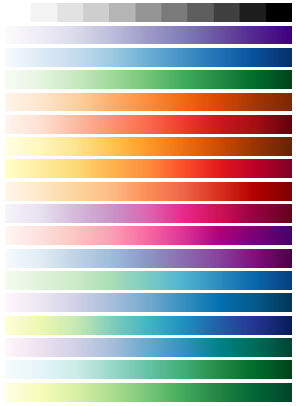
The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

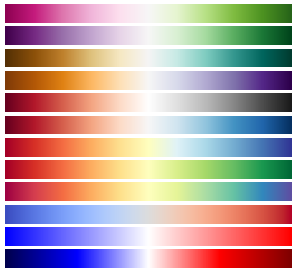
Uniform colormaps



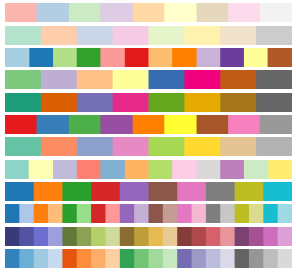
Sequential colormaps



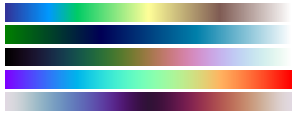
Diverging colormaps



Qualitative colormaps



Miscellaneous colormaps

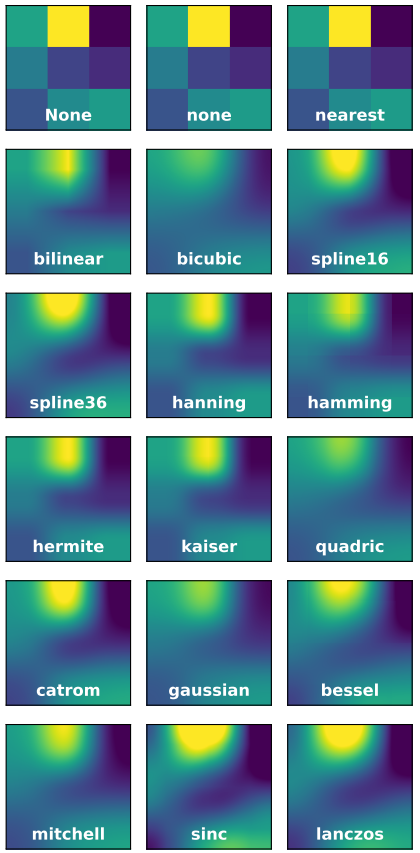


Color names

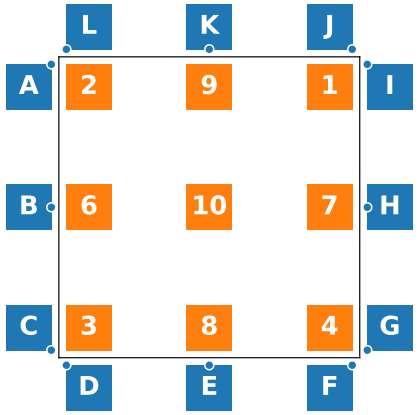
API

Image interpolation

API



Legend placement



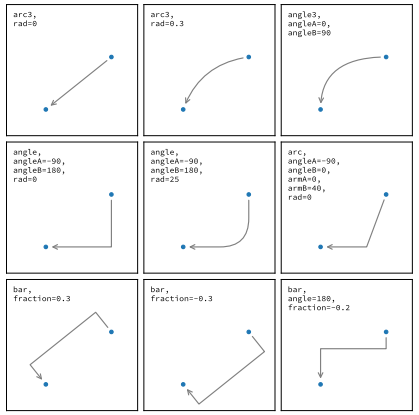
ax.**legend**(loc="string", bbox_to_anchor=(x,y))

2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / (-0.1,0.9) — B: center right / (-0.1,0.5)
C: lower right / (-0.1,0.1) — D: upper left / (0.1,-0.1)
E: upper center / (0.5,-0.1) — F: upper right / (0.9,-0.1)
G: lower left / (1.1,0.1) — H: center left / (1.1,0.5)
I: upper left / (1.1,0.9) — J: lower right / (0.9,1.1)
K: lower center / (0.5,1.1) — L: lower left / (0.1,1.1)

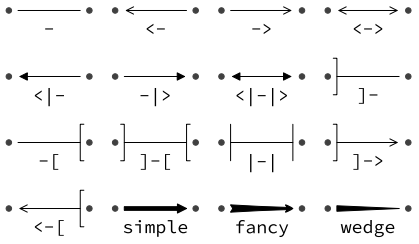
Annotation connection styles

API



Annotation arrow styles

API



How do I ...

... resize a figure?
→ fig.set_size_inches(w, h)
... save a figure?
→ fig.savefig("figure.pdf")
... save a transparent figure?
→ fig.savefig("figure.pdf", transparent=True)
... clear a figure/an axes?
→ fig.clear() → ax.clear()
... close all figures?
→ plt.close("all")
... remove ticks?
→ ax.set_[xy]ticks([])
... remove tick labels ?
→ ax.set_[xy]ticklabels([])
... rotate tick labels ?
→ ax.tick_params(axis="x", rotation=90)
... hide top spine?
→ ax.spines['top'].set_visible(False)
... hide legend border?
→ ax.legend(frameon=False)
... show error as shaded region?
→ ax.fill_between(X, Y+error, Y-error)
... draw a rectangle?
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... draw a vertical line?
→ ax.axvline(x=0.5)
... draw outside frame?
→ ax.plot(..., clip_on=False)
... use transparency?
→ ax.plot(..., alpha=0.25)
... convert an RGB image into a gray image?
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... set figure background color?
→ fig.patch.set_facecolor("grey")
... get a reversed colormap?
→ plt.get_cmap("viridis_r")
... get a discrete colormap?
→ plt.get_cmap("viridis", 10)
... show a figure for one second?
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                                  slow
plot(X, Y, marker="o", ls="")                  fast

for i in range(n): plot(X[i])                  slow
plot(sum([x+[None] for x in X],[]))            fast

cla(), imshow(…), canvas.draw()                slow
im.set_data(…), canvas.draw()                  fast
```

Beyond Matplotlib

Seaborn: Statistical data visualization
Cartopy: Geospatial data processing
yt: Volumetric data visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A grammar of graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE

randomForestSRC CHEAT SHEET

RFSRC
Fast Unified Forests

Basics

randomForestSRC is a fast OpenMP and memory efficient package for fitting random forests (RF) for univariate, multivariate, unsupervised, survival, competing risks, class imbalanced classification and quantile regression.
A basic grow call is of the form:

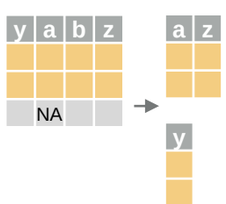
rfsrc(formula, data, ntree, mtry, nodesize)

Grow your RF through **rfsrc**,
specify your model in **formula**,
provide your data frame in **data**
and tune your model via **ntree, mtry, nodesize**.

Specify a formula

Survival	rfsrc(Surv(time, status) ~ ., data = veteran)
Competing Risk	rfsrc(Surv(time, status) ~ ., data = wihs)
Regression **Quantile Regression**	rfsrc(Ozone ~., data = airquality) quantreg(mpg ~ ., data = mtcars)
Classification **Imbalanced Two-Class**	rfsrc(Surv(time, status) ~ ., data=veteran) imbalanced(status ~ ., data = breast)
Multivariate Regression **Mixed Regression** **Quantile Regression** **MV Mixed Quantile**	rfsrc(Multivar(mpg, cyl) ~., data = mtcars) rfsrc(cbind(Species,Sepal.Length)~.,data=iris) quantreg(cbind(mpg, cyl) ~ ., data = mtcars) quantreg(cbind(Species,Sepal.Length)~.,data=iris)
Unsupervised **sidClustering** **Breiman (Shi-Horvath)**	rfsrc(data = mtcars) sidClustering(data = mtcars) sidClustering(data = mtcars, method = "sh")

Clean up and impute data



choose your variables in **formula** and grow a tree.

```
o <- rfsrc(y ~ a + z, data= dta, ntree = 1)
```

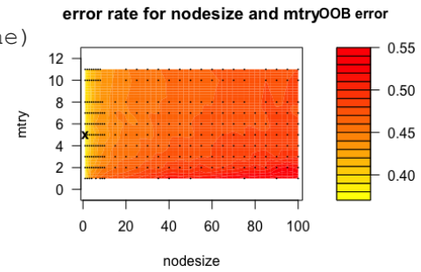
your outcome(s) will be saved in `o$y` and your predictors are in `o$x` from `dta` without missing values. To impute your data, use

```
o <- impute(y ~ a + z, data = dta)
o <- rfsrc(y ~ a + z, data = dta, na.action = "na.impute")
```

Tune mtry and nodesize

`tune` Find the optimal mtry and nodesize tuning parameter for a random forest using out-of-bag (OOB) error

```
o <- tune(quality ~ ., wine)
> o$optimal
nodesize      mtry
    1          5
```



`tune.nodesize` Find the optimal nodesize

Grow

Convenient interface for growing a CART tree

```
rfsrc.cart(formula, data, ntree = 1, mtry = ncol(data),
           bootstrap = "none")
```

Fast OpenMP parallel computing of random forests

```
rfsrc(formula, data, ntree = 500,
      mtry = NULL, ytry = NULL,
      nodesize = NULL, nodedepth = NULL,
      splitrule = NULL, nsplit = 10,
      importance = c(FALSE, TRUE, "none", "permute",
                     "random", "anti"),
      ensemble = c("all", "oob", "inbag"),
      bootstrap = c("by.root", "none", "by.user"),
      samptype = c("swor", "swr"),
      samp = NULL, membership = FALSE,
      na.action = c("na.omit", "na.impute"),
      nimpute = 1,
      ntime = 250, cause,
      proximity = FALSE, distance = FALSE,
      forest.wt = FALSE, xvar.wt = NULL,
      yvar.wt = NULL, split.wt = NULL,
      case.wt = NULL,
      forest = TRUE,
      var.used = c(FALSE, "all.trees", "by.tree"),
      split.depth = c(FALSE, "all.trees", "by.tree"),
      seed = NULL, do.trace = FALSE,
      statistics = FALSE, ...)
```

`rfsrc.fast` Fast approximate random forests using subsampling with forest options set to encourage computational speed

`rfsrc.anonymous` Random forests carefully modified so as not to save the original training data when sharing

`synthetic` Synthetic random forest using synthetic features

`imbalanced` Solutions to the two-class imbalanced problem

`quantreg` Univariate or multivariate quantile regression forest and returns its conditional quantile and density values

`sidClustering` Clustering of unsupervised data

Inference from the Forest

Ensemble Predicted Value for Training Data

```
o <- rfsrc(Ozone ~ ., data = airquality)
```

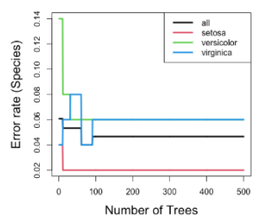
Inbag and out-of-bag (OOB) predicted values for the training dataset are in `o$predicted` and `o$predicted.oob`

Other Ensemble Values for Training Data

- For classification problem, we also have `$class` and `$class.oob` for class labels
- For survival problem, we have
  `$survival` and `$survival.oob` for survival function
  `$chf` and `$chf.oob` for cumulative hazard function
  `$cif` and `$cif.oob` for cumulative incidence function

Prediction Error for Assessing Model Performance

```
o <- rfsrc(Species ~ ., data=iris, block.size=1)
```



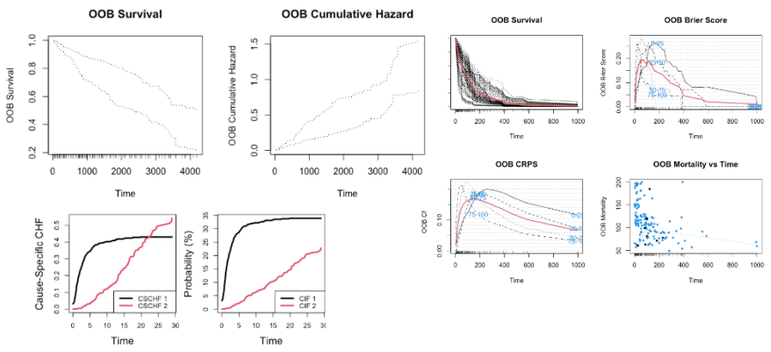
`o$err.rate` returns tree cumulative OOB error rate; `print(o)` lists OOB error rate in the bottom; `plot(o)` plots OOB error rate along with number of trees; `get.auc(y, prob)` obtains the value of AUC (area under the ROC curve)

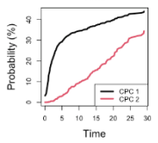
`get.mv.error` obtains error rate from a multivariate random forest

RFSRC
Fast Unified Forests

Visualization

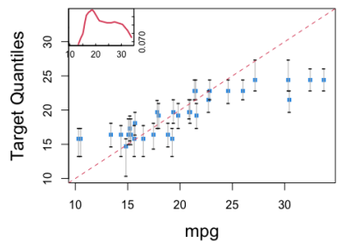
`plot.survival` plots various survival estimates





`plot.competing.risk` plots summary curves from a competing risk analysis

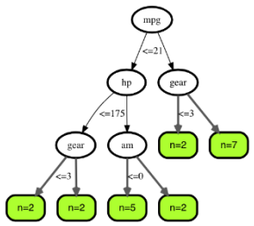
`plot.quantreg` plots quantiles obtained from a quantile regression forest



Tree Visualization

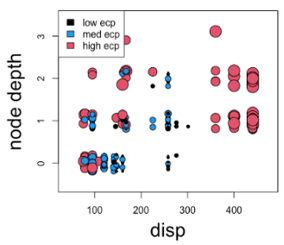
`get.tree` extract a single tree from a forest and plot it on your browser

```
mtcars.unspv <- rfsrc(data = mtcars)
plot(get.tree(mtcars.unspv, 5))
```



Split Statistics

`stat.split` acquires split statistic information. The end-cut preference (ECP) splitting property can be plotted



Predict on New Data

```
o.pred <- predict(object = o, newdata)
```

Predicted values for the new dataset are in `o.pred$predicted`

`get.mv.predicted` returns predicted value for multivariate regression analysis

Restore

Restoration using the `predict` function makes it possible for users to acquire information from the grow forest without the computational expense of having to regrow a new forest

Examples of restore are as follows (extract: proximity, variable splitting behavior, performance over specific trees)

```
o <- rfsrc(Ozone ~ ., data = airquality)

predict(o, proximity = TRUE)$proximity
predict(o.obj, var.used = "by.tree")$var.used
predict(o, get.tree=10:15)$err.rate
```

Variable Selection

Variable Importance (VIMP)

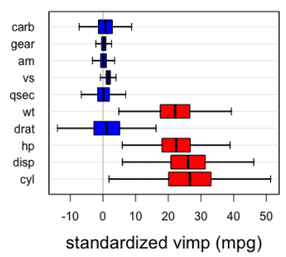
```
o <- rfsrc(Species ~ ., iris, importance = TRUE)
```
Or
```
obj <- rfsrc(Species ~ ., data = iris)
o <- vimp(obj)
```

`o$importance` returns permutation VIMP and `plot(o)` plots VIMP when setting importance to `"permute"` or `"TRUE"` in `rfsrc` or using `vimp`

`subsample` subsample forests for VIMP confidence intervals

`plot.sample` plots Subsampled VIMP confidence intervals

```
o <- rfsrc(mpg ~ ., mtcars)
smp.o <- subsample(reg.o, B=25,
                                subratio=.5)
plot.subsample(smp.o)
```



`holdout.vimp` calculates hold out VIMP from the error rate of blocks of trees grown with and without a variable

`get.mv.vimp` returns VIMP from a multivariate random forest

Minimal Depth

`max.subtree` extracts minimal depth and maximal subtree information used for variable selection and identifying interactions between variables

Variable Selection and Hunting

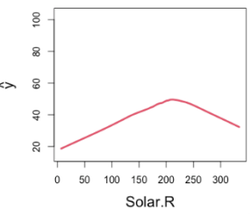
`var.select(formula, data, method)` Variable selection or hunting by setting method

`md`	Minimal depth (default)
`vh`	Variable hunting
`vh.vimp`	Variable hunting with VIMP

Partial Plot

Marginal Effect Plot

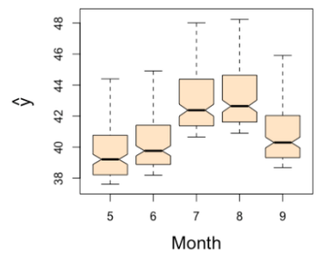
`plot.variable(o, xvar.names)`



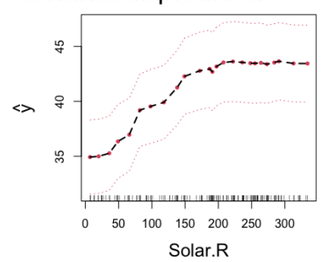
Partial Dependence Plot

`plot.variable(o, xvar.names, partial = TRUE)` and `partial`

Categorical predictor:



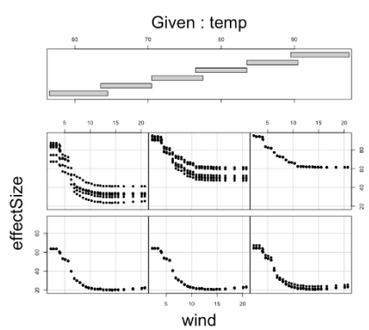
Continuous predictor:

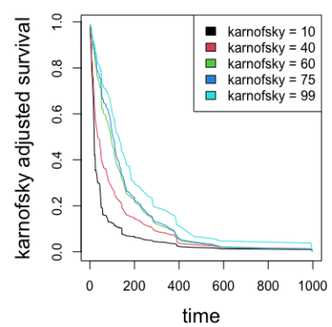


Set `surv.type` for survival analysis:

`mort`	Mortality
`rel.freq`	Relative frequency of mortality
`surv`	Predicted survival, where the predicted survival is for the time point specified using *time*
`years.lost`	The expected number of life years lost
`cif`	The cumulative incidence function
`chf`	The cumulative hazard function

`get.partial.plot.data` is a handy function that parses the output from `"partial.rfsrc"` in format suitable for plots





Large Language Models：：CHEAT SHEET

Large Language Models (LLMs)

LLMs are artificial intelligence models that can generate human-like text, based on patterns found in massive amounts of training data. They are used in applications such as language translation, chatbots, and content creation.

Some popular LLMs

Some popular LLMs include GPT-3 (Generative Pretrained Transformer by OpenAI, BERT (Bidirectional Encoder Representations from Transformers) by Google, and XLNet (eXtreme MultiLingual Language Model) by Carnegie Mellon University and Google.

BigScience	BLOOM	176B	July 2022
	T0pp	11B	October 2021
EleutherAI	GPT-J	6B	July 2021
	GPT-NeoX	20B	February 2022
清华大学 Tsinghua University	GLM	130B	August 2022
Google Research	UL2	20B	October 2022
	T5	11B	February 2020
Meta AI	OPT	175B	June 2022
	OPT	66B	June 2022
Yandex	YaLM	100B	June 2022

Choose between LLMs

When comparing different models, it's important to consider their architecture, the size of the model, the amount of training data used, and their performance on specific NLP tasks.

Components of LLMs

LLMs typically consist of an encoder, a decoder, and attention mechanisms. The encoder takes in input text and converts it into a set of hidden representations, while the decoder generates the output text. The attention mechanisms help the model focus on the most relevant parts of the input text.

Applications of LLMs

- LLMs are used in a wide range of applications, **including language translation, chatbots, content creation, and text summarization**.
- They can also be used to improve search engines, voice assistants, and virtual assistants.

How are LLMs trained?

LLMs are trained using a process called unsupervised learning. This involves feeding the model massive amounts of text data, such as books, articles, and websites, and having the model learn the patterns and relationships between words and phrases in the text. The model is then fine-tuned on a specific task, such as language translation or text summarization.

Preprocessing

Text normalization is the process of converting text to a standard format, such as lowercasing all text, removing special characters, and converting numbers to their written form.

Tokenization is the process of breaking down text into individual units, such as words or phrases. This is an important step in preparing text data for NLP tasks.

Stop Words are common words that are usually removed during text processing, as they do not carry much meaning and can introduce noise or affect the results of NLP tasks. Examples of stop words include "the," "a," "an," "in," and "is."

Lemmatization is the process of reducing words to their base or dictionary form, by taking into account their part of speech and context. It is a more sophisticated technique than stemming and produces more accurate results, but it is computationally more expensive.

Stemming and lemmatization are techniques used to reduce words to their base form. This helps to reduce the dimensionality of the data and improve the performance of models.

Fine-Tuning

Fine-tuning is the process of training a pre-trained large language model on a specific task using a smaller dataset. This allows the model to learn task-specific features and improve its performance.

The fine-tuning process typically involves freezing the weights of the pre-trained model and only training the task-specific layers.

When fine-tuning a model, it's important to consider factors such as the size of the fine-tuning dataset, the choice of optimizer and learning rate, and the choice of evaluation metrics

Example of fine-tuning LLMs

•Model Cost: $500 - $5000 per month, depending on the size and complexity of the language model

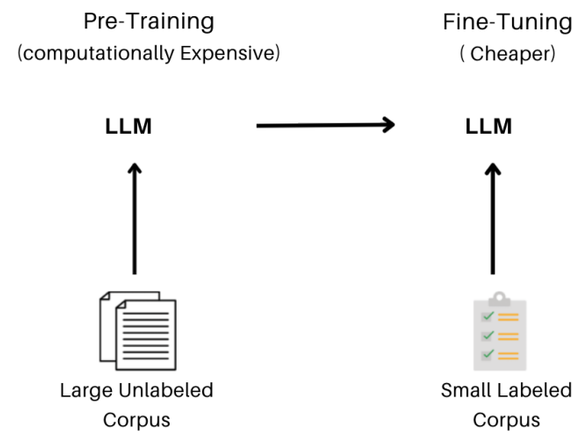
•GPU size: NVIDIA GeForce RTX 3080 or higher

•Number of GPUs: 1-4, depending on the size of the language model and the desired speed of fine-tuning. For example, fine-tuning the GPT-3 model, which is one of the largest language models available, would require a minimum of 4 GPUs.

•The size of the data that GPT-3 is fine-tuned on can vary greatly depending on the specific use case and the size of the model itself. GPT-3 is one of the largest language models available, with over 175 billion parameters, so it typically requires a large amount of data for fine-tuning to see a noticeable improvement in performance.

Note: fine-tuning GPT-3 on a small dataset of only a few gigabytes may not result in a significant improvement in performance, while fine-tuning on a much larger dataset of several terabytes could result in a substantial improvement. The size of the fine-tuning data will also depend on the specific NLP task the model is being fine-tuned for and the desired level of accuracy.

This is just one example, and actual costs and GPU specifications may vary depending on the language model, the data it is being fine-tuned on, and other factors. It's always best to check with the language model provider for the latest information and specific recommendations for fine-tuning.



Input Representations:

•**Word embeddings:** Each token is replaced by a vector that represents its meaning in a continuous vector space. Common methods for word embeddings include Word2Vec, GloVe, and fastText.

•**Subword embeddings:** Each token is broken down into smaller subword units (e.g., characters or character n-grams), and each subword is replaced by a vector that represents its meaning. This approach can handle out-of-vocabulary (OOV) words and can improve the model's ability to capture morphological and semantic similarities. Common methods for subword embeddings include Byte Pair Encoding (BPE), Unigram Language Model (ULM), and SentencePiece.

•**Positional encodings:** Since LLMs operate on sequences of tokens, they need a way to encode the position of each token in the sequence. Positional encodings are vectors that are added to the word or subword embeddings to provide information about the position of each token.

•**Segment embeddings:** In some LLMs, such as the Transformer, the input sequence can be divided into multiple segments (e.g., sentences or paragraphs). Segment embeddings are added to the word or subword embeddings to indicate which segment each token belongs to.

Ashish Patel • Principal Research Scientist • ashishpatel.ce.2011@gmail.com

Abonia Sojasingarayar • Machine Learning Scientist • aboniaa@gmail.com

Updated: 2023-02

Large Language Models：：CHEAT SHEET

Attention Mechanisms

Self-Attention:

•A mechanism that allows a sequence to weigh the importance of all other elements in the sequence when computing a representation for each element.

•Can capture relationships between different elements in the sequence, making it well-suited for tasks that require modeling long-range dependencies.

•Popularized by the Transformer model.

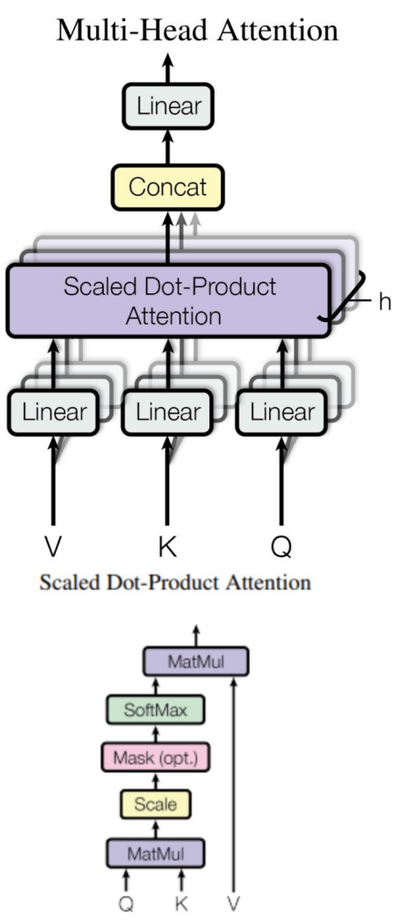
Multi-Head Attention:

•A variation of self-attention where the attention mechanism is applied multiple times in parallel with different sets of weights.

•Allows the model to attend to different aspects of the input sequence, improving its ability to capture complex patterns and dependencies.

•Each "head" produces a separate output that is concatenated and linearly transformed to produce the final output.

•Also commonly used in the Transformer model.



Evaluating LLMs

•**Accuracy** measures the proportion of correctly classified instances out of all instances. This metric is commonly used for text classification tasks such as sentiment analysis, where the goal is to correctly classify a text as positive, negative, or neutral.

•**F1-score** is a metric that takes into account both precision and recall. Precision is the proportion of true positive results out of all predicted positive results, while recall is the proportion of true positive results out of all actual positive results. The F1-score is the harmonic mean of precision and recall, and it provides a balanced measure of model performance on text classification, question answering, and other tasks.

•**Perplexity** : It measures how well a language model predicts the next word in a sequence. The lower the perplexity, the better the model is at predicting the next word. Perplexity is calculated as 2 to the power of the cross-entropy, which is a measure of how well the model's predicted probabilities match the true probabilities of the next word in the sequence.Also text can be evaluated using metrics such as BLEU or ROUGE scores.

Example - ChatGPT

ChatGPT is a large language model developed by OpenAI that has been trained on a diverse range of internet text, including articles, forums, and social media. This training allows ChatGPT to generate human-like text and answer a wide variety of questions on many topics.

Use Cases ChatGPT

Question Answering: ChatGPT can answer factual questions based on the information it has been trained on. Example:

Human : What is the capital of France?

ChatGPT: The capital of France is Paris.

Conversational: ChatGPT can engage in a conversation with a user. Example:

Human : Hi, how are you today?

ChatGPT: Hello! I'm just an AI, so I don't have emotions, but I'm functioning well today. How can I assist you?

Text Generation: ChatGPT can generate text based on a prompt or seed text. Example:

Human : Write a short story about a magical world.

ChatGPT: Once upon a time, in a land far away, there was a magical world filled with wonder and enchantment. The sky was always a brilliant shade of purple, and the stars twinkled like diamonds in the sky. The people of this world possessed magical powers that they used to create incredible wonders and to help those in need. One day, a young woman discovered that she too had magical powers, and set out on an incredible journey to explore the wonders of this magical world and to hone her skills.

Summarization: ChatGPT can summarize a long text or article. Example:

Human : Please summarize this article about the benefits of exercise.

ChatGPT: The article discusses the various physical and mental health benefits of exercise, including improved cardiovascular health, stronger muscles and bones, better weight management, and reduced stress and anxiety. It also highlights the importance of finding an exercise routine that is enjoyable and sustainable.

Translation: ChatGPT can translate text from one language to another. Example:

Translate this sentence from English to Spanish. "The sky is blue."

ChatGPT: El cielo es azul.

Tools&Libraries support LLMs

a. Popular NLP libraries, such as TensorFlow, PyTorch, spaCy, Hugging Face Transformers, AllenNLP,OpenAI GPT-3 API, AllenAI's ELMO, spaCy Transformers etc provide tools for working with large language models. These libraries allow for easy fine-tuning and deployment of models.

b. Some large language models, such as GPT-3, provide APIs for access to their models. This can simplify the process of integrating a large language model into a real-world application.

TensorFlow spaCy PyTorch OpenAI

Challenges and limitations with LLMs

- One of the main challenges with LLMs is the potential for biased or offensive language, as the models learn from the patterns found in the training data.
- Unethical considerations, such as gender and racial biases.
- Amount of computational resources needed to train and run LLMs, which can be expensive and energy-intensive.
- Handling out-of-vocabulary words
- Improving interpretability. While large language models have shown impressive performance on a variety of NLP tasks, they may not perform as well on specific tasks, such as those that require a deeper understanding of the underlying context.

Applications of LLMs

- LLMs are used in a wide range of applications, **including language translation, chatbots, content creation, and text summarization**.
- They can also be used to improve search engines, voice assistants, and virtual assistants.

Future of LLMs

The future of LLMs is promising, with ongoing research focused on improving their accuracy, reducing bias, and making them more accessible and energy-efficient.

As the demand for AI-driven applications continues to grow, LLMs will play an increasingly important role in shaping the future of human-machine interaction.

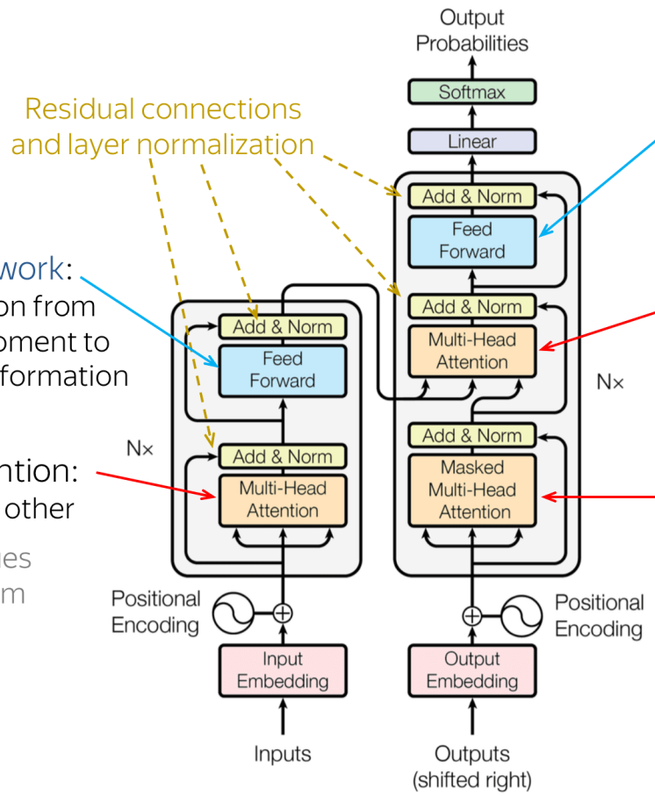
Ashish Patel • Principal Research Scientist • ashishpatel.ce.2011@gmail.com

Abonia Sojasingarayar • Machine Learning Scientist • aboniaa@gmail.com

Updated: 2023-02

Large Language Models : : CHEAT SHEET

Transformer Architecture



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

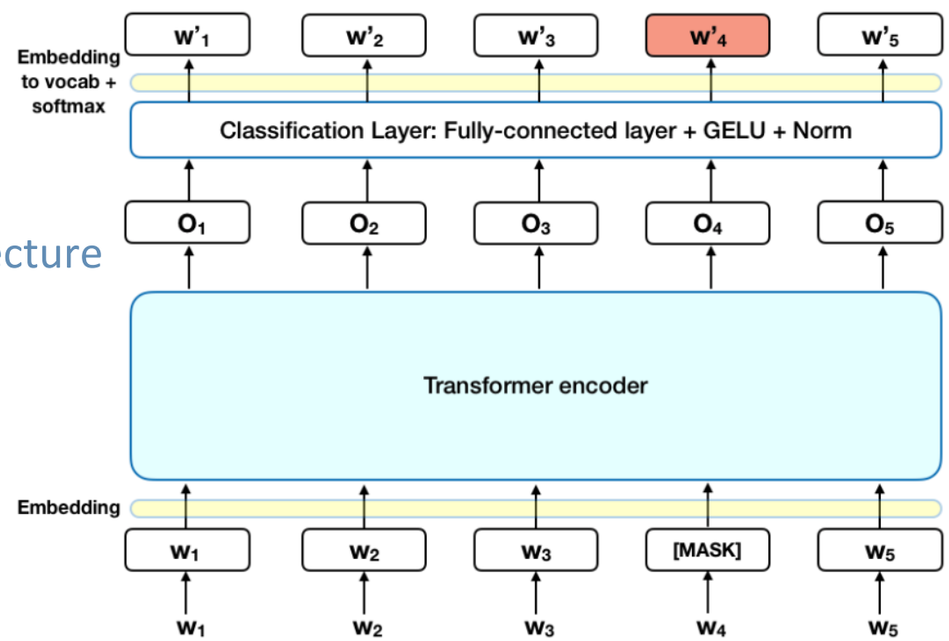
Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

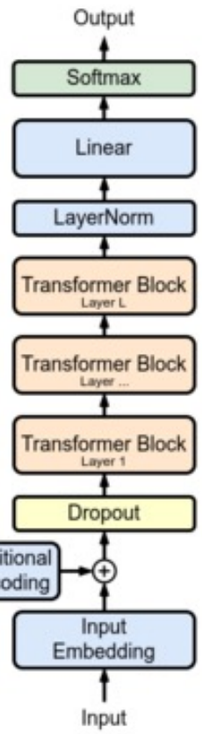
Decoder self-attention (masked): tokens look at the previous tokens

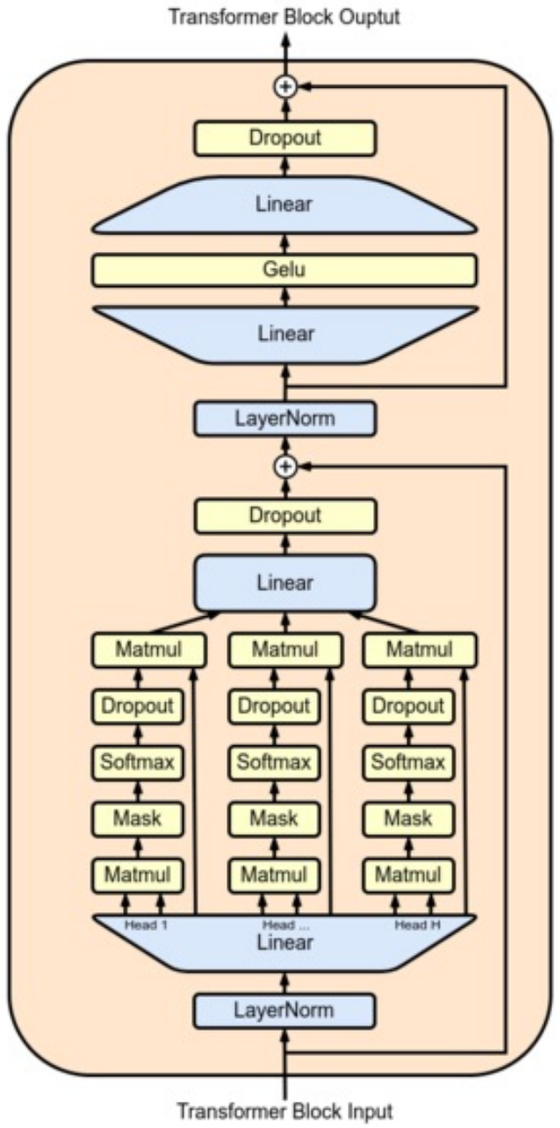
queries, keys, values are computed from decoder states

BERT Architecture



GPT Architecture





Ashish Patel • Principal Research Scientist • ashishpatel.ce.2011@gmail.com

Abonia Sojasingarayar • Machine Learning Scientist • aboniaa@gmail.com

Updated: 2023-02