Stanford CS229 : Machine Learning

Complete Handwritten Notes

Devansh Chaudhary

Date	Title	Page No.	Teacher's Sign / Remarks
Lec 1	Intro to Linear Algebra		
Lec 2	Matrix Calculus & Probability		
Lec 3	Probability and Statistics		
Lec 4	Linear Regression		
Lec 5	Perceptron & logistic Regrs.		
Lec 6	Exponential family & GLM		
Lec 7	GDA, Naive Bayes & Laplace Smooth		
Lec 8	Kernel methods & SVM		
Lec 9	Bayesian Methods - Param & Non		
Lec 10	Deep Learning - I		
Lec 11	Deep Learning - II		
Lec 12	Bias and Variance & Regularisation		
Lec 13	Statistical learning Uniform Conv.		
Lec 14	Reinforcement Learning - I		
Lec 15	Reinforcement Learning - II		
Lec 16	K-means, GMM, EM		
Lec 17	Factor Analysis & ELBO		
Lec 18	Principal & Independent CA		
Lec 19	Maximum Entropy & Calibration		
Lec 20	Variational Autoencoder		
Lec 21	Evaluation Matrices		
Lec 22	Practical tips and Course Recap		
Lec 23	Course Recap and Wrap Up		

Stanford CS229

Machine Learning | Lec 1 - Intro. and Linear Algeb.

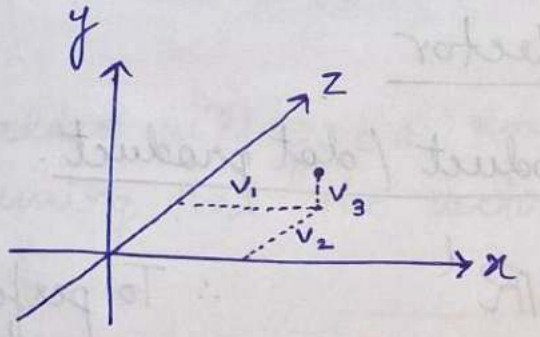
Machine learning is the study of computer algorithms that improve automatically through experience - Tom Mitchell, prof. CMU

Notation

Vector $v \in \mathbb{R}^d$: v is a vector and it is an element of this set and this set is a d-dimensional real space.

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$$v^T = [v_1 \dots v_d]$$



$\mathbb{R}^d \rightarrow \mathbb{R}^3$

* Generally for vector we use small letters notation and for Matrices we use Capital letters notation.

Matrix

$$A \in \mathbb{R}^{m \times n}$$

$\longleftarrow n \longrightarrow$

$$A = \begin{matrix} \uparrow \\ m \\ \downarrow \end{matrix} \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mxn} \end{bmatrix}$$

$m \rightarrow$ rows

$n \rightarrow$ columns

Identity Matrix (I) = $\begin{bmatrix} 1 & & 0 \\ & 1 & \\ 0 & & \ddots \end{bmatrix}$

Diagonal Matrix = $\begin{bmatrix} & & 0 \\ & \diagdown & \\ 0 & & \end{bmatrix}$

Symmetric Matrix = $A = A^T$

$A_{ij} = A^T_{ji}$

Trace of a Matrix $\sum_i A_{ii}$

Basic operations

- Vector - Vector

Inner product / dot product.

$x, y \in \mathbb{R}^d$

∴ To perform inner product the vectors have to be of the same dimension

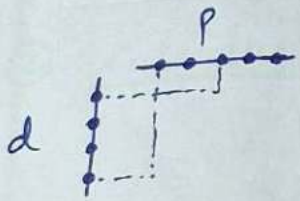
$\sum_{i=1}^{d} x_i y_i \; ; x^T y$

$d \;|\; d = \cdot$ (scalar)

Outer product

$x \in \mathbb{R}^d, y \in \mathbb{R}^l$ (The two vectors need not to be of the same dimension)

xy^T , yx^T (The two are different).

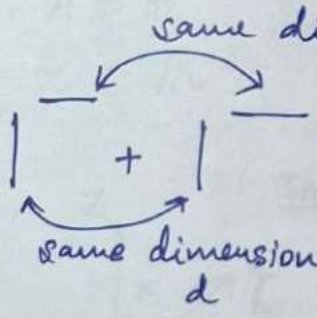


$= \begin{bmatrix} & a_{13} \\ \cdots & \\ a_{+1} & \end{bmatrix}$

Taking i^{th} element from x and j^{th} element from y, then multiply it and it gives ij element of the outer product

Rank-1 Matrix

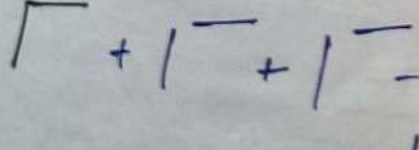
⇒ It is made of one pair of row and column.



$= [\ ]$ → Rank-2

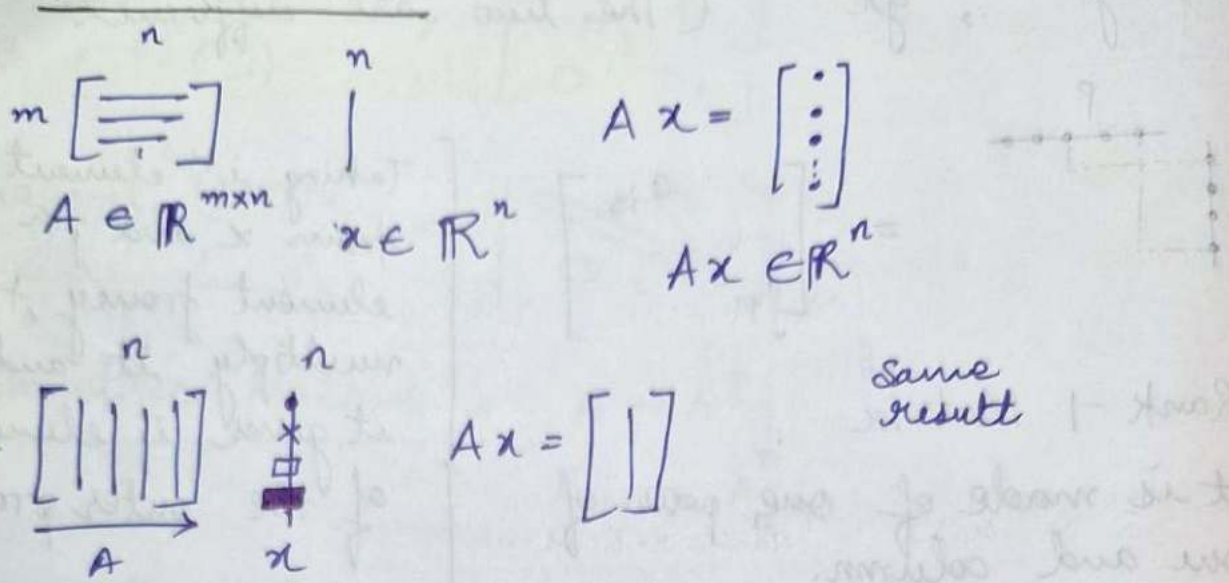
Two Rank-1 matrix added will give a Rank-2 Matrix ~~provided the~~ R. assuming that the vectors are linearly independent.

↳ LI vectors are a set of vectors that do not lie on the same plane or same line and cannot be expressed as a linear combination of each other.

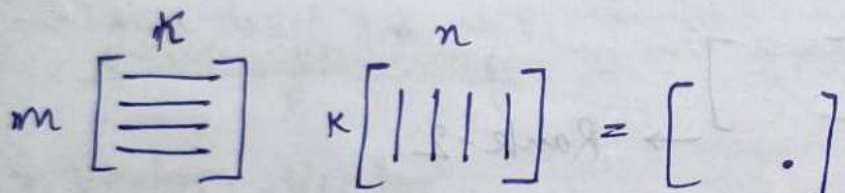


$= [\ ] \leq \min(d, p, k)$

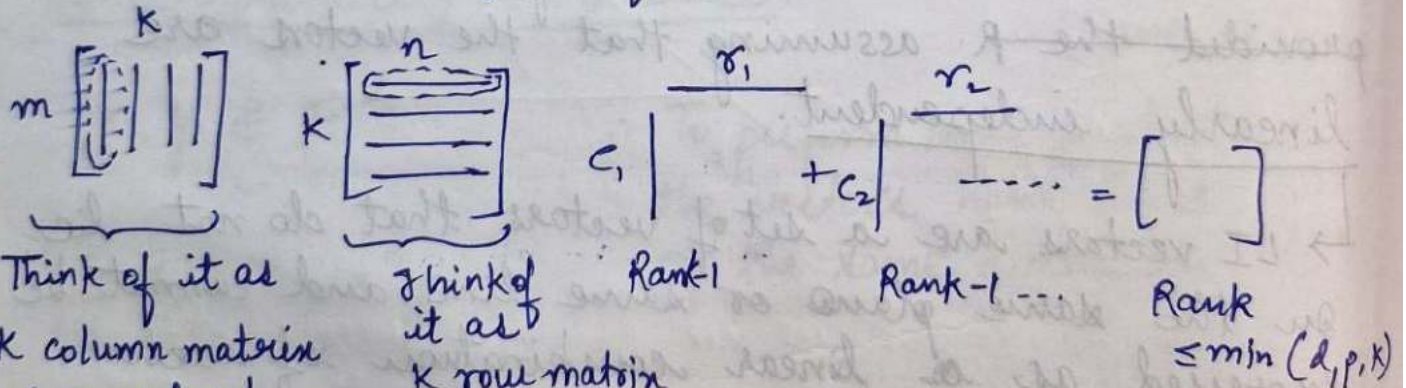
Matrix Vector



Matrix - Matrix



or can be thought of as



Why Linear Algebra

① Represent data

Both Kitchen...

$X = \text{house} \begin{bmatrix} 1 & 1 & 1 \\ \vdots & \vdots & \vdots \end{bmatrix}$ $\quad Y = \begin{bmatrix} \text{cost} \\ \vdots \end{bmatrix}$

② Covariance Matrices

$S \in \mathbb{R}^{d \times d}$

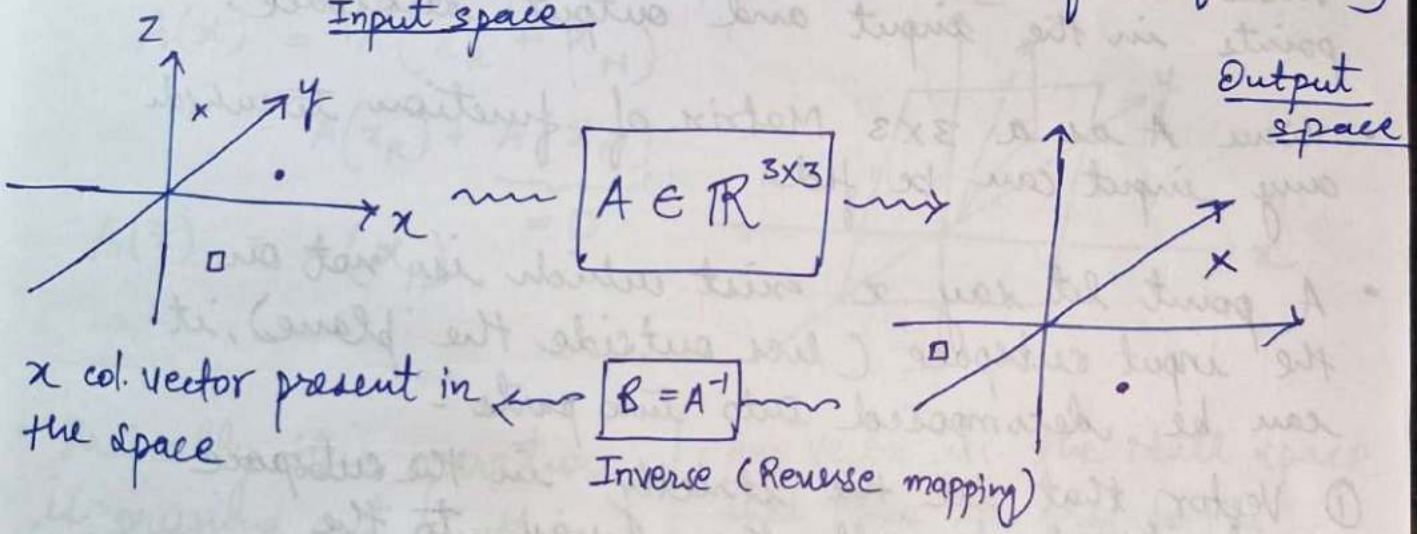
③ Calculus → Gradients - vector
Hessians - Matrix (symmetric)
Jacobians - Matrix

④ Kernal Method

Geometrical Interpretation

$A \in \mathbb{R}^{m \times n}$
$x \in \mathbb{R}^n \Rightarrow A(x) \in \mathbb{R}^m$

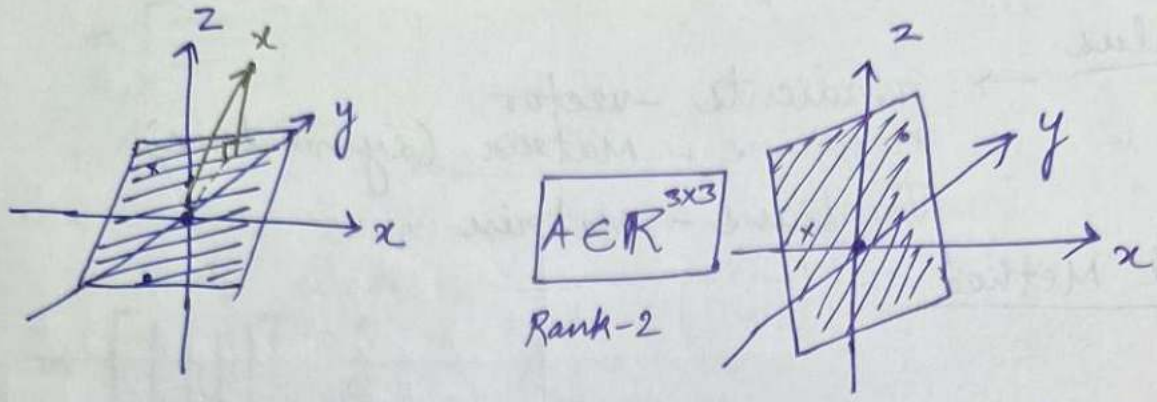
{ Think of Ax as A(x) - A as a func. of x }



If A is full Rank (i.e Rank 3) there is going to be unique mapping of every point in the input to every point in the output.

FULL RANK

RANK DEFICIENT



- There exist a 2D subspace that exists in the 3D space that is specific to this matrix. Also a corresponding 2D subspace in the output
- There exist a one-to-one mapping between the points in the input and output subspace.
- View A as a 3x3 Matrix of function to which any input can be fed
- A point let say x exist which is not on the input subspace (lies outside the plane), it can be decomposed into two parts -

① Vector that lie strictly in the subspace and it extends all the way to the nearest point to x.

② Point (Vector) ⊥ to the subspace which originates from the nearest point to the point x.

Basically splitting the vector into its components.

① $\bar{x}$ = Proj (x; Rowspace) + Proj (x; Nullspace).

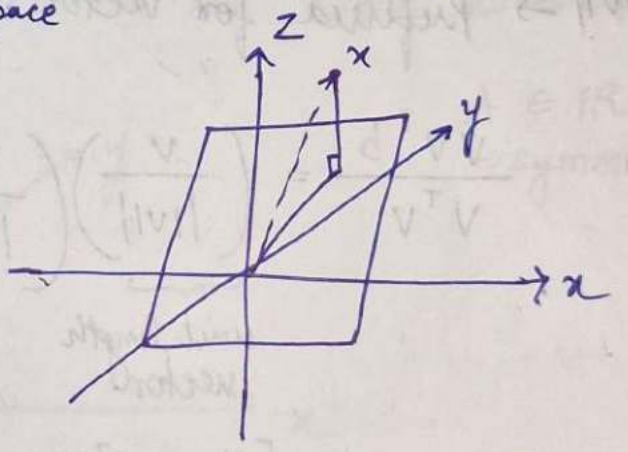
- The input is called the rowspace because it is made precisely of those points which can be represented as the linear combinations of the rows of A.
- Similarly for the output space, just the change is that it is the linear combination of the columns of A. (Can be called as the Range)
- Basically the input is precisely that subspace for which a bijection exist between the output and input.
- Subspace always passes through the origin by definition

Row ↑ Null space ↑

$$A(x) = A(x_R + x_N)$$

$$= A(x_R) + \underbrace{A(x_N)}_{=0}$$

$$A(x) = A(x_R)$$

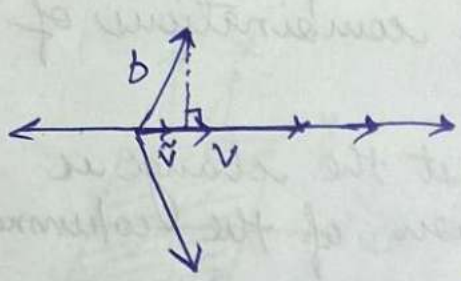


- Basically this operation is pruning out anything that is outside the row space from the input and project on the output.

- Any vector in the Null space will have its proj on the origin through A.

$$\Rightarrow A(x_N) = 0$$

Projection



Projection Matrix

$$(v) = \left[\frac{vv^T}{v^Tv}\right]b$$

$$\frac{vv^Tb}{v^Tv} = \left(\frac{v}{\|v\|}\right)\left(\frac{v}{\|v\|}\right)^T b$$

$\|v\|$ → Norm of a vector
generalise to length or magnitude

$|v|$ → Mag. (Preferred for scalars)

$\|v\|$ → Prefered for vectors.

$$\frac{vv^Tb}{v^Tv} = \underbrace{\left(\frac{v}{\|v\|}\right)}_{\text{unit length vector}}\left(\frac{v}{\|v\|}\right)^T b$$

$$= [\tilde{v}\ \tilde{v}^T]b$$

$$(v) = \underbrace{\tilde{v}}_{\text{scalar}}\left(\underbrace{\tilde{v}^Tb}_{\text{length}}\right)$$

$\equiv$

- If we want to project B onto subspace spanned by multiple vectors. Then in place of v we will use some matrix $x\,[|||]$ → complete subspace

$$\left[\frac{VV^T}{V^TV}\right]b = V(V^TV)^{-1}V^Tb$$

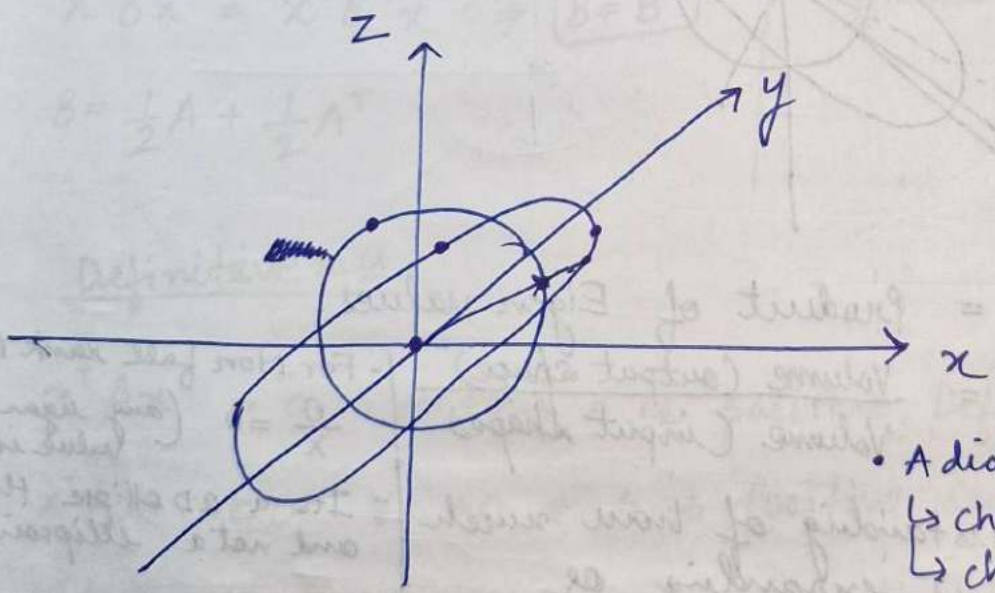
$$\Rightarrow X(X^TX)^{-1}X^T$$

$(m\times n)\ (n\times n)\ (n\times m)$ → dimension

- For no point in the input space can we reach a point that lives outside the column space or output subspace.

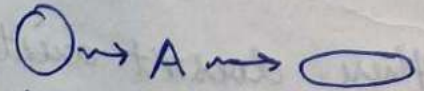
Square Matrices

↳ Input and output space have the same dimension.



$A \in \mathbb{R}^{3\times 3}$
↳ symmetric

- A did two things
  - ↳ change in mag
  - ↳ change in dirn
- For some points there is going to be only change in mag. and not direc^n ⇒ called eigen vectors.
- It will have 3 eigen vectors.

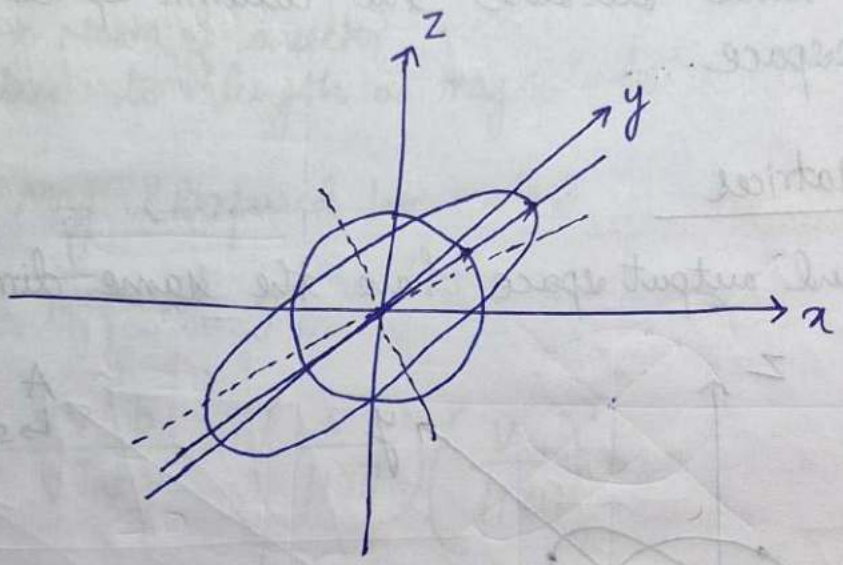
- 

Rank of a Matrix is precisely the number ← of non-zero eigen values

$$\text{Eigen value} = \frac{\text{length of output vector}}{\text{length of input vector}}$$

Lecture-2 : Matrix Calculus and Probability theory

- Eigen vectors are those special vectors for square matrices where the operation of the matrix does not change the angle, it only rescales it by some amount.
- The eigen value is the ratio by which the vector gets rescaled. So each eigen vector comes with corresponding eigen value.



$A \in \mathbb{R}^3$ symmetric

- Determinant = Product of Eigen values.

$$= \frac{\text{Volume (output shape)}}{\text{Volume (input shape)}}$$

- For Non full rank Matrix $\frac{0}{x} = 0$ (any eigen value is 0)
- Its a 2D ellipse then and not a ellipsoid

- Det. gives understanding of how much the Matrix is expanding or contracting.
- If any eigen value is 0. Then there doesn't exist Matrix inverse (2D ellipse cannot be mapped back to a 3D sphere)

Spectrum - Collection of Eigen values

Spectral theorem - For every square matrix $A \in \mathbb{R}^{d \times d}$ which is also symmetric $A = A^T$ has

- Real valued Eigen values
- Orthonormal Eigen vectors.

Quadratic form

$A \in \mathbb{R}^{d \times d} \quad x \in \mathbb{R}^d$

$x^T A x \rightarrow$ Quadratic form

$x^T A x = x^T B x$

$x^T B x = x^T B^T x \Rightarrow \boxed{B = B^T}$

$B = \frac{1}{2} A + \frac{1}{2} A^T$

- Its good to assume that Matrix A is symmetric too, as it can always be expressed as in the form of other matrix B which is symmetric

Definitiveness

$x^T A x > 0 \quad \forall x \neq 0 \Rightarrow$ A is positive Definite (PD)

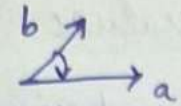
$x^T A x \geq 0 \quad \forall x \neq 0 \Rightarrow$ A is Positive Semi Definite (PSD)

$x^T A x < 0 \quad \forall x \neq 0 \Rightarrow$ A is Negative Definite (ND)

$x^T A x \leq 0 \quad \forall x \neq 0 \Rightarrow$ A is Negative Semi Definite (NSD)

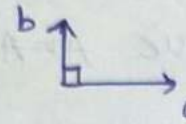
$x^T A x \gtrless 0 \Rightarrow$ A is indefinite.

$a^T b$ $\equiv$ $a^T b$

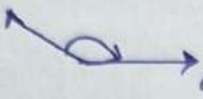


> 0

Dot product



$= 0$



< 0

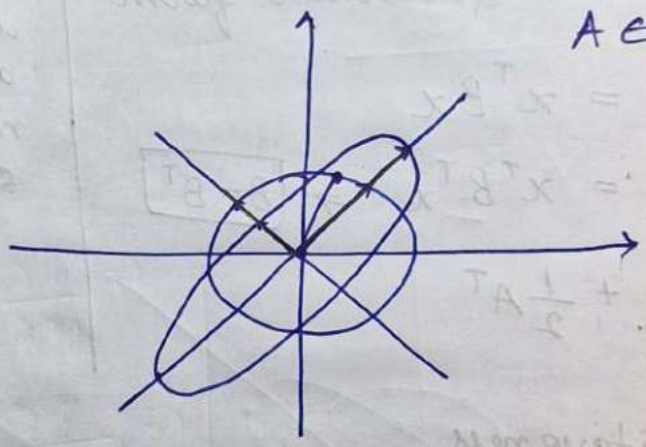
$x^T A x \Rightarrow (x)^T (Ax)$

a b

- So

	eigen values
PD	> 0
PSD	≥ 0
ND	< 0
NSD	≤ 0
ID	$>, <$

- The definitiveness of a matrix which is defined by the quadratic equation/form has a one-to-one relation which the spectrum of the matrix.

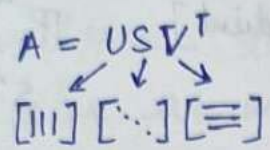
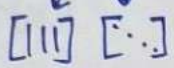


$A \in \mathbb{R}^2$

- The eigen vectors kind of act as pivot
- It means vectors inside one quadrant remain in the same quadrant in the output space.

Decomposition

- Singular value decomposition
- Eigen value decomposition
- Cholesky decomposition

	A	Decomposition	
SVD	Any	$A = USV^T$  $A(x) = U(S(V^T(x)))$ Nested pipeline	• U, V orthonormal $S = [\ddots] \rightarrow$ Real $D = [\ddots] \rightarrow$ Eigen values can be complex
EVD	Square	$A = UDU^{-1}$  $A(x) = U(D(U^{-1}(x)))$	

A	Step 1	Step 2	Step 3
SVD	V^T Rotation	Scaling along basis (Real valued)	U Rotation
EVD	U^{-1} Rotation	Scaling along axis (Rotation) EV is complex	(Rotation) U inverse of step1

- A arbitrary SVD
- A square symm.
  ↳ SVD = EVD
- A square
  ↳ can do both but the decomposed components may not be identical.

- Eigen value decomposition defined only for square matrices while SVD is defined for any matrices.

$Ax = \lambda x$ ⇒ Eigen value is basically take a matrix A, multiply by x. We get an output vector x which is scaled by eigen value λ.
⇒ A has to be square

MATRIX CALCULUS

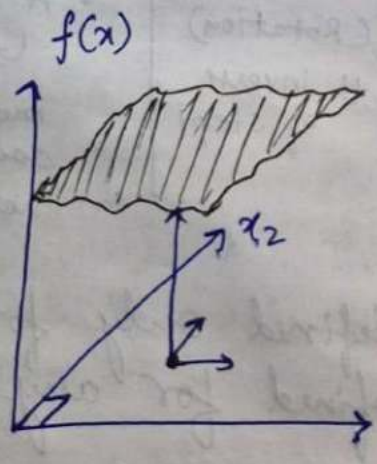
function	Eg	Value	First deriv	Second deriv
$f: R \to R$	x^2	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$
$f: R^d \to R$ (vec) (sca)	Loss func.	$\mathbb{R}$	$\mathbb{R}^d$ [Gradient]	$\mathbb{R}^{d \times d}$ [Hessian] S^d
$f: R^d \to R^p$ vec vec	NN layer	$\mathbb{R}^p$	$\mathbb{R}^{d \times p}$ [Jacobian]	$\mathbb{R}^{d \times p \times p}$ [High order tensor]

Gradient

It is the direction of the the steepest ascent

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d}(x) \end{bmatrix}$$

$\nabla_x f(x_1, x_2 \cdots x_d)$



The gradient tells the direction in which we want to move to increase the value of the function the most. of the func.

Here we are taking vector as input and giving scalar as output.

$$f: R^d \to R$$

$A \rightarrow$ Matrix

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial a_{11}}(A) & \cdots \\ & \\ \frac{\partial}{\partial a_{mn}} f(A) & \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

$\# f: R^{m \times n} \rightarrow R$

f: is a function that takes Matrix A as input and gives scalar as output.

Hessian

$$\nabla_x^2 f(x) \qquad f: R^d \rightarrow R$$

$$\nabla_x^2 f(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_d} \\ \vdots & & \\ \frac{\partial^2 f(x)}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_d \partial x_d} \end{bmatrix}$$

Example

$$\nabla_x b^T x = \begin{bmatrix} \vdots \\ \frac{\partial}{\partial x_i}(b^T x) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_i}(b_1 x_1 + b_2 x_2 \cdots b_i x_i + \cdots b_d x_d) \end{bmatrix}$$

$$\nabla_x b^T x = \begin{bmatrix} \vdots \\ b_i \\ \vdots \end{bmatrix} = b$$

Product Rule

$\nabla_x \underbrace{x^T}_{f(x)} \underbrace{Ax}_{g(x)} \Rightarrow x^T \nabla_x (Ax) + Ax \nabla_x (x^T)$

$x^T A + A x \quad (1)$

$x^T (A + A^T)$ $\quad \therefore A$ is symmetric $A = A^T$

$$\boxed{\nabla_x (x^T A x) = 2Ax}$$

$$\boxed{\nabla_A \log |A| = A^{-1}}$$

$$\frac{d}{dx} \log(x) = x^{-1}$$

Review of Probability theory

- Set of all possible outcomes is called the sample space. (Ω) $\{HH, HT, TH, TT\}$
- An event A is subset of the sample space.
  $A \subseteq \Omega$ $\{HH, HT\}$
- The set of all possible subsets of sample space (F)

Probability Measure $P: F \to R$ (Here we are assigning prob. to events not outcomes)

$P(A) \geq 0 \quad \forall A \in F$

$P(\Omega) = 1$

- If $A_1, A_2 \ldots$ disjoint events then $P(\cup A_i) = \sum P(A_i)$

Conditional probability

Let $P(B) \neq 0$

$$P(A/B) = \frac{P(A \cap B)}{P(B)}$$

$A \perp B$ only when $P(A \cap B) = P(A) \cdot P(B)$

$A \perp B$ only when $P(A/B) = \frac{P(A)\,P(B)}{P(B)} = P(A)$.

$\perp \to$ Independent of.

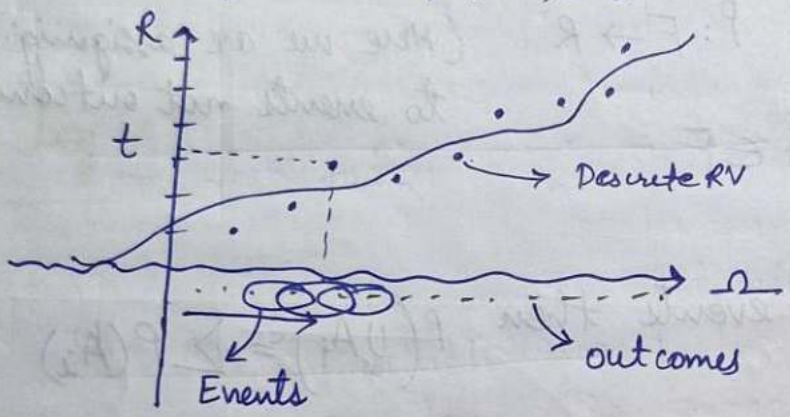
Random Variable (RV)

A random variable is a mapping from outcomes to real values.

$\omega_0 = HHHTTHTTHTTTT$

A RV is $X : \Omega \rightarrow R$

# of heads $X(\omega_0) = 5$
# of tails $X(\omega_0) = 8$



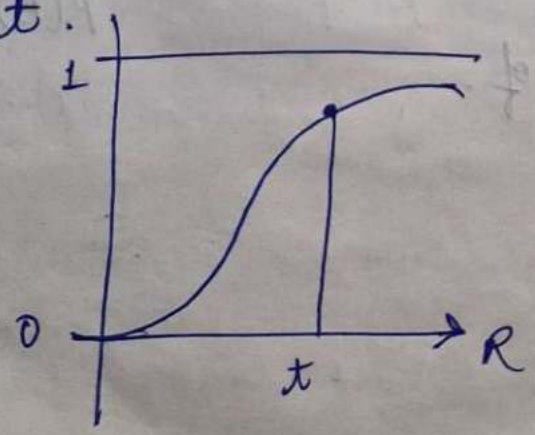
$Val(X) = X(\Omega)$

Commulative Prob. distribution

$F_X(x) = P(X < t)$

$P\left[\{\omega : X(\omega) < t\}\right]$

Basically it means to get the set of all ω such that $X(\omega) < t$ (lies below t). Get A set of those events and then measure of Prob. on that event.



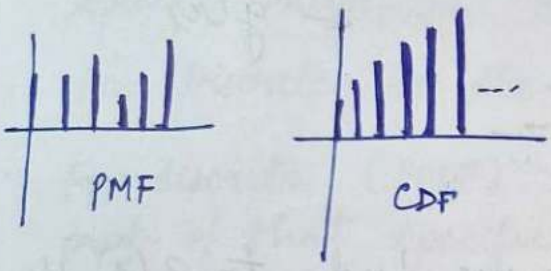
Discrete vs Continuous RV

Cummulative distribution func. (CDF)

→ ① Probability mass function (PMF) — Discrete

→ ② Probability density function (PDF) — Continuous.

PMF



Discrete RV

Val(X) countable

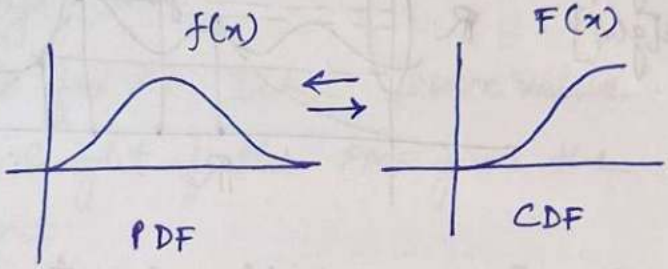
$$P(X=k) = P(\{\omega \mid X(\omega)=k\})$$

PMF

$$p(x): Val(X) \to [0,1]$$

$$\sum_{x \in Val(x)} p_x(x) = 1$$

PDF



$$\frac{d}{dx} F(x) = f(x)$$

or

$$\int_{-\infty}^{x} f(x)\,dx = F(x)$$

Continuous RV

Val(X) uncountable

$$P(a \le X \le b) = P(\{\omega \mid a \le X(\omega) \le b\})$$

PDF

$$f_x : R \to R$$

$$f_x(x) = \frac{d}{dx} F_x(x)$$

$$f_x(x) \neq P(X=x)$$

$$\int_{-\infty}^{\infty} f_x(x)\,dx = 1$$

$$P(x \le X \le x+dx$$

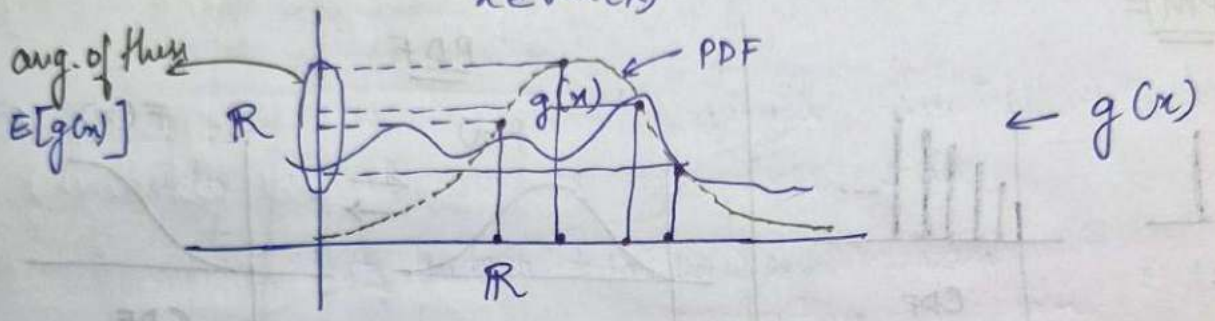
Expected value and Variance

$g : R \to R$

Exp value

Let X be a discrete RV with PMF p_X

$$E[g(X)] = \sum_{x \in Val(X)} g(x)\, p_x(x). \quad \text{(discrete)}$$



$E[g(x)]$: If the input that we feed to $g(x)$ is a RV which means the actual input fed is random the expectation of $g(x)$ is sum over every possible x that can be fed as input to g. Multiplied by the prob. that x can be fed.

For continuous just take integral.

- Basically what is the avg. value / Exp. value of $g(x)$ we get after feeding the input.

for continuous RV with PDF f_x

$$E[g(X)] = \int_{-\infty}^{\infty} g(x)\, f_x(x)\, dx$$

$$\# \quad \boxed{\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} g(x^{(i)})} \rightarrow \int_{-\infty}^{\infty} g(x)\, p(x)\, dx.$$

Monte Carlo Estimate

Law of large Numbers

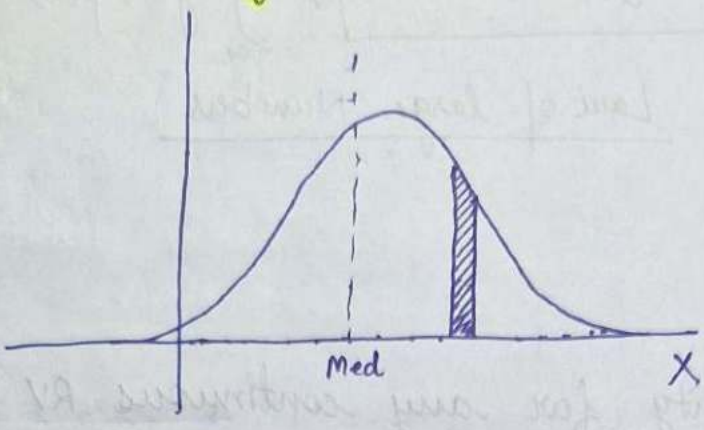
Note

- The Probability for any continuous RV takes a specific value is always 0.
- For Discrete RV the Prob. for RV takes some value.
- For discrete (PMF) the height of the PMF gives the prob. of that specific input.
- But PDF i.e prob. density function is different as the light it not the prob. We can only find the probability in the intervals and not at specific point.
- Expectation is basically what this ~~value~~ RV takes the value on average.
- Variance is the measure of how spread apart is the distribution in general or how spread apart is this RV around its mean.

$$Var(X) = E\left[(X - E[X])^2\right]$$

$$Var(X) = E[X^2] - E[X]^2$$

Lecture 3 - Probability and Statistics



- **Median** - It is the point which divides the mass into equal masses on either side
- For symm. distribution $E(X)$ = Median
  expectation = Median

Two Random Variable.

Binariate CDF

$$F_{XY}(x,y) = P(X \le x, Y \le y)$$

Binariate PMF

$$P_{XY}(x,y) = P(X=x, Y=y)$$

Joint Probability

- **Joint Prob. Mass function** -

Joint(PMF) $f: R^n \longrightarrow [0,1]$

$$f(x_1, \dots, x_n) = P(X_1 = x_1, \dots X_n = x_n)$$

for $n=2$

$$f(x,y) = P(X=x, Y=y)$$

CDF : $F: R^n \longrightarrow [0,1]$

$$F(x_1, \dots x_n) = P(X_1 \le x_1, \dots, X_n \le x_n)$$

• Marginal Probability mass function

$X, Y \quad f(x,y) = P(X=x, Y=y)$

$\{X=x, Y=y\} \qquad \{X=x\} \longrightarrow P(X=x) = f_X(x)$

(Marginal Prob.)

\# X and Y are two discrete RV with JP mass function $f(x,y)$. The prob. mass func. of only X is $f_X(x)$ and is called marginal prob. mass func. of X. It basically doesn't consider what is Y but find prob for X only.

\# Similarly for Y : $f_Y(y) \rightarrow$ MPMF of Y.

Range $(X) = \{x_1, \cdots, x_n\}$ and Range $(Y) = \{y_1, \cdots, y_m\}$

mar Prob : $f_X(x_i) = \sum_{j=1}^{m} f(x_i, y_j)$ for $i = 1, \cdots, n$

$f_Y(y_j) = \sum_{i=1}^{n} f(x_i, y_j)$ for $j = 1, \cdots, m$

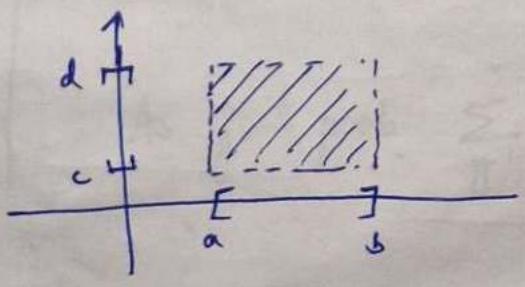
• Joint Probability density function

$P(a \le X \le b) = \int_a^b f(x)\,dx \qquad P(X=x) = 0$

$X, Y \qquad P(X=x, Y=y) = 0$

$P(a \le X \le b, c \le y \le d)$

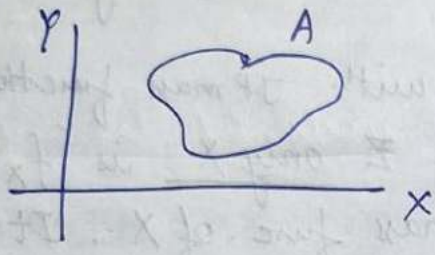
$\{a \le x \le b, c \le y \le d\}$



$$P(a \le X \le b) = \int_a^b f(x)\,dx$$

$$\Rightarrow P(a \le X \le b,\ c \le Y \le d) = \int_a^b \int_c^d f(x,y)\,dy\,dx$$

Generalization



Event: $\{(X,Y) \in A\}$

$$P((X,Y) \in A) = \int_A f(x,y)\,dx\,dy$$

or

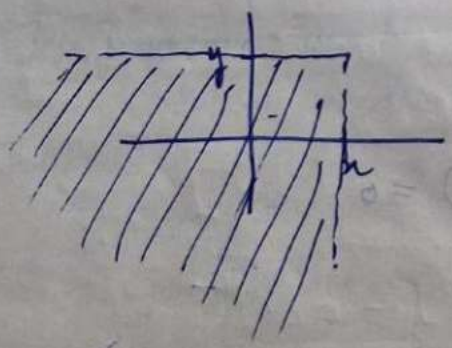
$$\iint_A f(x,y)\,dx\,dy$$

$f: \mathbb{R}^2 \to \mathbb{R} \Rightarrow$ for all $A \in \mathbb{R}^2$

CDF

$$F(x) = P(X \le x) = \int_{-\infty}^{x} f(u)\,du \rightarrow f(x) = \frac{dF}{dx}$$

CDF: $F(X \le x, Y \le y) = \int_{-\infty}^{x} \int_{-\infty}^{y} f(u,v)\,du\,dv$



$$\frac{\partial^2 F}{\partial x\,\partial y} = f(x,y)$$

Joint CDF

Joint PDF

• Marginal Probability density function

$f(x,y) \quad \{a \le X \le b, c \le Y \le d\} \quad \int_a^b \int_c^d f(x,y)\, dy\, dx$

$\{a \le X \le b\} \quad \int_a^b f_X(x)\, dx$

Now marginal of X and Y

$$f_X(x) = \int_{-\infty}^{\infty} f(x,y)\, dy$$

$$f_Y(y) = \int_{-\infty}^{\infty} f(x,y)\, dx$$

Bayes's Theorem

$$p(x,y) = p(x) \cdot p(y|x) \quad \text{——①}$$

Joint Marginal Conditional

$$p(x,y) = p(y) \cdot p(x|y) \quad \text{——②}$$

Equate ① and ②

$$p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)}$$

→ Given Cond. prob. $p(x|y)$
→ we find reverse cond. prob. $p(y|x)$ using Bayes's theorem

$$p(y|x) = \frac{p(x|y) \cdot p(y)}{\sum_{y'} p(y') \cdot p(x|y')}$$

As joint prob. $\sum_{y'} p(x,y') = p(x)$

for continuous X and Y

$$f(y|x) = \frac{f(x|y)\,f(y)}{f(x)}$$

where: $f(x) = \int_{y' \in \text{value } y} f(x|y')f(y')\,dy'$

Independence

$P(A \cap B) = P(A) \cdot P(B)$

$\underbrace{F(a,b)}_{\text{joint CDF}} = \underbrace{F_X(a)}_{\text{marginal CDFs}} \cdot \underbrace{F_Y(b)}_{\text{marginal CDFs}}$

after diff.

$$f(x,y) = f_X(x) \cdot f_Y(y)$$

Expectation

X, Y → continuous RV

g, R2 → R

$$E(g(x,y)) = \int_{x \in \text{Val}(x)} \int_{y \in \text{Val}(y)} g(x,y)\, f_{XY}(x,y)\,dx\,dy$$

Ex

$g(x,y) = 3x$

$f_{XY} = 4xy$

$0 < x < 1$

$0 < y < 1$

$$E = \int_0^1 \int_0^1 3x \cdot 4xy\,dx\,dy$$

$$12 \int_0^1 \int_0^1 x^2 y\,dx\,dy$$

$$12 \int_0^1 \frac{x^3}{3}\Big|_0^1 y \Rightarrow \int_0^1 4y\,dy = 2y^2\Big|_0^1$$

$\Rightarrow (2)$

Covariance of two Random variable X and Y

$$Var(X) = E[X^2] - (E[X])^2$$

$$\boxed{Cov(X,Y) = E[XY] - E[X]E[Y]}$$

$$Cov(X,X) = Var(X)$$

If X and Y independent then

$$E(XY) = E(X) \cdot E(Y) \implies Cov[X,Y] = 0$$

$$Var[X+Y] = [E(X+Y)]^2 - E((X+Y)^2)$$

simple proof

$$Var[X+Y] = Var[X] + Var[Y] + 2\,Cov[X,Y]$$

Multivariant Gaussian (Normal) distribution

$x \in \mathbb{R}^n$

$$\boxed{P(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]}$$

Σ: Covariance matrix
and it is PSD (Positive semi definite). Its Full Rank

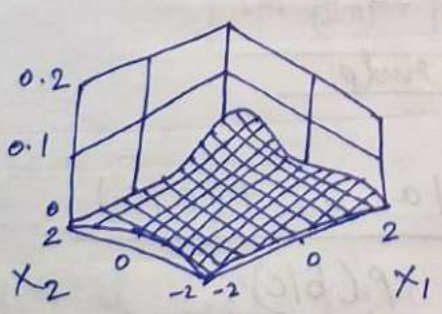
$|\Sigma|$: Determinant

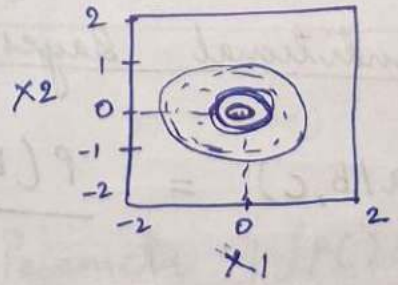
The exp form is of the quadratic form: V^TAV

Ex

$\Sigma = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$

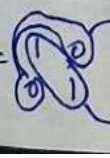
$\mu = [0\ 0]^T$





Heat map.

$\mu[0\,0]^T$ → mean the center of distribution is at 0, 0

$\Sigma =$ 

→ Covariance b/w RV, cov=0 mean no strong rel between X_1, X_2 (correlation)

→ Variances of the marginal (variation)

Conditional expectation

$E[X]$ = constant

$E[X|Y]$ = random variable

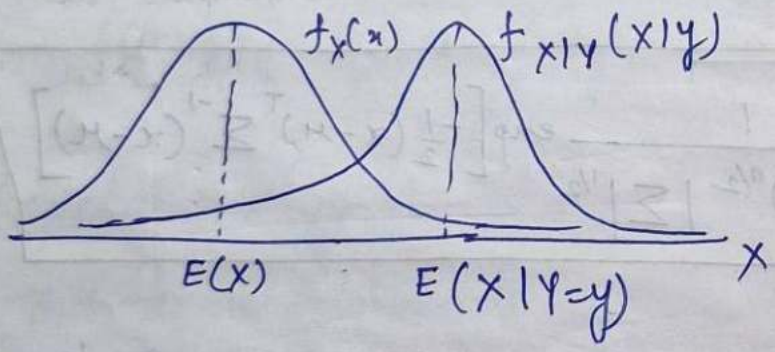
$E[X|Y=y]$ = function of y

X, Y $\quad\quad$ $Y=y$ (partial info).

$$f_X(x) \longrightarrow f_{X|Y}(x|y) = \frac{f(x,y)}{f_Y(y)}$$

- Pdf change
- Exp will also change

$$E(X) = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dy \longrightarrow E(X|Y=y) = \int_{-\infty}^{\infty} x \cdot f_{X|Y}(x|y)\, dx$$



Hence

Law of total expectation

$$E[X] = E\big[E[X|Y]\big]$$

Conditional Bayes rule

$$P(a|b,c) = \frac{P(b|a,c)\,P(a|c)}{P(b|c)}$$

Statistics in ML

Parameters

μ, Σ

- Method of Moment
- Maximum Likelihood estimation

Observations / Data

$x \in \mathbb{R}^n$

Training data
(x, y)
⋮

Maximum likelihood estimation (MLE)

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

$x^{(i)} \rightarrow i^{th}$ example of $x \in \mathbb{R}^d$

Gaussian Data

$x \in \mathbb{R}^d$

{ sampled independently & identically distributed }

Probability density

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right]$$

$$p(x^{(1)}, \ldots, x^{(n)}) = \prod_{i=1}^{n} p(x^{(i)})$$

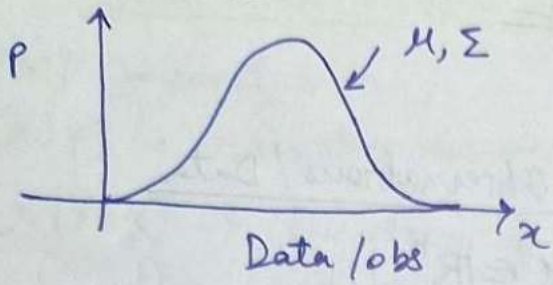
$$p(x^{(1)}, \ldots, x^{(n)}; \mu, \Sigma) = \prod_{i=1}^{n} \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x^{(i)}-\mu)^T \Sigma^{-1} (x^{(i)}-\mu)\right]$$

Likelihood: $L(\mu, \Sigma; x^{(1)}, \ldots, x^{(n)}) =$,, (same)

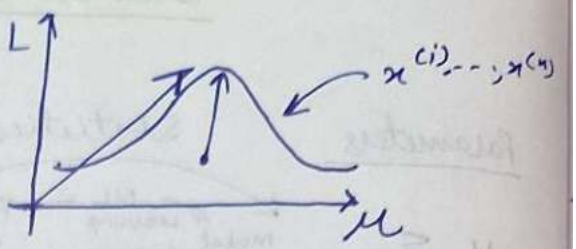
$p \rightarrow$ Probability of data given Parameter | $\int p(\,)\,dx = 1$

$L \rightarrow$ Likelihood of parameter given data | $\int L(\,) =$ nothing

Prob



Likelihood



- We use subscript $\hat{}$ over things that are estimated as a output of some estimation process

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

$$\hat{\theta}_{MLE} = \arg\max_{\theta} \prod_{i=1}^{n} L(\theta; x^{(i)})$$

$$= \arg\max_{\theta} \log \prod_{i=1}^{n} L(\theta; x^{(i)})$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} \ell(\theta; x^{(i)}) \quad \{\ell(\theta) = \log L(\theta)\}$$

For Gaussian

$$\hat{\mu}, \hat{\Sigma} = \arg\max_{\mu, \Sigma} \sum_{i=1}^{n} \log \left[\frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left[\frac{-1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu) \right] \right]$$

$$= \arg\max_{\mu, \Sigma} \left[\sum_{i=1}^{n} \left(K - \frac{1}{2} \log |\Sigma| - \frac{1}{2} (x^{(i)} - \mu)^T \Sigma^{-1} (x^{(i)} - \mu) \right) \right]$$

To find arg max tak diff.

μ

$$\nabla_\mu \sum_{i=1}^{n} K - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(x^{(i)}-\mu)^T\Sigma^{-1}(x^{(i)}-\mu)$$

$$\nabla_\mu \sum_{i=1}^{n} -\frac{1}{2}\left[x^{(i)T}\Sigma^{-1}x^{(i)} - x^{(i)T}\Sigma^{-1}\mu - \mu^T\Sigma^{-1}x^{(i)} + \mu^T\Sigma^T\mu\right]$$

$$\nabla_\mu \sum_{i=1}^{n} -\frac{1}{2}\left[-2(\Sigma^{-1}x^{(i)})^T\mu + \mu^T\Sigma^{-1}\mu\right]$$

$$\sum_{i=1}^{n}\left(\Sigma^{-1}x^{(i)} - \Sigma^{-1}\mu\right) = 0$$

$$n\Sigma^{-1}\mu = \sum_{i=1}^{n}\Sigma^{-1}x^{(i)}$$

$$\Sigma^{-1}\mu = \Sigma^{-1}\left[\frac{1}{n}\sum_{i=1}^{n}x^{(i)}\right]$$

$$\boxed{\mu = \frac{1}{n}\sum_{i=1}^{n}x^{(i)}}$$

Σ

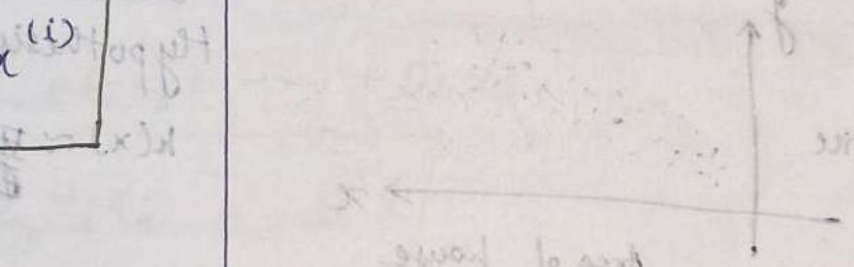
$$S = \Sigma^{-1}, \quad \nabla_S \ell = 0 \iff \nabla_{\Sigma^{-1}}\ell = 0$$

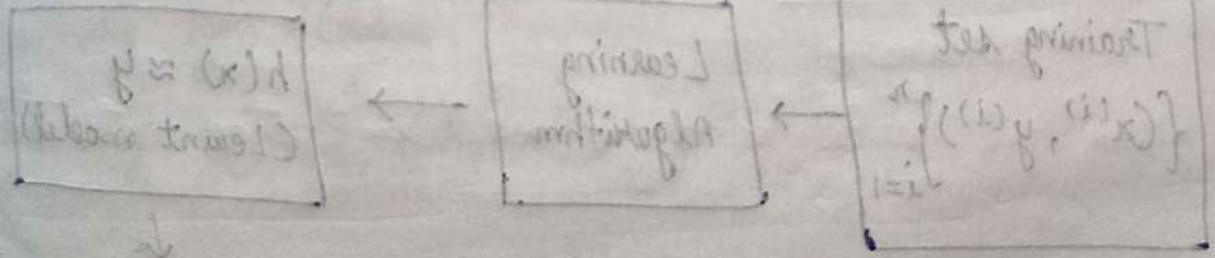
$$\nabla_S \sum_{i=1}^{n}\frac{1}{2}\log|S| - \frac{1}{2}\left((x^{(i)}-\mu)^T S(x^{(i)}-\mu)\right)$$

$$\frac{1}{2}\left[nS^{-1} - \sum_{i=1}^{n}(x^{(i)}-\mu)(x^{(i)}-\mu)^T\right) = 0$$

$$\boxed{S^{-1} = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)}-\mu)(x^{(i)}-\mu)^T}$$

similar $\approx E\left[(X - E[X])^2\right]$





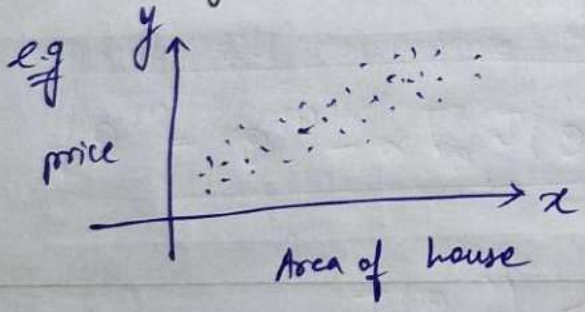
Lecture 4 - Linear Regression

Supervised learning

$X \longrightarrow Y$ mapping
(Input) (output)

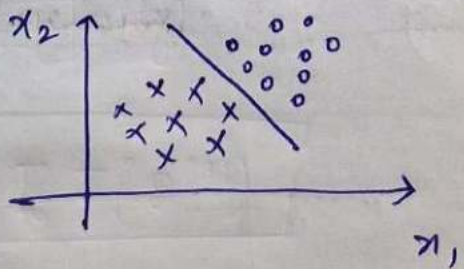
n - No. of (x, y) examples in training set
d - $x \in \mathbb{R}^d$ dimension of input
$x^{(i)}$ - i^{th} example input
$y^{(i)}$ - i^{th} example output
$(x^{(i)}, y^{(i)})$ - i^{th} example

Regression (prediction)

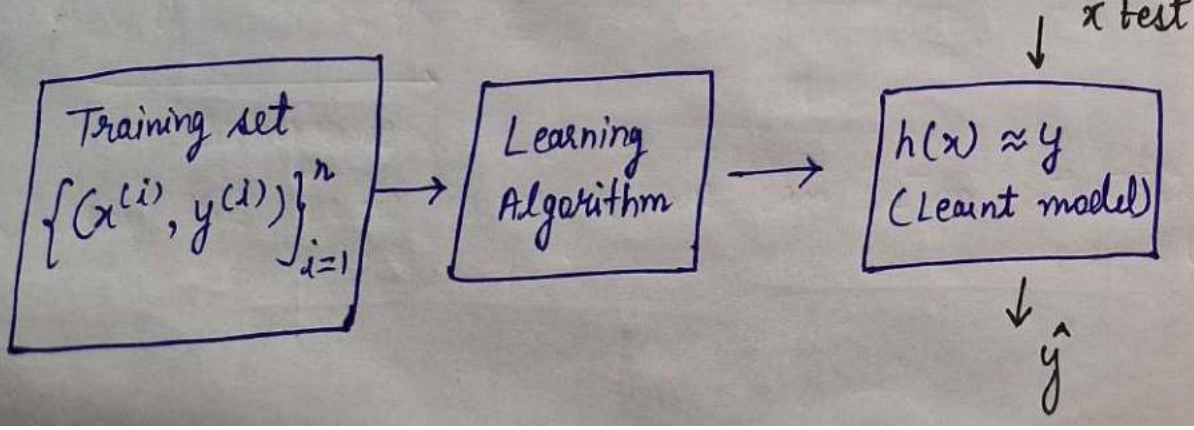


Learn
Hypothesis
$h(x) \approx$ Y

Classification



Flowchart of Approach



Linear Regression

$x \in \mathbb{R}^d$, $y \in \mathbb{R}$, n such examples

The hypothesis that we want to learn has some parameter θ. The goal of linear regression is to learn a suitable set of parameters θ that makes the y value as close as possible to $h_\theta(x)$.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_d x_d$$

$$\theta \in \mathbb{R}^{d+1}$$

$$h_\theta(x) = \left(\sum_{i=1}^{d} \theta_i x_i \right) + \theta_0$$

$\boxed{x_0 = 1}$ – intercept term

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_d x_d$$

$$\boxed{h_\theta(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x}$$

Cost function / Loss function

It is simply the amount of displeasure a specific hypothesis causes to us.

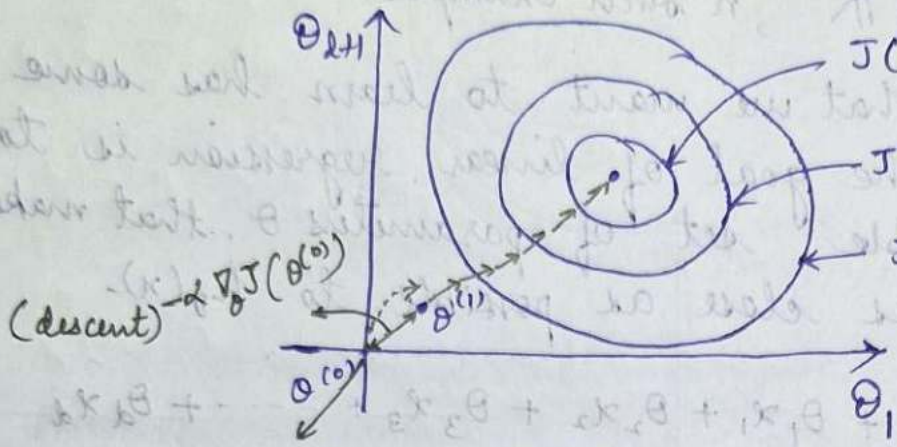
$$\boxed{J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2}$$

squared error

$$\hat{\theta} = \arg\min_\theta J(\theta)$$

$$= \arg\min_\theta \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Algorithm : Gradient Descent



Contour plot of $J(\theta)$

$\theta^{(0)}$ = Initialization

$$\theta_j^{(1)} = \theta_j^{(0)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta^{(0)})$$

α - Learning rate

Repeat till convergence

$$\theta^{(1)} = \theta^{(0)} - \alpha \nabla_\theta J(\theta^{(0)})$$

$$J(\theta^{(1)}) < J(\theta^{(0)})$$

(hopefully)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J(\theta^{(t)})$$

$\theta^{(0)}, \theta^{(1)}, \cdots, \theta^{(t)}, \theta^{(t+1)}$

$\longrightarrow$ Convergence

- When this series converges, we can stop iterating

check for convergence

$\| \theta^{(t)} - \theta^{(t-1)} \|$, $\| J(\theta^{(t)}) - J(\theta^{(t-1)}) \|$

\# If the norm of θ is too small

\# If the loss has going down

or

$\| \nabla_\theta J(\theta^{(t)}) \|$

\# If the norm of the gradient is too small.

Gradient descent on linear regression.

$\theta^{(0)} =$ ——— (init)

Repeat until convergence

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta J(\theta^{(t)})$$

$$= \theta^{(t)} - \alpha \nabla_\theta \left[\frac{1}{2} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right]$$

$$= \theta^{(t)} - \alpha \nabla_\theta \left[\frac{1}{2} \sum_{i=1}^{n} \left(\theta^T x^{(i)} - y^{(i)} \right)^2 \right]$$

$$= \theta^{(t)} - \alpha \left[\frac{1}{2} \sum_{i=1}^{n} 2 \left(\theta^T x^{(i)} - y^{(i)} \right) x^{(i)} \right]$$

$$\boxed{\theta^{(t+1)} = \theta^{(t)} - \alpha \left[\sum_{i=1}^{n} \left(\theta^T x^{(i)} - y^{(i)} \right) x^{(i)} \right]}$$

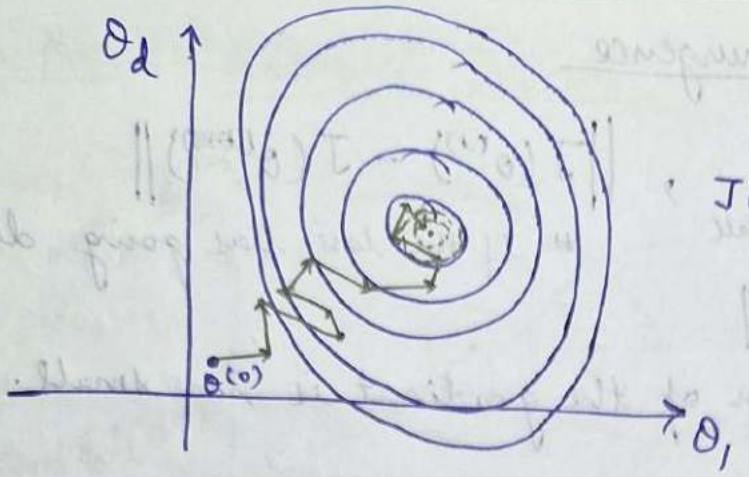
Stochastic Gradient Descent (SGD)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_\theta \tilde{J}(\theta)$$

where,

$$\tilde{J}(\theta) = \frac{1}{2} \left(\theta^T x^{(K)} - y^{(K)} \right)^2$$

∴ K uniformly at random sampled from training set.



In GD we perform descent in a proper manner i.e approaching towards minima with each step using the loss function which is considering every example. Meanwhile in SGD we randomly pick an example and follow gradient descent using that. That is why it is noisy but saves a lot of expense / memory as we are not considering all examples.

Eventually using SGD we reach a region near minima where it is not a lot noisy.

Computational cost of SGD <<< GD. Hence we can trade of No. of steps to reach minima with computation cost.

GD and SGD are numerical algorithms (iterative algos) gives a value of α as output.

cost function - $$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

Design Matrix ($d+1$ dimension)

$$X = \begin{bmatrix} - x^{(1)} - \\ \vdots \\ - x^{(n)} - \end{bmatrix} \uparrow n \downarrow \qquad \vec{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \uparrow n \downarrow$$

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_{d+1} \end{bmatrix} \uparrow d+1 \downarrow$$

$X\theta - Y$

$\mathbb{R}^{n\times(d+1)} \quad \mathbb{R}^{(d+1)} \to \mathbb{R}^n$

$\mathbb{R}^n - \mathbb{R}^n \Rightarrow \boxed{\mathbb{R}^n}$

$$X\theta - Y = \begin{bmatrix} x^{(1)T}\theta - y^{(1)} \\ x^{(1)T}\theta - y^{(i)} \\ \vdots \\ x^{(n)T}\theta - y^{(n)} \end{bmatrix}$$

$$\Rightarrow J(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(\theta^T x^{(i)} - y^{(i)}\right)^2 = \underbrace{\boxed{\frac{1}{2}(X\theta - Y)^T(X\theta - Y)}}_{\text{vector form}}$$

$$\Rightarrow \nabla_\theta J(\theta) = 0$$

$$\nabla_\theta \frac{1}{2}(X\theta - Y)^T(X\theta - Y)$$

$$\nabla_\theta \frac{1}{2}\left[(X\theta)^T X\theta - (X\theta)^T Y - Y^T(X\theta) + Y^T Y\right]$$

$$\nabla_\theta \frac{1}{2}\left[\underbrace{\theta^T(X^TX)\theta}_{\text{Quadratic form}} - \underbrace{2\theta^T(X^TY)}_{\text{Vec x Vec = scalar}} + \underbrace{Y^TY}_{\text{scalar}}\right]$$

$$\nabla_\theta \frac{1}{2}[-] = \frac{1}{2}\left[2(X^TX)\theta - 2X^TY\right] = 0$$

$$(X^TX)\theta = X^TY \quad \text{Normal equation}$$

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

Exact solution for cal. θ / min cost func.

(only if X^TX invertible)

Probabilistic Interpretation

In this we make an assumption that $y^{(i)}$ is generated using this approach.

$$y^{(i)} = \underbrace{\theta^T}_{\text{Unknown Vector that we are calculating}} \underbrace{x^{(i)}}_{\text{Eg. Features (price of house)}} + \underbrace{\varepsilon^{(i)}}_{\text{Random Gaussian noise distributed acc. to gaussian variable.}}$$

$$\varepsilon^{(i)} \sim N(0, \sigma^2)$$

$$\varepsilon^{(i)} = y^{(i)} - \theta^T x^{(i)} \sim N(0, \sigma^2)$$

Property of Gaussian noise

$$\hookrightarrow \quad y^{(i)} - \theta^T x^{(i)} \sim N(0, \sigma^2)$$

$$y^{(i)} \sim N(\theta^T x^{(i)}, \sigma^2) \qquad (\text{I.I.D})$$

$$\Rightarrow \boxed{p(y^{(i)} | x^{(i)}; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-1}{2}\left(\frac{y^{(i)} - \theta^T x^{(i)}}{\sigma^2}\right)^2\right]}$$

Data

X, Y

Parameters

μ, σ^2

θ, σ^2

$$\log L(\theta) = \log \prod_{i=1}^{n} p(y^{(i)} | x^{(i)}; \theta)$$

$$l(\theta) = \sum_{i=1}^{n} \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-1}{2}\frac{(y^{(i)} - \theta^T x^{(i)})^2}{\sigma^2}\right]\right]$$

$$= K - \frac{1}{2\sigma^2}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

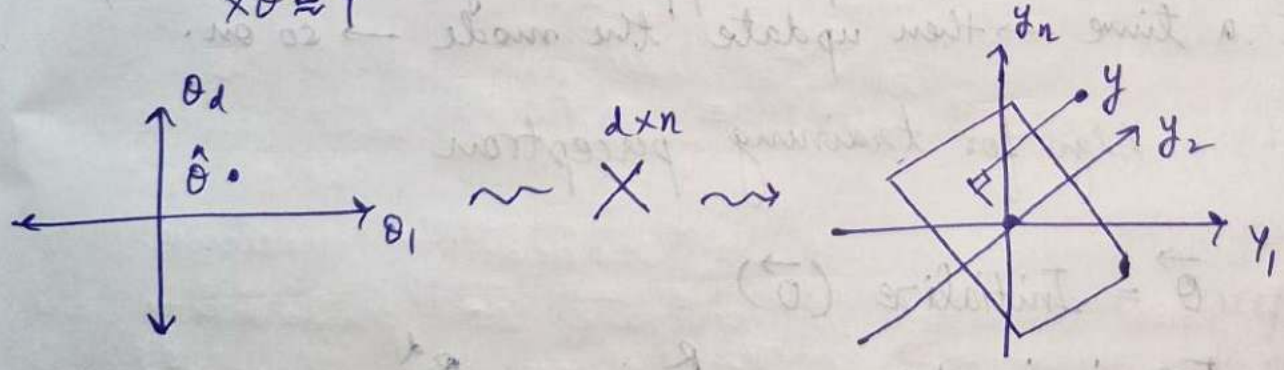
$$l(\theta) = nK - \frac{1}{\sigma^2}\left[\sum_{i=1}^{n} \frac{1}{2}\left(y^{(i)} - \theta^T x^{(i)}\right)^2\right]$$

$$\arg\max_{\theta} l(\theta) = \arg\min_{\theta} J(\theta).$$

* The log likelihood function by making a probabilistic assumption about the noise is gonna give us the -ve of the scaled version of the original cost function
* which means by performing max. likelihood, we are minimizing the squared error.

Another view : Projection Interpretation

$X\theta \approx Y$



$X(X^TX)^{-1}X^T$ (Projection matrix)

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

* We want $X\theta \approx \vec{y}$ but its never the case, so we use projection $X\theta \approx \hat{y}$

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

Lecture 5 - Perceptron and Logistic Regression

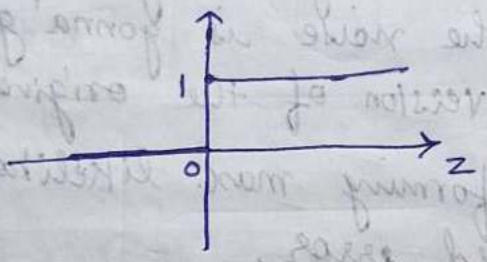
Perceptron algorithm

Simple classification algorithm

$$x^{(i)} \in \mathbb{R}^d, \quad y^{(i)} \in \{0, 1\}$$

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

$$g(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$



\# It is also called a <u>streaming algorithm</u> because it is designed to work in a setting where you are encountering examples one after another. We do not have all examples upfront, given one example at a time → then update the model → so on.

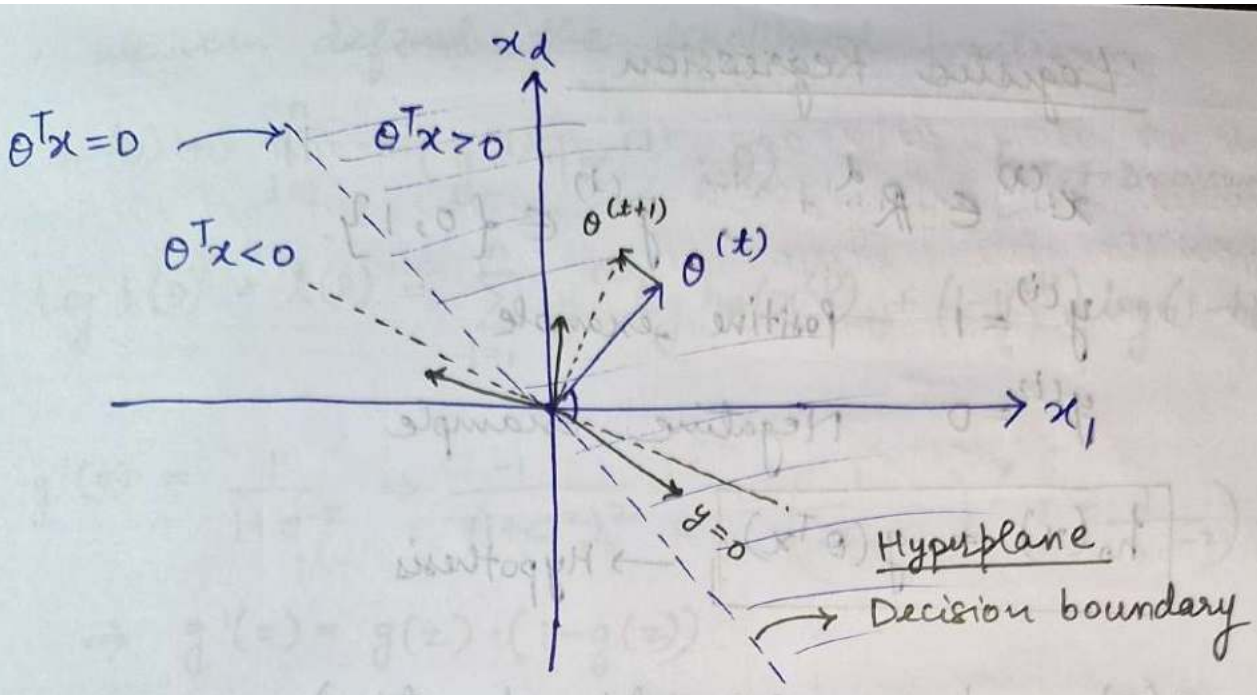
Algo for training perceptron

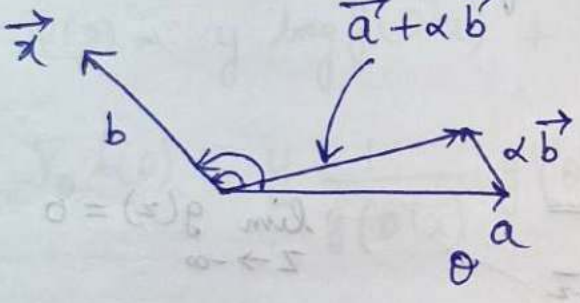
$\vec{\theta}$ = Initialize $(\vec{0})$

For i in 1, 2, ... R

$$\underbrace{\theta}_{\mathbb{R}^d} = \theta + \alpha \underbrace{(y^{(i)} - h_\theta(x^{(i)}))}_{\mathbb{R}} \underbrace{x^{(i)}}_{\mathbb{R}^d} \quad \text{(update rule)}$$

\# This looks similar to update rule of lin. regression but is actually diff. since hypothesis is diff. It is $g(\theta^T x)$ and not $\theta^T x$.





$a^T b < 0$ at present

$\Rightarrow$ So how do we make it positive

$$a^T(b + \alpha a) = a^T b + \alpha \underbrace{a^T a}_{\text{always +ve}}$$

$$\Rightarrow a^T b + \alpha a^T a \geq a^T b$$

- Similarly, we can do for -ve.
- If $y^{(i)} = h_\theta(x^{(i)})$ means classification is done correctly, therefore no need to update θ.
- If you are given a dataset where you are given that the classes are linearly separable then this perceptron algorithm will find the separating hyperplane.

Logistic Regression

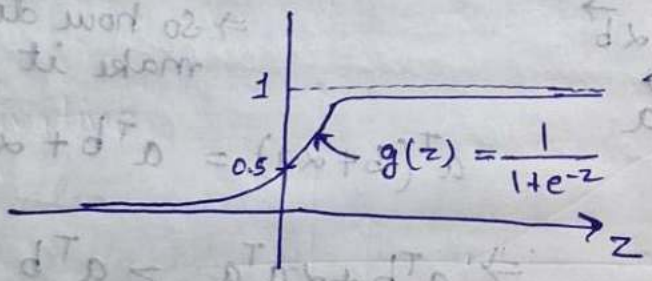
$x^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \{0, 1\}$

$y^{(i)} = 1$ Positive example

$y^{(i)} = 0$ Negative example

$$\boxed{h_\theta(x) = g(\theta^T x)} \rightarrow \text{Hypothesis}$$

$$\Rightarrow g(z) = \frac{1}{1+e^{-z}} \quad \text{(logistic function)}$$



$\lim_{z \to -\infty} g(z) = 0$

$\lim_{z \to \infty} g(z) = 1$

$$P(y^{(i)} = 1 \mid x^{(i)}; \theta) = h_\theta(x)$$

$$P(y^{(i)} = 0 \mid x^{(i)}; \theta) = 1 - h_\theta(x)$$

* Logistic regression is a probability machine. For any given example it outputs a probability and then it is upto us to choose a threshold and then compare the probability that it outputted to that and make decision.

$$\boxed{P(y \mid x; \theta) = [h_\theta(x)]^y \times [1 - h_\theta(x)]^{1-y}}$$

Similar to Bernoulli distribution

We can defind the likelihood function.

$$L(\theta) = \prod_{i=1}^{n} P(y^{(i)} | x^{(i)}; \theta) \quad [\text{I.I.D assumption}]$$

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

$$g'(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{-1}{(1+e^{-z})^2} \cdot e^{-z} = \left(\frac{1}{1+e^{-z}}\right)\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\Rightarrow g'(z) = g(z) \cdot (1-g(z))$$

$$l(\theta) = y \log g(\theta^T x) + (1-y) \log (1-g(\theta^T x))$$

$$\nabla_\theta l(\theta) = y \cdot \frac{1}{g(\theta^T x)} \cdot g'(\theta^T x) \cdot x + (1-y) \frac{1}{1-g(\theta^T x)} \cdot (-1) g'(\theta^T x) \cdot x$$

$$= y \cdot \frac{g(\theta^T x)(1-g(\theta^T x)) \cdot x}{g(\theta^T x)} + \frac{(1-y)}{(1-g(\theta^T x))} \cdot (-1)(g(\theta^T x)) \cdot (1-g(\theta^T x)) \cdot x$$

$$\nabla_\theta l(\theta) = y(1-g(\theta^T x)) \cdot x + (1-y)(-1) g(\theta^T x) \cdot x$$

$$= x\left[y(1-g(\theta^T x)) - (1-y) g(\theta^T x) \right]$$

$$= \left[y - y g(\theta^T x) - g(\theta^T x) + y g(\theta^T x) \right] \cdot x$$

$$\nabla_\theta l(\theta) = [y - h_\theta(x)] \cdot x$$

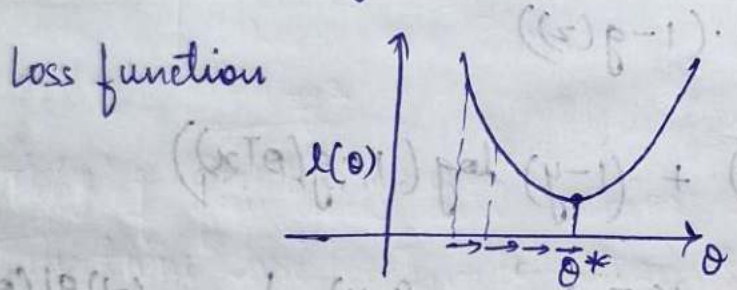
$$\boxed{\theta = \theta + \alpha [y - h_\theta(x)] x}$$

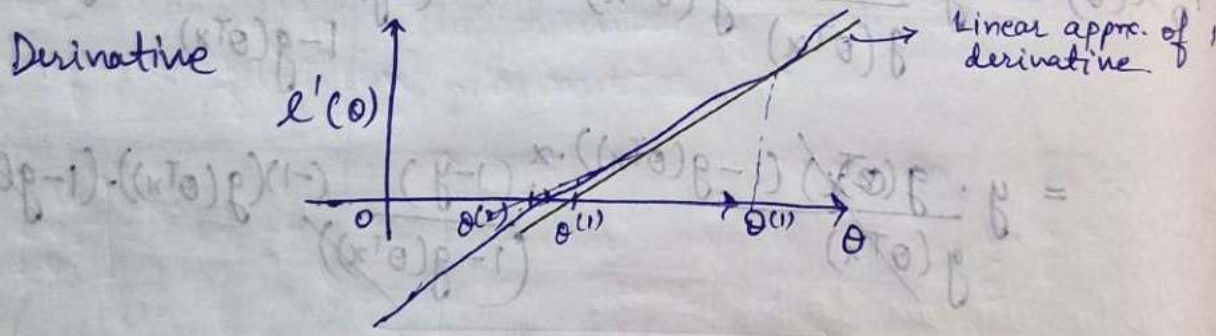
Note

- We are using $(y|x)$ and not (y,x) because when the model is deployed to a system we get the input from the user hence we use y given x. But we will be using (y,x) too later.

Newton's Method

- Alternative to gradient descent (GD).



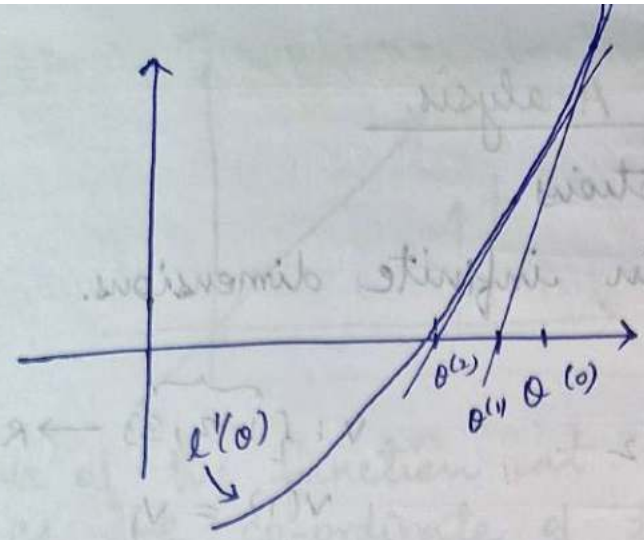


Newton's Method is a way in which for any given function $f(x)$, it finds the point where $f(x)=0$. (value of input).

It is basically a root finding method. Finds the input where $f(x)=0$.

$\theta^{(0)}, \theta^{(1)}, \theta^{(2)}, \ldots \theta^*$

The rate at which newton's method converges is much much faster than the regular gradient descent.



$$\theta^{(t+1)} = \theta^{(t)} - \frac{f(\theta^{(t)})}{f'(\theta^{(t)})}$$

$$\theta^{(t+1)} = \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}$$

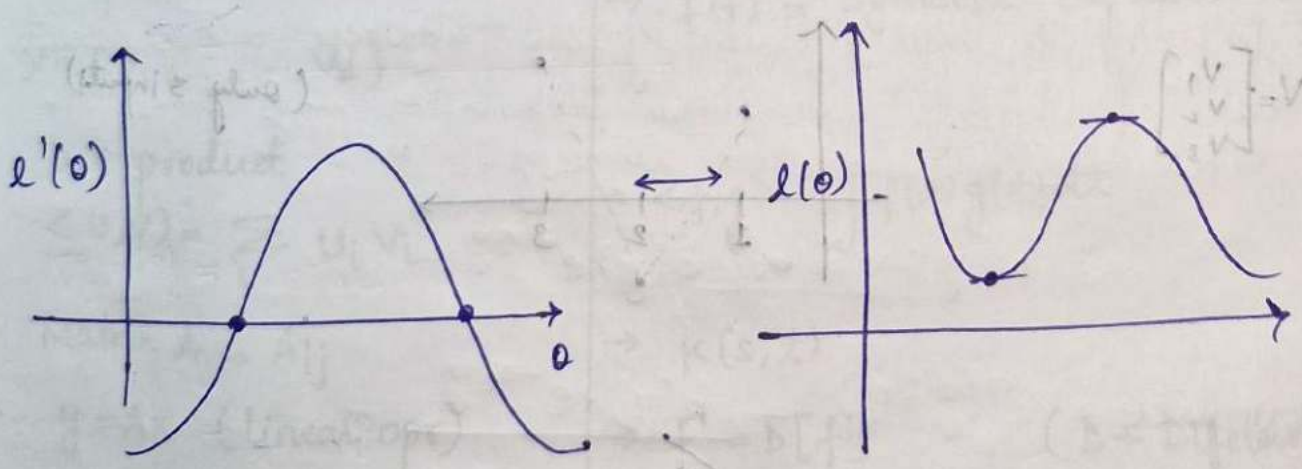
scalar θ

For vector θ

$$\theta^{(t+1)} = \theta^{(t)} - \alpha H^{-1} \nabla_\theta l(\theta^{(t)})$$

Newton Raphson method

H = Hessian of loss $l(\theta)$

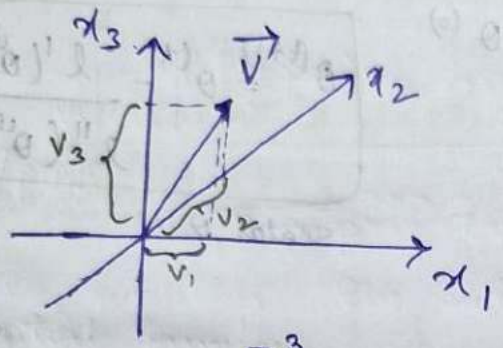


- Newton's method will take you to the nearest stationary point of that function.
- It works well with a convex function than concave function

Functional Analysis

⇒ Study of functions

⇒ Linear algebra in infinite dimensions.



$\mathbb{R}^3$

$v : \{1, 2, 3\} \to R$

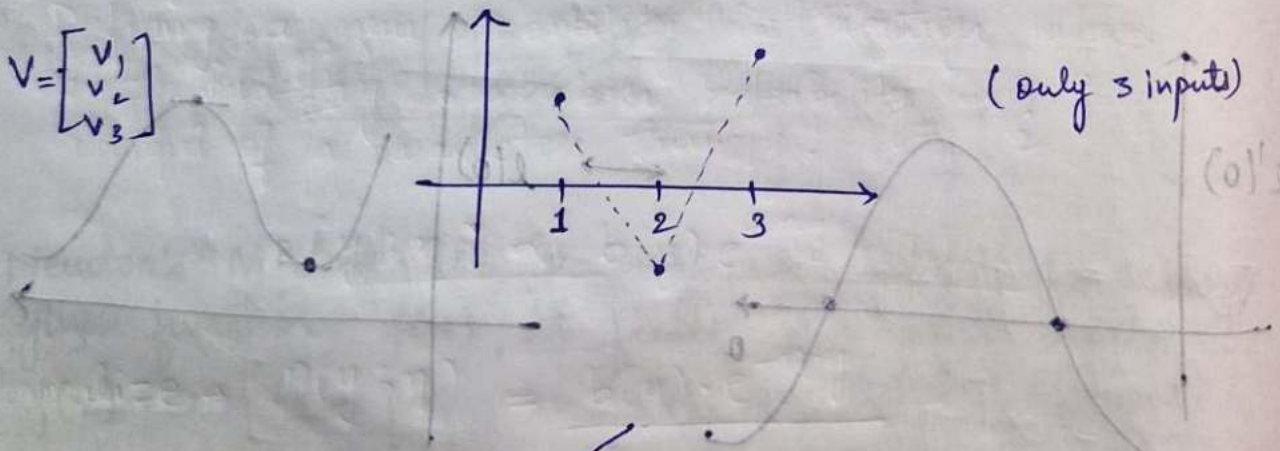
$v(1) = v_1$

$v(2) = v_2$

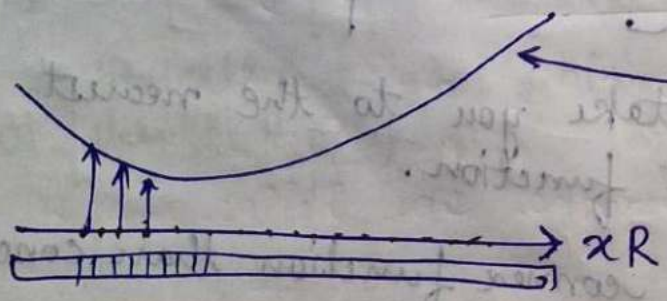
$v(3) = v_3$

- Think of functions and vectors as the same.
- Functions are infinite dimensional vector.
- In the vector, the number of axis is the domain of the function.

$V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$



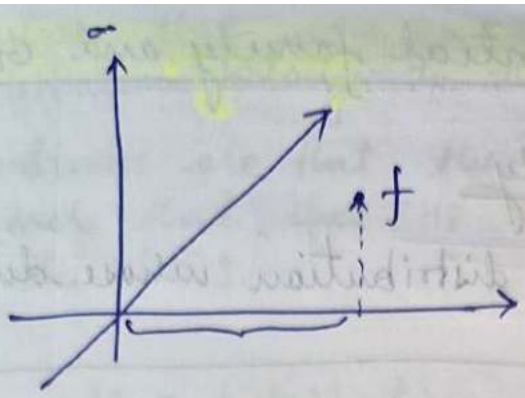
(only 3 inputs)



(Infinite inputs)

$f = \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ \infty \end{bmatrix}$

*// Basically functions are points in infinite dimensional space.



← Infinite No. of axis.

- The value of the function at a given input is the same as the co-ordinate of the vector at that axis. Same analogy as of vector, just extends it to infinite dimensions.

Finite

→ Vector $\vec{v}$

→ Index $\{1, \cdots, d\}$

→ Components

→ Explicit $\mathbb{R} \in D$

→ $V = \{v_1, \cdots, v_d\}$

→ Dot product

$$\langle U, V \rangle = \sum_i U_i V_i$$

→ Matrix $A \to A_{ij}$

→ $y = Ax$ (Linear opr)

→ Eigen Vector A

$Ax = \lambda x$

→ $\vec{u} = A\vec{v}$

$$u_i = \sum_j A_{ij} V_j$$

Infinite

→ function $f(t)$

→ Domain $\mathbb{R}$

→ Values

→ $f(t)$ = Symbolic Representation

→ $\langle f, g \rangle = \int f(t) g(t) dt$

→ $K(s, t)$

→ $f' = D[f]$ ($D \to$ Differentiation)

→ $f' = D[f] = \lambda f$

$$D[C^{Kt}] = Ke^{Kt}$$

→ $g(s) = \int K(s, t) f(t) dt$

Lecture 6 - Exponential family and GLM

Exponential family

It is a probability distribution whose density has the form

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

pdf/pmf

$\eta \rightarrow$ prameter (eta)

We are modelling the output of our model with exponential family.

$b(y)$: Base measure

$T(y)$: Sufficient statistic { $T(y) = y$ for most cases }

$a(\eta)$: Log-partition function

η : Natural Parameter

Defining a new probability distribution, whose density is defined by some base probability in y times the exponent of a parameters times y.

$$p(y;\eta) \propto b(y)e^{-\eta \cdot y}$$

Normalize $\rightarrow$ $$p(y;\eta) = \frac{b(y)\cdot e^{-\eta\cdot y}}{\underbrace{\int b(y)e^{-\eta\cdot y}dy}_{= E[e^{-\eta y}]}}$$

This is also called as exponential tilting

$$p(y;\eta) = \frac{b(y)e^{-\eta^T y}}{\underbrace{E[e^{-\eta y}]}_{A(\eta)}} = \boxed{b(y)\cdot e^{\eta^T y - \underbrace{\log A(\eta)}_{a(\eta)}}}$$

$A(\eta)$: Normalizing constt.

- Any distribution that can be expressed in the form of this distribution, belongs to the exponential family.
- The only constraints are that the choices of b, T and a should be such that the RHS is always non-negative and integrates to 1.

Bernoulli distribution

$$\begin{aligned} P(y;\phi) &= \phi^y (1-\phi)^{1-y} \\ &= \exp\left[\log\left[\phi^y (1-\phi)^{1-y}\right]\right] \\ &= \exp\left[y \log\phi + (1-y)\log(1-\phi)\right] \\ &= \exp\left[\log\left(\frac{\phi}{1-\phi}\right)\cdot y + \log(1-\phi)\right] \end{aligned}$$

Compare this with general expression

$$P(y;\eta) = b(y)\exp\{\eta^T T(y) - a(\eta)\}$$

$\Rightarrow \eta = \log\left(\frac{\phi}{1-\phi}\right) \Rightarrow \boxed{\phi = \frac{1}{1+e^{-\eta}}}$ logistic

$\Rightarrow T(y) = y$

$\Rightarrow a(\eta) = -\log(1-\phi) \Rightarrow \boxed{\log(1+e^{\eta}) = a(\eta)}$

$\Rightarrow b(y) = 1$

Gaussian distribution

Univariate gaussian

$$p(y;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{\frac{-1}{2}\frac{(y-\mu)^2}{\sigma^2}\right\}$$

Assuming $\sigma^2 = 1$

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

$$= \left[\frac{1}{\sqrt{2\pi}} \exp\left(\frac{-1}{2}y^2\right)\right] \cdot \exp\left\{\mu y - \frac{1}{2}\mu^2\right\}$$

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

Matching pattern

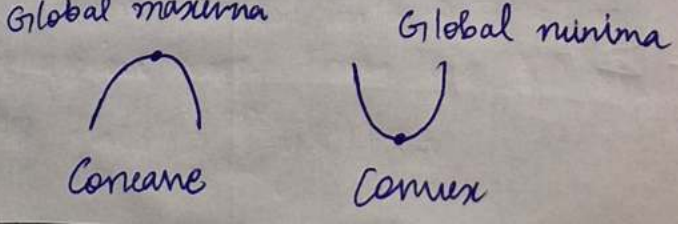
$b(y) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-y^2}{2}\right)$	$\eta = \mu$ $\mu = \eta$	$T(y) = y$ $a(\eta) = \mu^2/2 = \eta^2/2$

Properties of Exp families

① $\log p(y;\eta)$ is concave η. MLE is concave in η which means NLL is convex in η.

② $E[y;\eta] = \frac{\partial}{\partial \eta} a(\eta)$

③ $\text{Var}[y;\eta] = \frac{\partial^2}{\partial^2 \eta} a(\eta)$



Generalised linear Models (GLM)

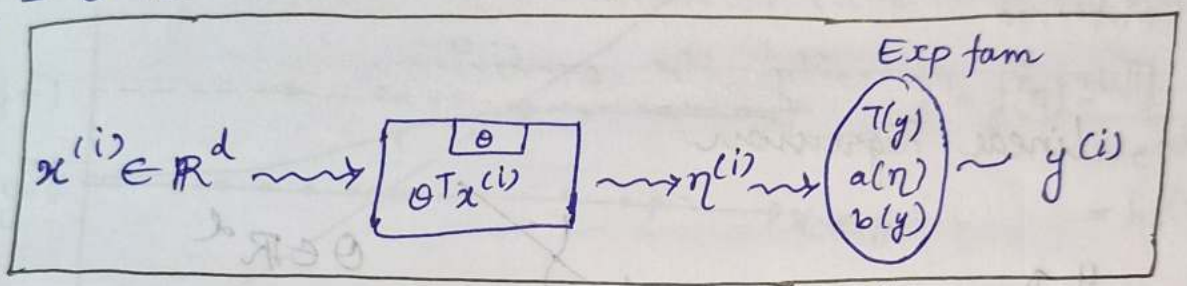
In GLM we use the exponential families to introduce a relation between a set of variables x to the variable y.

① $Y|X;\theta \sim$ Exp Family (η)

Given X and some θ, Y belongs to exp family with parameter η.

② $h_\theta(x) = E[y|x;\theta]$

③ $\eta = \theta^T x$



Flowchart of data generation

$\theta \rightarrow$ global (only one)

$\eta^{(i)} \rightarrow$ diff for each example
Each example will give its own η.

Ordinary least Square (OLS) : Linear Regression

Exp Family in Gaussian

$$h_\theta(x) = E[y|x;\theta]$$
$$= \mu$$
$$= \eta$$
$$= \theta^T x$$
$$h_\theta(x) = \theta^T x$$

Logistic Regression

Exp family is Bernoulli

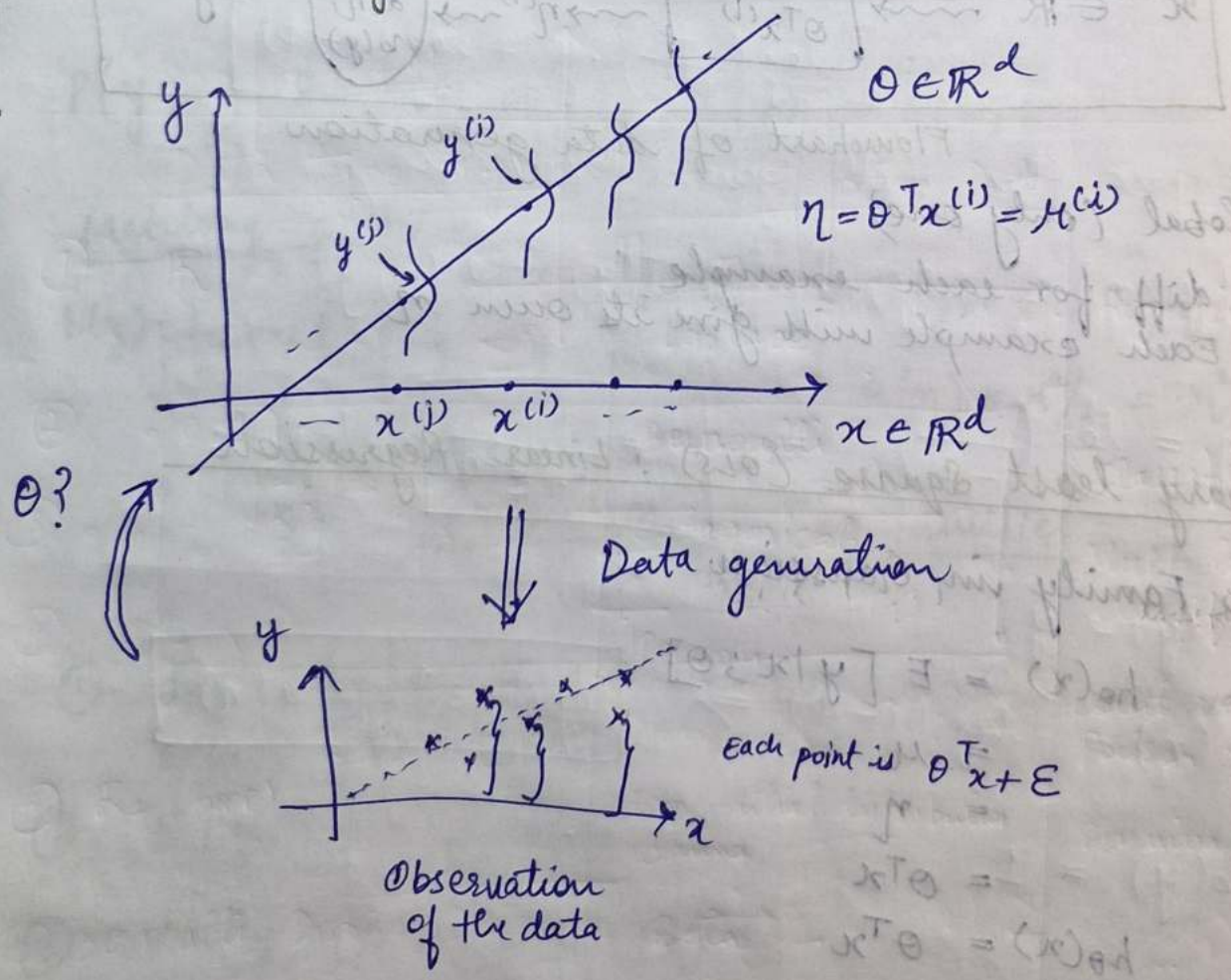
$h_\theta(x) = E[y|x;\theta]$

$= \phi$

$= \frac{1}{1+e^{-\eta}}$

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = g(\theta^T x)$; $g(z) = \frac{1}{1+e^{-z}}$

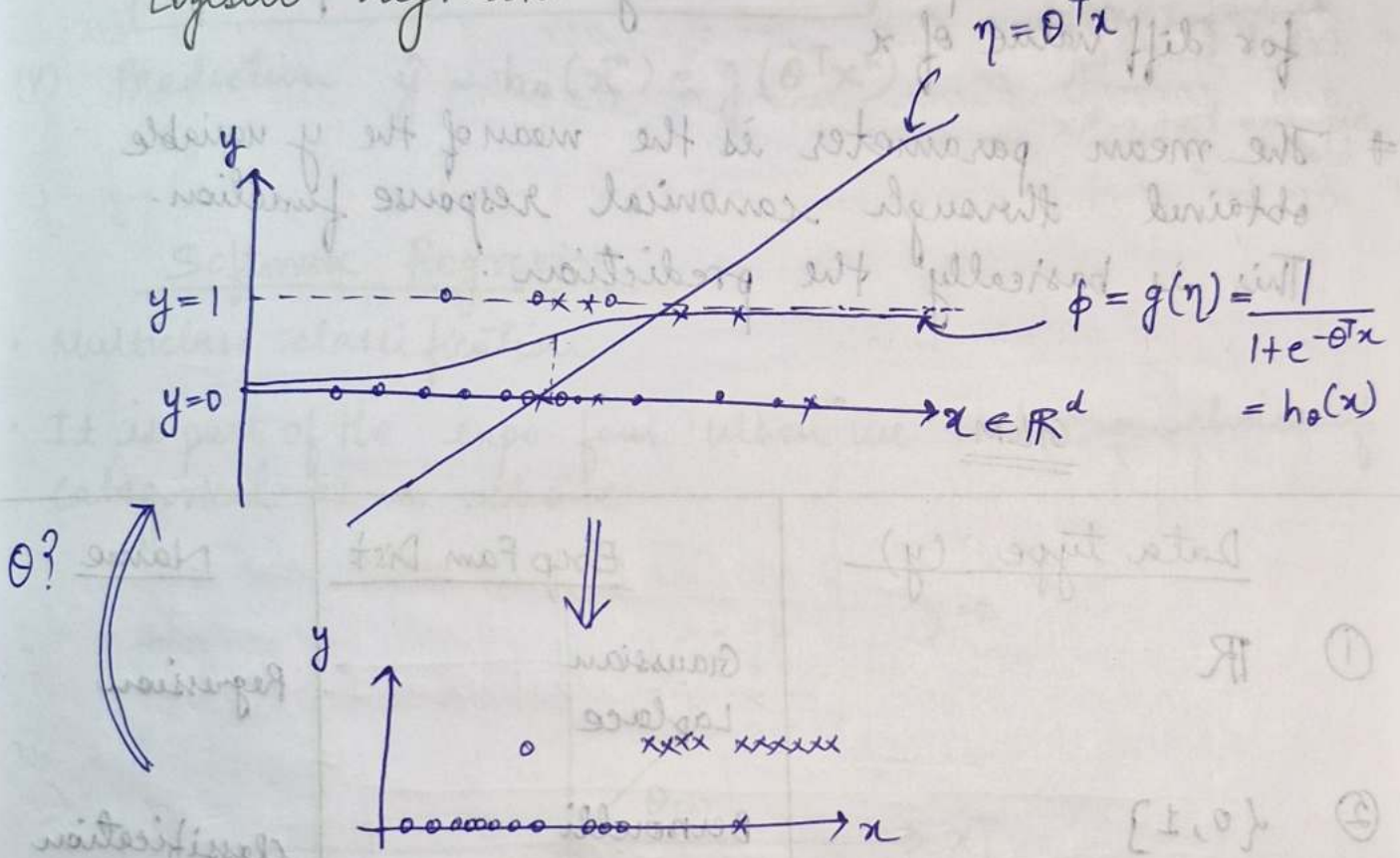
Linear Regression



For every input value $x^{(i)}$, we started with θ to get η and that $\eta = \mu$ of a Gaussian diss. From this GD we sample out y. That makes the $(x^{(i)}, y^{(i)})$ pair

Basically the visualization is like there is some θ which acts as the mean for each $y^{(i)}$ that we gonna sample from. The sampled $y^{(i)}$ will have some noise, we will start with these noisy observations and target to extract away the signal from that.

Logistic regression.



Model Parameters	Natural Parameter	Mean Parameter
$\theta \in \mathbb{R}^d$	$\eta^{(i)}$	μ: Gaussian $\mu = g(\eta) = \eta$
$x^{(i)} \in \mathbb{R}^d$	g	ϕ: Bernoulli $\phi = g(\eta) = \frac{1}{1+e^{-\eta}}$
$\theta^T x^{(i)}$	g	λ: Poisson
	g	

$$h_\theta(x) = E[y|x;\theta] = g(\theta^T x)$$

Y: Response variable
g = canonical response fun.
g^{-1} = canonical link function.

The model parameters: When we perform gradient descent, MLE on generalized linear model, these are the parameters that get trained. These are the parameters we are adjusting in each step of gradient descent or stochastic descent.

The Natural parameter is just the output of $\theta^T x^{(i)}$ for diff value of x

The mean parameter is the mean of the y variable obtained through canonical response function. This is basically the prediction.

GLM

Data type (y)	Exp Fam Dist.	Name
① $\mathbb{R}$	Gaussian Laplace	Regression
② $\{0,1\}$	Bernoulli	classification
③ $\{1,\dots,K\}$	Catagorical	Multiclass classification (Softmax)
④ $\mathbb{N}$	Poisson	Count reg. Poisson reg.
⑤ $\mathbb{R}_+$ (time)	Exponential distribution, Gamma	Survival analysis
⑥ Bernoulli Distri	Beta	.....

(i) Make choice of Distribution according to Data type

(ii) Express in Exponential family. $\rightarrow a(\eta), b(y), T(y)$

$\mu, \phi = g(\eta)$

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

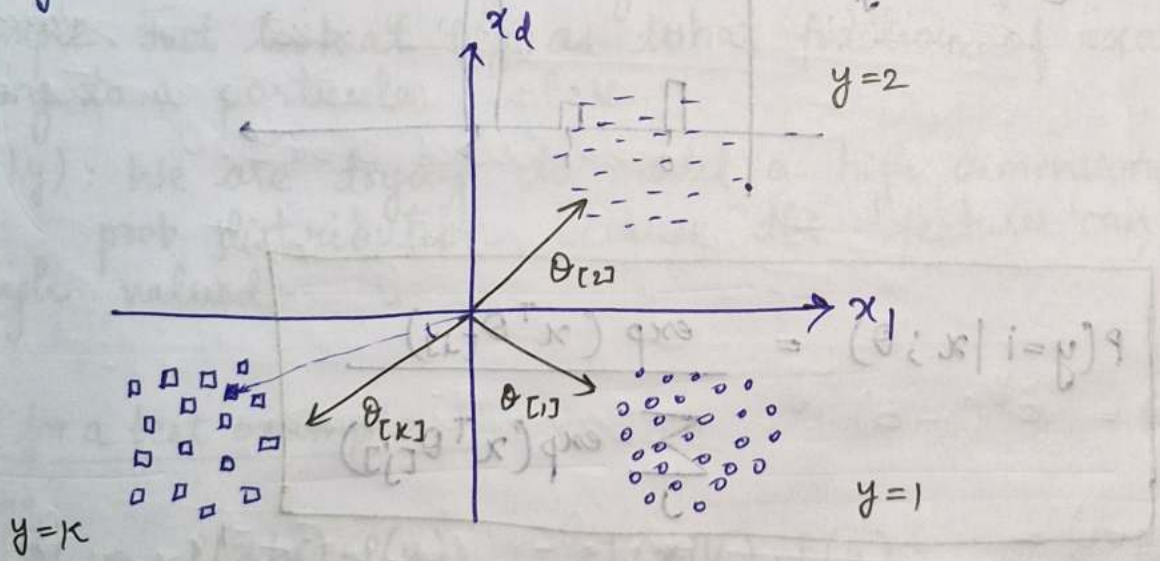
(iv) $\boxed{\theta = \theta + \alpha (y^{(i)} - h_\theta(x^{(i)})) \cdot x^{(i)}}$ ← No need to find Log, Now we can directly find it just use $h_\theta(x)$.

(v) Prediction $\hat{y} = h_\theta(x^*) = g(\theta^T x^*)$

$x^* \rightarrow$ test example (unseen)

Softmax Regression

- Multiclass classification
- It is part of the expo. fam. when we make ~~a choice of~~ Categorical as a choice.

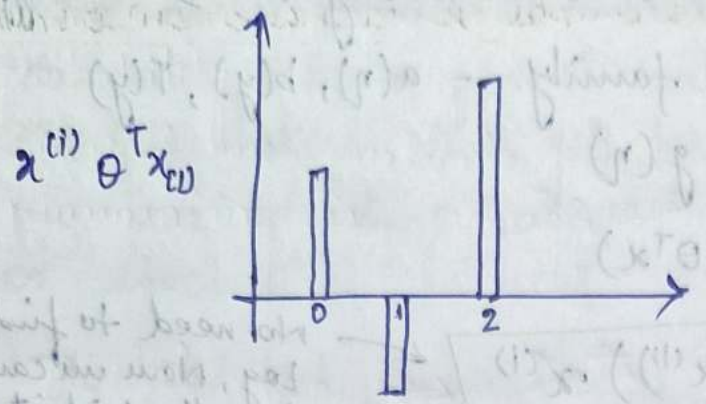


$\boxed{\theta^T x = 0}$

$$\theta_{[1]}^T x \Rightarrow R$$
$$\theta_{[2]}^T x \Rightarrow R$$
$$\vdots \quad \vdots$$
$$\theta_{[K]}^T x \quad R$$

$\Big\}$ arg max

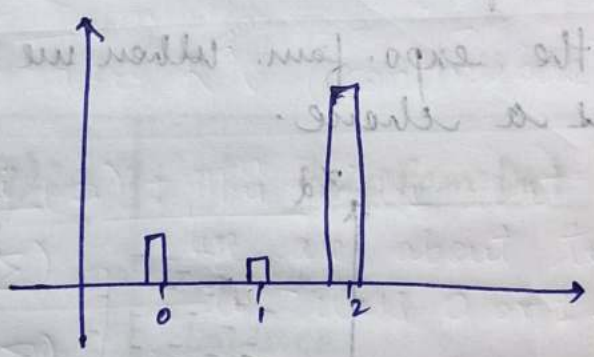
$$\theta = \begin{matrix} \uparrow \\ K \\ \downarrow \end{matrix} \begin{bmatrix} -\theta_{[1]}^T- \\ -\theta_{[2]}^T- \\ \vdots \\ -\theta_{[K]}^T- \end{bmatrix} \begin{bmatrix} x \end{bmatrix} \Big\} d$$



Softmax takes the set of these scalar valued dot prod and converts them into a probability distribution.

So for prob. distribution ⇒ All the values should be non neg.
↳ sum up to 1.

⇒ $\exp\left[x^{(i)}\theta_{[i]}\right]$



$$P(y=i \mid x;\theta) = \frac{\exp(x^T\theta_{[i]})}{\sum_j \exp(x^T\theta_{[j]})}$$

Softmax Function

$\vec{x}^T \cdot \text{Softmax}(\vec{x}) \approx \max(\vec{x})$

Lecture 7: GDA, Naive Bayes & Laplace Smoothing

Focus on X → Model $\underbrace{P(x,y)}$ = $P(x|y) \cdot \underbrace{P(y)}$

$P(y)$: class prior

So far we have studied Discriminative Algorithms where we directly model $P(y|x)$ but now our focus is on X.

$\underbrace{P(x,y)}$ = $\underbrace{P(x|y)}$ · $\underbrace{P(y)}$

Model — High dimensional — class prior

- We will be focusing on where y is discrete value
- GDA and Naive Bayes are best suited for classification problem.

\# y indicates the class of a given example. Just look at $P(y)$ as what fraction of example belong to a particular class.

* $P(x|y)$: We are trying to model a high dimensional prob. distribution, because the features can be high valued.

Let for a test example

$$\underbrace{P(y=1 \text{ or } 0|x)}_{\text{posterior Distribution}} = \frac{P(x|y)\cdot P(y)}{P(x)} = \frac{P(x|y)\cdot P(y)}{P(x|y=0)\cdot P(y=0) + P(x|y=1)\cdot P(y=1)}$$

Here y is assumed binary. $\{0,1\}$.

$$\hat{y} = \arg\max_y p(y|x) = \arg\max_y \frac{p(x|y)\cdot P(y)}{P(x)} = \arg\max_y p(x|y)P(y)$$

When we make prediction as a class rather than a probability. So $P(y=1 \text{ or } y=0 \mid x)$ gives a prob. for that class if we evaluate denominator too. Otherwise Numerator is enough to decide the class.

Model (Joint $p(x,y)$) $\Leftrightarrow$ Data Generating Process

Hierarchy of steps

$\Downarrow$

Factorize our joint probability

Gaussian Discriminant Analysis (GDA)

$y = \{0, 1\}$ and $x \in \mathbb{R}^d$ (continuous)

Steps

$y \sim \text{Bernoulli}(\phi)$: This decides that the example we are about to generate belong to class 0 or 1.

$x \mid y=0 \sim \mathcal{N}(\mu_0, \Sigma)$

$x \mid y=1 \sim \mathcal{N}(\mu_1, \Sigma)$ $\rightarrow$ Generating input

$P(y) = \phi^y (1-\phi)^{1-y}$: Bernoulli distri.

$$P(x \mid y=0) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2} (x-\mu_0)^T \Sigma^{-1} (x-\mu_0) \right\}$$

$$P(x \mid y=1) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left\{ -\frac{1}{2} (x-\mu_1)^T \Sigma^{-1} (x-\mu_1) \right\}$$

$y \in \{0, 1\}$

$x \in \mathbb{R}^d$

In discriminative algorithms all we do is to discriminate if an example belong to this class or the other. Which can actually limit ourselves to look for certain queues in the input and ignore the rest of the example.

Whereas, in GDA we generate an entire example. We come up with full description of what makes up $y=1$ or $y=0$. In simple terms Generative modelling is harder than Discriminative modelling.

$$\underbrace{P(x,y)}_{\text{Generative model}} = \underbrace{P(y|x)}_{\text{Discriminative mode}} \cdot \underbrace{P(x)}$$

So in order to learn a gen. model, we have to learn a discriminative model and also something else.

Max Likelihood to learn parameters..

After generating the data, we observe the data. We can use the data to fit our model and learn these parameters $(\phi, \mu_0, \mu_1, \Sigma)$ using MLE.

A likehood is a function over parameters

Log likelihood

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{n} P(x^{(i)}, y^{(i)})$$

$$= \log \prod_{i=1}^{n} P(x^{(i)}|y^{(i)}) P(y^{(i)})$$

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \log L(\phi, \mu_0, \mu_1, \Sigma) = \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)})$$

$$\ell(\phi, \mu_0, \mu_1, \Sigma) = \sum_{i=1}^{m} \log p(x^{(i)}, y^{(i)})$$

$$= \sum_{i=1}^{m} \left(\log p(x^{(i)} | y^{(i)}) + \log p(y^{(i)}) \right)$$

* MLE: $\nabla \ell(\phi, \mu_0, \mu_1, \Sigma) = 0$

① Gradient with respect to ϕ:

$$p(y^{(i)}) = \phi^{y^{(i)}} (1-\phi)^{1-y^{(i)}}$$

$$\sum_{i=1}^{m} \log p(y^{(i)}) = \sum_{i=1}^{m} \left[y^{(i)} \log \phi + (1-y^{(i)}) \log (1-\phi) \right]$$

$$\frac{\partial \ell}{\partial \phi} = \sum_{i=1}^{m} \left[\frac{y^{(i)}}{\phi} - \frac{1-y^{(i)}}{1-\phi} \right] = 0$$

$$\boxed{\hat{\phi} = \frac{1}{m} \sum_{i=1}^{m} y^{(i)}}$$

This is MLE for ϕ, which is simply the fraction of data points where $y^{(i)} = 1$.

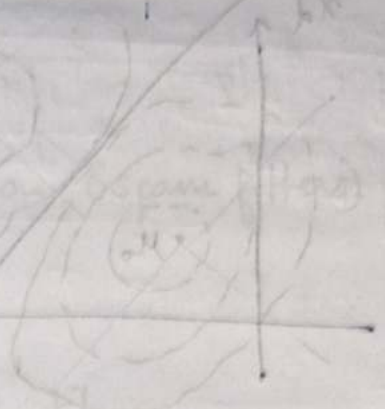
② Gradient with respect to μ_0:

$$\sum_{i : y^{(i)}=0} \log p(x^{(i)} | y^{(i)} = 0)$$

$$= \frac{-1}{2} \left((x^{(i)} - \mu_0) \right)^T \Sigma^{-1} (x^{(i)} - \mu_0) + \log |\Sigma| + d \log (2\pi)$$

$$\frac{\partial l}{\partial \mu_0} = \sum_{i:y^{(i)}=0} \Sigma^{-1}(x^{(i)} - \mu_0)$$

$$\hat{\mu}_0 = \frac{\sum_{i:y^{(i)}=0} x^{(i)}}{\sum_{i:y^{(i)}=0} 1}$$



③ Gradient with respect to μ_1

$$\sum_{i:y^{(i)}=1} \log p(x^{(i)} | y^{(i)}=1)$$

$$\frac{\partial l}{\partial \mu_1} = \sum_{i:y^{(i)}=1} \Sigma^{-1}(x^{(i)} - \mu_1)$$

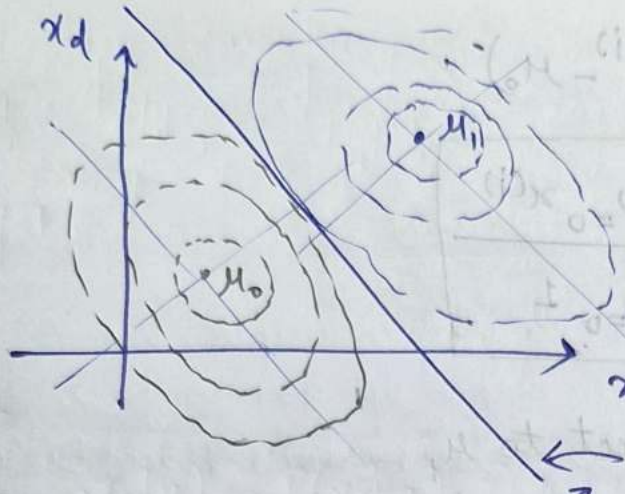
$$\hat{\mu} = \frac{\sum_{i:y^{(i)}=1} x^{(i)}}{\sum_{i:y^{(i)}=1} 1}$$

④ Gradient w.r.t Σ

$$\sum_i \log p(x^{(i)} | y^{(i)})$$

$$\frac{\partial l}{\partial \Sigma} = -\frac{1}{2} \sum_{i=1}^{m} \left[\Sigma^{-1} - \Sigma^{-1}(x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T \Sigma^{-1} \right]$$

$$\hat{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$



- Two means for two diff class
- Share same cov. matrix.

← Hyperplane

$p(x|y=0) = p(x|y=1)$ ↗ ~~$p(y=1|x) = p(y=0|x) = 0.5$~~

For any GDA models that shares the same Σ, the posterior distribution of $(y|x)$ can be represented as a Logistic regression model, where the θ only depends on the parameters of the model $(\mu_0, \mu_1, \phi, \Sigma)$

GDA $\Rightarrow$ Logistic Regression : $p(y=1|x) = \frac{1}{1+e^{-\theta^T x}}$

Naive Bayes

x - Discrete valued

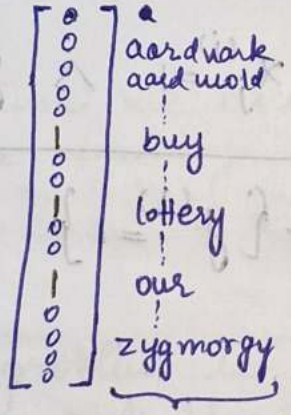
Used in for eg text classification (spam filters)

Conditional Independence

$P(x_j | x_k) = p(x_j)$ [independent]

y: $P(x_j | x_k, y) = P(x_j | y)$ [Conditional independance]

Bernoulli Event Model

"Buy our lottery" = 

Vocabulary - d

$x \in \{0,1\}^d$

$x_j \in \{0,1\}$

Model

$P(y=1) = \text{Bernoulli}(\phi_y)$ — 1 parameter

$P(x_j=1|y=0) = \text{Bernoulli}(\phi_{j|y=0})$ — d parameters

$P(x_j=1|y=1) = \text{Bernoulli}(\phi_{j|y=1})$ — d parameters

The intuition that eg $P(x_j=1|y=1)$ whats the probability that this word corresponding to index j will show up in an email that is spam.

$$l(\underbrace{\phi_y}_{1}, \underbrace{\phi_{j|y=0}}_{d}, \underbrace{\phi_{j|y=1}}_{d}) = \log \prod_{i=1}^{n} P(x^{(i)}, y^{(i)}; \phi)$$

$$= \log \prod_{i=1}^{n} P(y^{(i)}; \phi_y) \left(\prod_{j=1}^{d} P(x_j^{(i)} | y^{(i)}; \phi \right)$$

Property

$$P(x_1, x_2, \cdots, x_n | y) = P(x_1|y) \cdot P(x_2|x_1, y) \cdot P(x_3|x_1, x_2, y) \cdots$$

$$= P(x_1|y) \cdot P(x_2|y) \cdot P(x_3|y) \cdots$$

MLE estimates

* $$\phi_{j|y=1} = \frac{\sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}$$

* $$\phi_{j|y=0} = \frac{\sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$

* $$\phi_y = \frac{\sum_{i=1}^{n} 1\{y^{(i)} = 1\}}{n}$$

Now if a test word comes
• Calculate the probality and compare.

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{P(x|y=0)P(y=0) + P(x|y=1)P(y=1)}$$

$$P(y=1|x) = \frac{\prod_{j=1}^{d} P(x_j^{(i)}|y=1)\,P(y=1)}{\text{denom.}}$$

$$P(y=0|x) = \frac{\prod_{j=1}^{d} P(x_j^{(i)}|y=0)\,P(y=0)}{\text{denom}}$$

} compare them to decide spam or not

Now what if at test time we encounter a word that was never seen in the training set.

$$P(y|x) = \frac{0}{0+0}$$

With this method the problem is that, in this case we cannot tell what our prediction should be. To overcome this a technique is used called Laplace smoothing.

Eg Suppose a coin tose, and on all 10 trials H turns up. Therefore $X \in \{0, 1\}$ but to encounter such

$$\phi = p(H = {}^{10}/_{10} = 1$$

problem in Laplace smoothing we initialize count of every example by 1. (Basically assuming that we have seen that example at least once).

After Laplace smoothing

* $$\phi_{j|y=1} = \frac{1 + \sum_{j=1}^{n} 1(x_j^{(i)} = 1 \wedge y^{(i)} = 1)}{2 + \sum_{i=1}^{n} 1(y^{(i)} = 1)}$$

* $$\phi_{j|y=0} = \frac{1 + \sum_{j=1}^{n} 1(x_j^{(i)} = 1 \wedge y^{(i)} = 0)}{2 + \sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$

Multinomial Event Model

$y \sim$ Bernoulli (ϕ) (1 dimension)

$X|y=0 \sim$ ~~Multino~~ Catagorical (ϕ) $(|V|-1$ dimension)

MLE

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 1\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 1\} d_i}$$

* K is the word in the vocabulary is given by → Summing over each message, within each message we are summing over every word and counting only those words in those messages which happen to be K and belong to the class we are interested in. Divided by total no. of words across all messages in that class.

Similarly $\phi_{k|y=0}$. ⇒

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{d_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = 0\}}{\sum_{i=1}^{n} 1\{y^{(i)} = 0\} d_i}$$

Laplace smoothing

* $\phi_{k|y=1} = \frac{1 + N_{k|y=1}}{|V| + D_{k|y=1}}$

* $\phi_{k|y=0} = \frac{1 + N_{k|y=0}}{|V| + N_{k|y=0}}$

- N - Numerator from previous proof.
- D - Denominator from the previous proof.

Bernoulli method

	a	aardvark	...	buy	...	exam	...
"buy our lottery"	0	0	0	1	0	0	1 ...
"buy this watch, this watch only"	0	0	0	1	0	0	0 1 0 1 ...
	$\phi_{j\|y=1}$			2/2			
"when is your exam"	0	0	0	0	0	1	0 1 0
"When is your homework due"	0	0	0	0	1	0	0 1 0 0
				$\phi_{j\|y=0}$	1/2		

* We put 1 only for the no. of message in which the words appear, doesn't matter how many times the word appears in the same message the count will be 1 only.

* $\phi \rightarrow \frac{\text{Messages in which the word appear}}{\text{Total no. of messages.}}$

Multinomial model

	a aaaaaaa — — buy — — exam — — —
"buy our lottery"	0 0 1 — — 1 0
"buy this match, this match only"	0 0 1 — — 1 2
	4 2 3 2 2
	$\Sigma = 4+2+3+2+2$
"When is your exam"	
"When is your homework due"	

Here we consider the number of times a word appear across all examples in your training set, whereas in Bernoulli it was the number of messages in which the word appear.

In Bernoulli each word has its own bernoulli variable, so we normalized it locally. But in Multinomial model we want a distribution over words, so we normalize the entire thing.

Lecture 8 - Kernel Methods & SVM

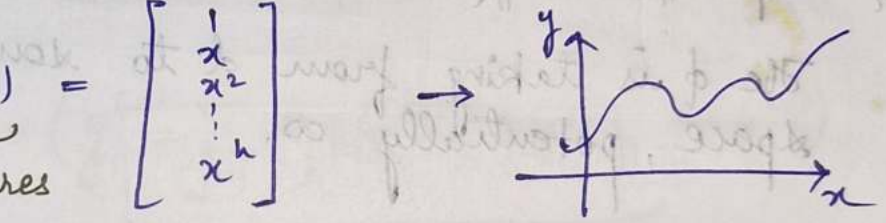
Kernel Methods

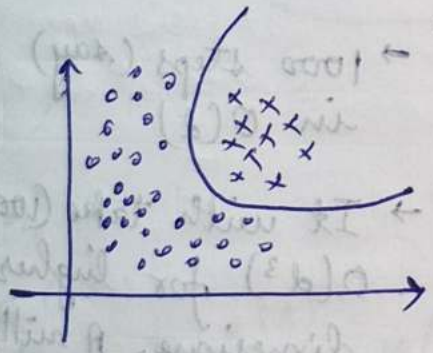
So far we learned the linear and logistic regression are linear in nature (linear models). They provide linear hypothesis.

But we can construct non-linear hypothesis using linear regression and similarly non-linear classifier using logistic regression by including higher order features.

$x \rightarrow \phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^n \end{bmatrix}$

x: attributes; $\phi(x)$: Features





* With kernel methods we can map an example to potentially infinite dimensional feature vector and perform the learning algorithm in the infinite dimensional space.

Linear regression with gradient descent.

update rule is

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T x^{(i)}\right) \cdot x^{(i)} \longrightarrow [\theta \in \mathbb{R}^d]$$

With feature map.

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \sum_{i=1}^{n} \left(y^{(i)} - \theta^T \phi(x^{(i)})\right) \cdot \phi(x^{(i)})$$

$\longrightarrow [\theta \in \mathbb{R}^p]$

$\Rightarrow \quad \phi : \mathbb{R}^d \to \mathbb{R}^p$

The ϕ is taking from d to some high dimensional space, potentially ∞.

Eg

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_1^2 \\ \vdots \\ x_1^2 x_2 \end{bmatrix}$$

monomial terms of order ≤ 3

$p \approx O(d^3)$

- → 1000 steps (say) in $O(d)$
- → It with take $(1000)^3$ $O(d^3)$ for higher dimesions. A million times slower.

The claim is that the $\theta^{(t)}$ that we encounter at any stage of our gradient descent can always be represented as a linear combination of the features.

$$\theta^{(t)} = \sum_{i=1}^{n} \beta_i^{(t)} \phi(x^{(i)})$$

$$\theta^{(1)} = \sum_{i=1}^{n} \underbrace{\alpha y^{(i)}}_{\beta_i^{(1)}} \cdot \phi(x^{(i)})$$

Inductive approach to prove.

$$\theta^{(t+1)} = \theta^{(t)} + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^{(t)^T} \phi(x^{(i)}))\phi(x^{(i)})$$

$$= \sum_{i=1}^{n} \beta_i^{(t)} \phi(x^{(i)}) +$$

$$\alpha \sum_{i=1}^{n} \left(y^{(i)} - \underbrace{\left(\sum_{j=1}^{n} \beta_i^{(t)} \phi(x^{(i)})\right)^T}\phi(x^{(i)})\right) \cdot \phi(x^{(i)})$$

$$\theta^{(t+1)} = \sum_{i=1}^{n} \underbrace{\left[\beta_i^{(t)} + \alpha\left(y^{(i)} - \sum_{j=1}^{n} \beta_j \phi(x^{(j)})^T \phi(x^{(i)})\right]}_{\beta_i^{(t+1)}} \phi(x^{(i)})$$

$$\Downarrow$$

For $i = 1, \ldots, n$

$$\beta_i^{(t+1)} = \beta_i^{(t)} + \alpha\left(y^{(i)} - \sum_{j=1}^{n} \beta_j^{(t)} \phi(x^{(j)})^T \phi(x^{(i)})\right)$$

Kernal

$$\triangleq \boxed{K: X \cdot X \longrightarrow \mathbb{R}}$$

$x \in \mathbb{R}^d \qquad X = \mathbb{R}^d$

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

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

* A Kernel is a function which takes two element of that space and returns a Real valued scalar. It also

satisfies the property that the evaluated value can be expressed as the inner product between two feature maps (feature map of input 1 and dot product of feature map of input 2).

Eg.

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1^3 \\ \vdots \end{bmatrix} \qquad x \in \mathbb{R}^d$$

$\therefore P \approx O(d^3)$

$$K(x,z) = 1 + \underbrace{\langle x,z\rangle}_{O(d)} + \underbrace{\langle x,z\rangle^2}_{O(d)} + \underbrace{\langle x,z\rangle^3}_{O(d)} \Big\} O(d)$$

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

$\phi(d^3)$ - otherwise but this approach Reduce it to $O(d)$

$$\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_1^2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ z_1 \\ z_2 \\ \vdots \\ z_1^2 \end{bmatrix} = \begin{matrix} 1 \\ +x_1 z_1 \\ +x_2 z_2 \\ \vdots \\ x_d z_d \end{matrix} \Big\} O(d^3)$$

* Basically an order $O(d)^n$ operation is reduced to $O(d)$.

\# For a feature ϕ can have a corresponding kernel K where the kernel function has a more compact representation which is equivalent to performing a dot product in the high dimensional space.

$$\phi : \mathcal{X} \longrightarrow \mathbb{R}^q$$

Linear Regression (Kernalized)

① Precompute:

$$K_{ij} = K(x^{(i)}, x^{(j)})$$

$$K_{ij} = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$$

* We use same K notation for both matrix & function here K_{ij} is the square symm. matrix and $K(x^{(i)}, x^{(j)})$ is function.

② Loop: $\forall\ i \in \{1, 2, \ldots, n\}$, $\beta^{(t)} \in \mathbb{R}^n$ (vector)

$$\beta_i^{(t+1)} = \beta_i^{(t)} + \alpha\left(y^{(i)} - \sum_{j=1}^{n} \beta_j^{(t)} K_{ij}\right)$$

$$\beta^{(t+1)} = \beta^{(t)} + \alpha\left(\vec{y} - K\beta^{(t)}\right)$$

Prediction

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

$$= \sum_{i=1}^{n} \beta_i \phi(x^{(i)})^T \phi(x)$$

$$= \sum_{i=1}^{n} \beta_i K(x^{(i)}, x) \qquad \{x - \text{test example}\}$$

Observations

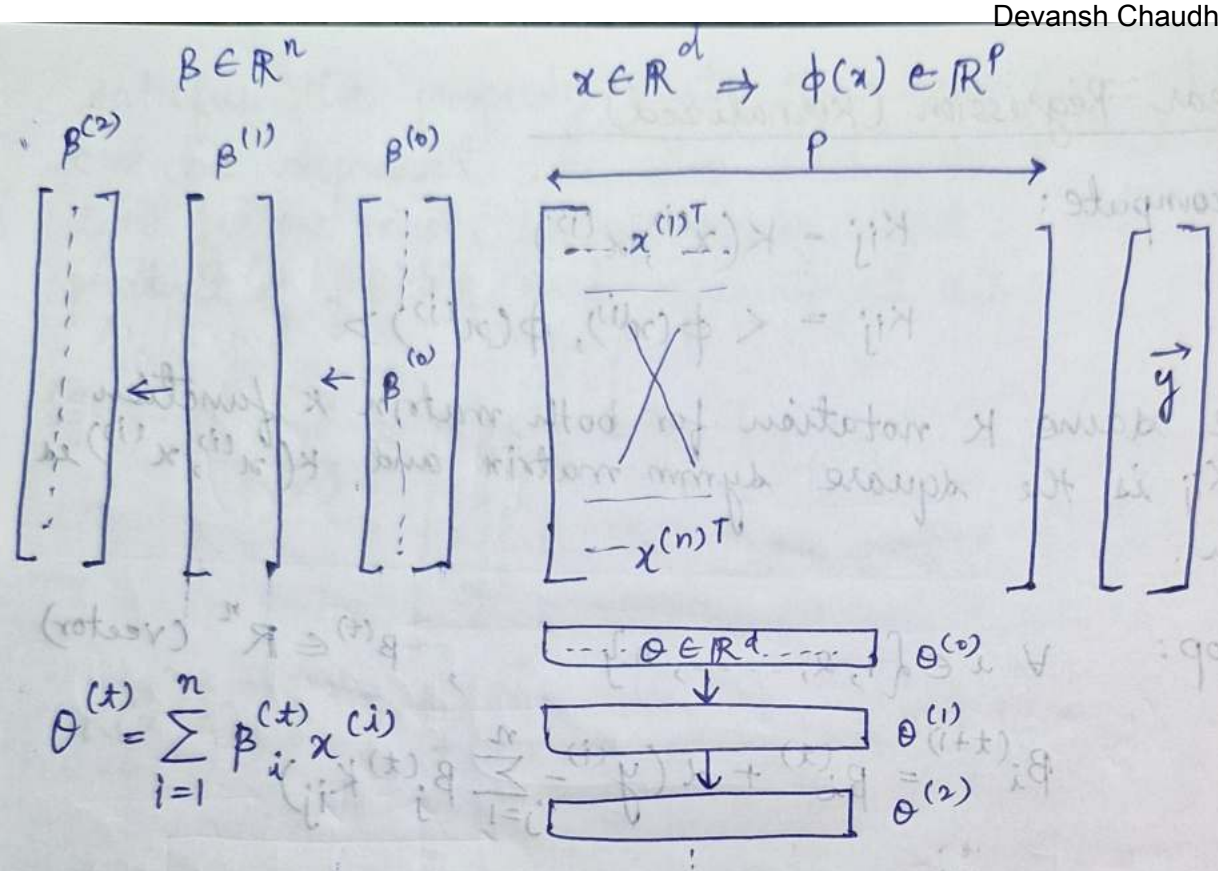
① Train = $\beta = \beta + \alpha(\vec{y} - K\beta)$

Test = $\hat{y} = \sum_{i=1}^{n} \beta_i K(x^{(i)}, x)$

- $\phi(x)$ does not appear
- If we had $\phi(x)$ we have to compute it and it could be ∞ dimensional

② For prediction we need training examples to be stored in memory.

③ As can be inferred from ①, we need to remember all our training examples for testing



θ at any descent can be represented as a linear combination of vectors.

θ has one component per feature.

β has one component per example

Every β vector has a corresponding θ vector

This helps us to has infinite feature wide, and we achieve scaling.

Kernalization is a general technique that can be applied to a lot of places - Gen or discrimi, super or unsup classification or regression etc.

Kernel examples

eg A $K(x,z) = \langle x,z \rangle^2 \quad x \in \mathbb{R}^d$

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ \vdots \\ x_d x_d \end{bmatrix}$$

eg B $K(x,z) = (x^T z + c)^2$

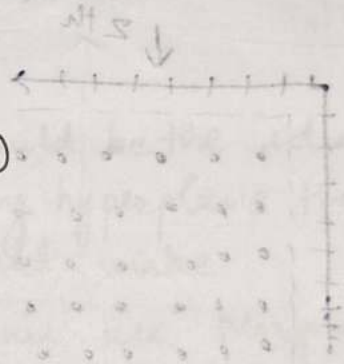
$$\phi(x) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ \vdots \\ \sqrt{2c}\, x_1 \\ \sqrt{2c}\, x_2 \end{bmatrix} \qquad K(x,z) = \phi(x)^T \phi(z)$$

$K(x,z)$ ↑ (for similar x, z)

$K(x,z)$ ↓ (for non similar x, z)

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

$K(x,z) = e^{+\left(\frac{-\|x-z\|^2}{2\sigma^2}\right)}$



Necessary condition for K to be a kernel

- K should be symmetric

  $K(x,z) = K(z,x)$

- $\{x^{(1)}, \ldots\ldots, x^{(m)}\}$

  $K_{ij} = K(x^{(i)}, x^{(j)})$

  K is symmetric & PSD

- $z^T K z = \sum_i \sum_j z_i K_{ij} z_j$

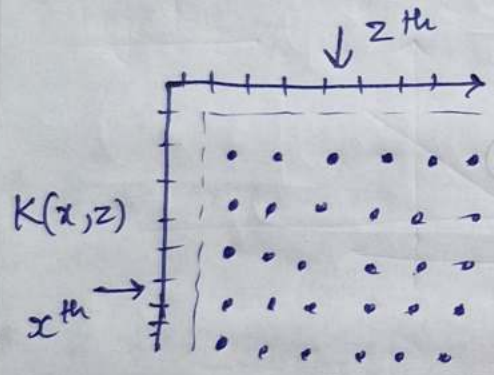
  $= \ldots\ldots = \sum_k \left(\sum z_i \phi_k(x^{(i)})\right)^2 \geq 0$

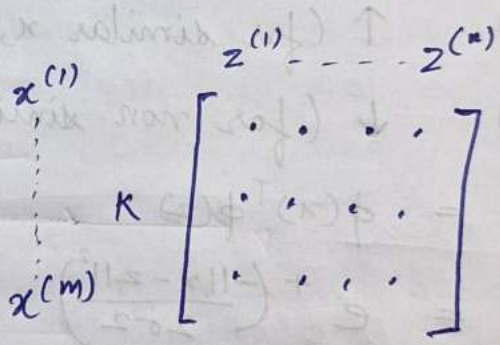
Mercer's theorem

Let $K: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be given.
For K to be a kernel, it is necessary and sufficient for any $\{x^{(1)}, \ldots, x^{(m)}\}$ the corresponding Kernel matrix

$$K_{ij} = K(x^{(i)}, x^{(i)}) \text{ is PSD and symmetric.}$$

Vector $\longleftrightarrow$ functions
Matrix $\longleftrightarrow$ Operators / function (x, y)





To prove K is kernel

1. Construct ϕ such that $K(\,,\,) = \phi^T \phi$
2. Mercer's theorem
   $\{x^{(1)} \ldots x^{(m)}\}$
   $K_{ij} = K(x^{(i)}, x^{(j)})$ is PSD
3. $\iint f(x)\, K(x, x')\, f(x')\, dx\, dx' \geq 0$

Support Vector Machines (SVM)

- Discriminative
- Classification

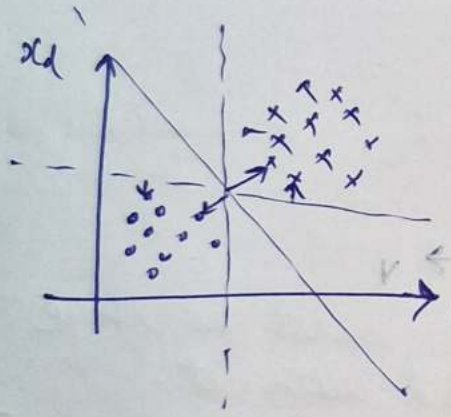
Notations

$y^{(i)} \in \{+1, -1\}$

Parameters w, b

$w \in \mathbb{R}^d$, $b \in \mathbb{R}$

$x \in \mathbb{R}^d$ $\Rightarrow w^T x + b$.



→ What should be the ideal separating hyperplane that we should make

→ For this we use Margin.

Margin = $y^{(i)}(w^T x^{(i)} + b) > 0$

$w^T x + b > 0$ for $y = +1$

$w^T x + b < 0$ for $y = -1$

Desire: Margin to be large

The idea of SVM is to calculate the margin w.r.t all the examples and maximize the smallest margin.

SVM is an algorithm that tries to maximize the geometric margin (i.e the actual geometric distance b/w these separating hyperplane).

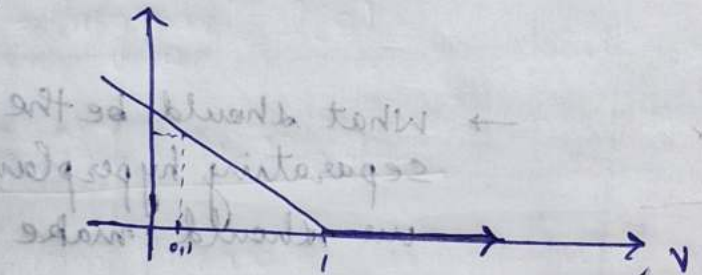
Problem with $y^{(i)}(w^T x + b) > 0$ is that it can be scaled just by increasing coeff. i.e $y^{(i)}(2w^T x + 2b)$

SVM addresses this problem

$$\min_{w,b} \left(\sum_{i=1}^{n} \max\left[0, \underbrace{1 - y^{(i)}(w^T x^{(i)} + b)}_{\text{Margin} = v^{(i)}}\right] \right) + \underbrace{\frac{1}{C}\|w\|^2}_{\text{safeguard for scaling}}$$

Hinge loss / SVM loss.

Hinge loss



$$\Rightarrow \min_{\xi, w, b} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i$$

Such that,

$$y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i; \ \forall i \in \{1, \ldots, n\}$$

$$\xi_i \geq 0, \quad i = 1, \ldots, n$$

Primal Convex Problem

$$\Rightarrow \max_{\alpha} = \sum_{i=1} \alpha_i - \frac{1}{2}\sum_{i,j} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle$$

Such that $0 \leq \alpha_i \leq C$

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

Dual Convex problem

A property of SVM is that once we solve the set of α, as a result the set of α will be sparse. which means only a small number of α_i will be non zero. but most of the $\alpha_i = 0$.

Those set of example for which $\alpha_i \neq 0$ are called the support vectors. Intuitively those examples will be nearest to the separating hyperplane.

Lecture 9 - Bayesian Methods - Parametric & Non

Frequentist Methods

In the methods that we have studied previously we take θ as some constant

Unknown constant θ

$$l(\theta) = \log P(\text{data}; \theta)$$

$-l(\theta)$ = loss function

$$\hat{\theta} = \arg\min_{\theta} -l(\theta)$$

$$\hat{\theta} = \arg\max_{\theta} l(\theta)$$

Bayesian Method

θ - Random Variable unobserved

↳ which implies that there must be some prior distribution associated with it.

$\theta \sim$ Prior Distribution

Y, X (Data) $\sim P(X|\theta)$

$$\underbrace{P(\theta|X)}_{\text{Posterior Dist.}} = \frac{P(X|\theta) \cdot P(\theta)}{P(x)} \quad \therefore \text{Bayesian rule}$$

$$= \frac{P(x|\theta)\, P(\theta)}{\int P(x|\theta) \cdot P(\theta)\, d\theta} = \frac{\sim}{\int P(x, y)\, d\theta}$$

In the Bayesian method there is no loss function, we don't take MLE, don't calculate gradients. All we

do is apply Bayes rule and get update our beliefs about θ given the Data.

In Supervise ML

$\theta \sim$ Prior , $\theta \perp x$ (Independent)

$Y \sim P(y|X, \theta)$

Posterior distri: $P(\theta|X,Y) = \dfrac{P(y|X,\theta)\cdot P(\theta)}{P(y|x)}$

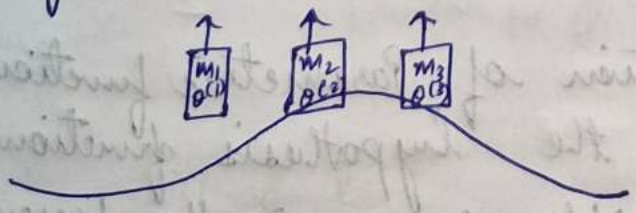
Posterior predictive distribution

$$P(Y_*|X, Y, X_*) = \int P(Y_*|X_*, \theta)\cdot P(\theta|X,Y)\,d\theta$$

↓ Interpretation

With the Posterior distribution we are coming up with a probability estimate of how likely a given θ is the correct θ.

$\int P(Y_*|X_*,\theta)\cdot P(\theta|X,Y)\,d\theta \Rightarrow$ Making a prediction for Y for some given value of θ. Then we are calculating the weighted avg. of all the predictions where the weights are decided by Posterior distribution.

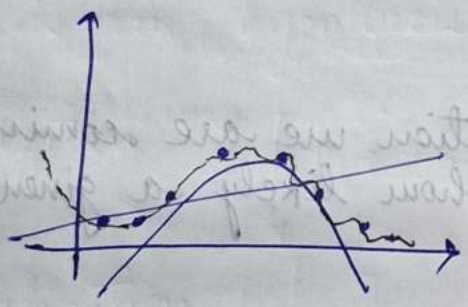


Parametric

This approach we discussed will be called parametric because $p(y|x,\theta)$ has some functional form for which θ is a parameter

$$\Rightarrow \quad Y|X;\theta = \frac{1}{1+e^{-\theta^T x}}$$

* The degree of freedom here we have is to just change θ and that in a way limits the set of all possible models that we can consider.



- May be linear, quadratic or some higher degree polinomial

* By limiting ourselves to certain kinds of parametric function, we are essentially saying that no matter how much data is given to us we are not flexible to fit all the variance.

* That is the limitation of Parametric function that the functional form of the hypothesis function limits us from being flexible even in the presence of lot of data which suggests that there is some other pattern.

Bayesian Linear Regression

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

$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$, $\theta \in \mathbb{R}^d$

$\varepsilon^{(i)} \sim N(0, \sigma^2)$, $x^{(i)} \in \mathbb{R}^d$

$\theta \sim N(\vec{\theta}, \tau I)$

Frequentist approach

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

Bayes

Posterior: $\theta|S \sim N\left(\frac{1}{\sigma^2} A^{-1} X^T \vec{y}, A^{-1}\right)$ where $A = \frac{1}{\sigma^2} X^T X + \frac{1}{\tau^2} I$

$$\theta|S \sim N\left(\underbrace{\left(X^T X + \frac{\sigma^2}{\tau^2} I\right)^{-1} X^T Y}_{\text{similar to } (X^T X)^{-1} X^T Y}, \left(\frac{1}{\sigma^2} X^T X + \frac{1}{\tau^2} I\right)^{-1}\right)$$

Posterior Predictive: Property of Gaussians $\theta \sim N(\mu, \Sigma)$

$y_* = x^T \theta \sim N(\mu^T x, x^T \Sigma x)$

$$Y_*|X_*, S \sim N\left(\frac{1}{\sigma^2} x_*^T A^{-1} X^T \vec{y}, x_*^T A^{-1} x_* + \sigma^2\right)$$

* In general, Bayesian methods tends to be heavy utilizers of probability theory because all we do is taking Probability distributions and conditioning them to apply Bayes rule.

Gaussian Processes

Vector : functions :: MV Gaussians : Gaussian Processes
↓ ↓
Vectors. functions

Properties

The function over which we will define our gaussian process is our hypothesis function $h(x)=y$.

① Normalization $\int_x P(x; \mu, \Sigma)\, dx = 1$

② Marginalization

$$x = \begin{bmatrix} x_A \\ x_B \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}, \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix} \right)$$

μ Σ

$$P(x_A) = \int_{x_B} P(x; \mu, \Sigma)\, dx_B$$

$$= N(\mu_A, \Sigma_{AA})$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \cdots & \Sigma_{1d} \\ \vdots & & \\ \Sigma_{d1} & \cdots & \Sigma_{dd} \end{bmatrix} \right)$$

③ Conditioning

$$x_A | x_B \sim \mathcal{N}\left(\mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(x_B - \mu_B),\ \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA}\right)$$

* Covariance $= \rho \cdot \text{Std}(A) \cdot \text{Std}(B)$

$$\begin{bmatrix} a \\ b \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}, \begin{bmatrix} \sigma_a^2 & \rho\sigma_a\sigma_b \\ \rho\sigma_a\sigma_b & \sigma_b^2 \end{bmatrix}\right)$$

$$a|b \sim \mathcal{N}\left(\mu_a + \frac{\rho\sigma_a\sigma_b}{\sigma_b^2}(b - \mu_b),\ \sigma_a^2 - \rho\sigma_a\sigma_b\frac{\rho\sigma_a\sigma_b}{\sigma_b^2}\right)$$

$$\sim \mathcal{N}\left(\mu_a + \sigma_a \cdot \rho \cdot \left(\frac{b - \mu_b}{\sigma_b}\right),\ \sigma_a^2(1-\rho)\right)$$

④ Summation

$$x \sim \mathcal{N}(\mu_1, \Sigma_1)$$

$$y \sim \mathcal{N}(\mu_2, \Sigma_2)$$

$$\boxed{x + y \sim \mathcal{N}(\mu_1 + \mu_2,\ \Sigma_1 + \Sigma_2)}$$

MV Gaussian

Gaussian Process

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_d \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_d \end{bmatrix}, \begin{bmatrix} & \\ & \end{bmatrix} \right) \Leftrightarrow \begin{bmatrix} | \\ f \\ | \end{bmatrix} \sim GP\left(\begin{bmatrix} m \end{bmatrix}, \begin{bmatrix} k \end{bmatrix} \right)$$

Vector — Mean Vector — Covariance Matrix — functions — Mean func. — Kernel

$$\Updownarrow$$

$$\begin{bmatrix} f(x^{(1)}) \\ f(x^{(2)}) \\ \vdots \end{bmatrix}$$

Marginalize out all irrelevant Examples i.e eg not in training or testing set.

$$\begin{bmatrix} f(x^{(1)}) \\ \vdots \\ f(x^{(m)}) \\ \hline f(x_*^{(1)}) \\ \vdots \\ f(x_*^{(n_*)}) \end{bmatrix} = N\left(\begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} K(x^{(1)}x^{(1)}) & \cdots & K(x^{(1)}, x_*^{(n_*)}) \\ & & \\ - & - & - \end{bmatrix} \right)$$

→ PSD (Mercer Theorem)

$$\begin{bmatrix} \vec{f} \\ \vec{f}_* \end{bmatrix} = N\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix} \right)$$

$y = f(x) + \varepsilon$ * $\vec{\varepsilon} \sim N(0, \sigma^2 I)$.

$$\begin{bmatrix} Y \\ Y_* \end{bmatrix} = \begin{bmatrix} f \\ f_* \end{bmatrix} + \begin{bmatrix} \varepsilon \\ \varepsilon_* \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \end{bmatrix}, \begin{bmatrix} K(X,X) + \sigma^2 I & K(X, X_*) \\ K(X, X_*) & K(X_*, X_*) + \sigma^2 I \end{bmatrix} \right)$$

Posterior Predictive distribution

$Y_* | Y, X, X_* \sim N(\mu_*, \Sigma_*)$

$$\mu_* = K(X_*, X)\left[K(X,X) + \sigma^2 I\right)]^{-1} Y.$$

$$\Sigma_* = K(X_*, X_*) + \sigma^2 I - K(X_*, X)\left[K(X,X) + \sigma^2 I\right]^{-1} K(X, X_*)$$

Basically Kernel functions and Covariance ~~functions~~ Matrix are essentially the same.

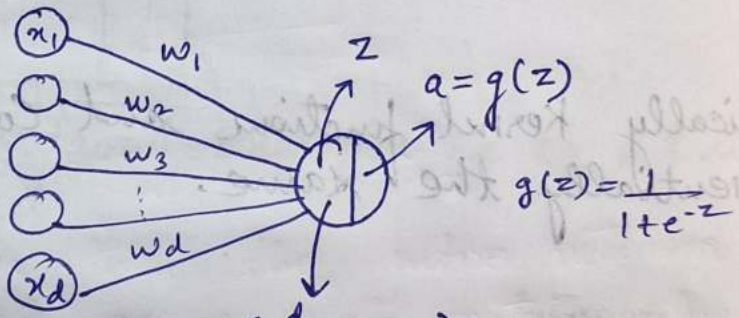
Lecture 10 : Deep Learning - I

In Deep learning the neural networks gives a way in which features themselves are learned automatically.

Models we have seen before - given features we construct linear model. With neural networks not only are we learning that model given the features but we are also learning what the right set of features are.

$x \in \mathbb{R}^d$

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ \sin(x) \\ \vdots \end{bmatrix}$$



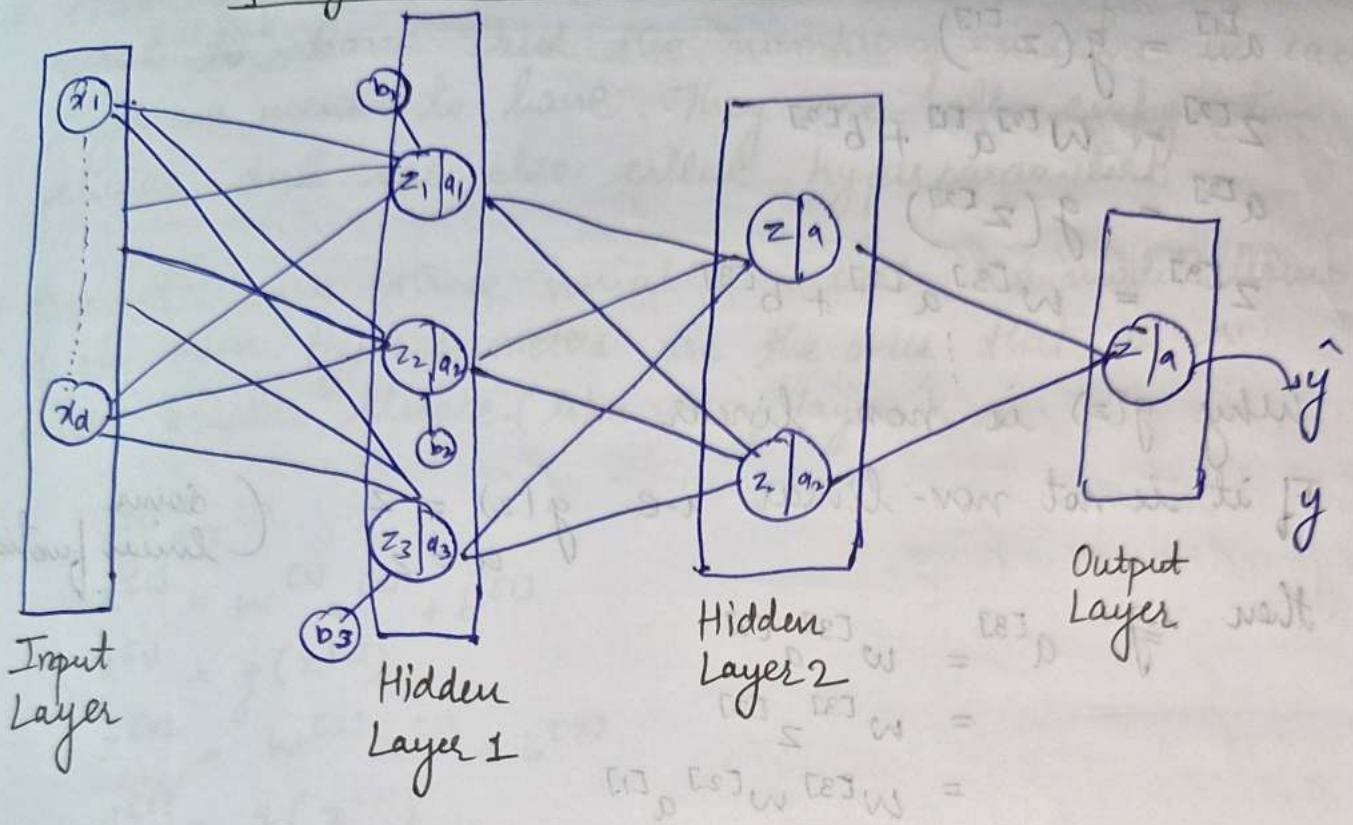
$$\Rightarrow \left(\sum_{i=1}^{d} x_i w_i\right) + b \qquad \{\theta^T x\} \rightarrow w, b$$

$$\Rightarrow z = x^T w + b$$

This is just logistic regression in the example, we will just find $\hat{y}$, y (label) then loss and perform gradient descent

But in Neural networks we do it differently.

Fully connected Network



$W_{ij}^{[l]}$ - connection (It is a weigth matrix)

$b_i^{[l]}$ - biad

$z_i^{[l]}$ - $W \cdot ( ) + b$ (linear combination)

$a_i^{[l]}$ - $g(z)$

$x - a^{[0]}$

$$z_1^{[1]} = \sum_j W_{1j}^{[1]} a_j^{[0]} + b_1$$

$$z_2^{[1]} = \sum_j W_{2j}^{[1]} a_j^{[0]} + b_2$$

$$a_1^{[1]} = g(z_1^{[1]})$$

No. of neuron in $l-1^{th}$ layer

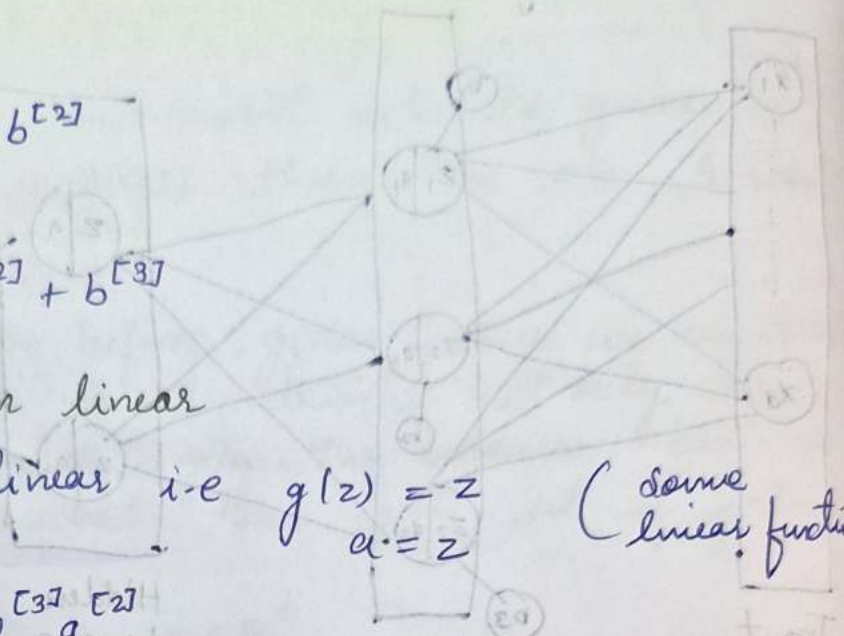
No. of neuron in l^{th} layer

$W^{[l]}$

In vector notation

$$\underbrace{z^{[1]}}_{3} = \underbrace{W^{[1]}}_{3 \times d} \underbrace{a^{[0]}}_{d} + \underbrace{b^{[1]}}_{3}$$

$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$
$$z^{[3]} = w^{[3]} a^{[2]} + b^{[3]}$$



Why $g(z)$ is non linear

If it is not non-linear i.e $g(z) = z$ (some linear function)
$a = z$

Then e.g. $a^{[3]} = w^{[3]} a^{[2]}$

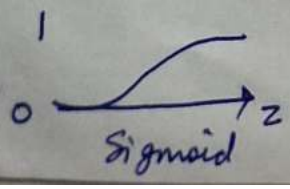
$$= w^{[3]} z^{[2]}$$
$$= w^{[3]} w^{[2]} a^{[1]}$$
$$= w^{[3]} w^{[2]} z^{[1]}$$
$$a^{[3]} = w^{[3]} w^{[2]} \cdot w^{[1]} \cdot x = \tilde{w} x$$

If g is a linear function then the entire network can now be represented as a single matrix, which means we haven't gone beyond the scope of linear models.

So g be non-linear is essential to have depth be meaningful. Otherwise no matter how many levels deep the network is. It can always be collapsed into a single matrix and it will effectively be as expressive as a single layer network.

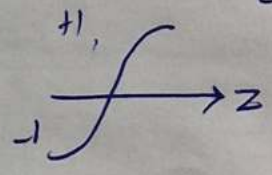
Commonly used $g(z)$

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

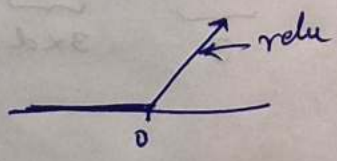


Sigmoid

$g = \tanh = \frac{e^{z} - e^{-z}}{e^{z} + e^{z}}$



$g = \text{Relu} = \max(z, 0)$



From the fully connected network - The No. of layers we want to have and the number of neurons in each layer we want to have. They are all configuration choices and are also called hyperparameters.

Parameters are those variables which the model learns of its own. Hyperparameters are the ones that we as model builders choose. (like no. of layers).

#
$$a^{[0]} = x^{(i)}$$
$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$
$$\vdots \qquad \vdots$$
$$a^{[L]} = g(z^{[L]})$$
$$\hat{y}^{(i)} = a^{[L]}$$

$$\Rightarrow L(y^{(i)}, \hat{y}^{(i)}) = -\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)})\right]$$

For l in $1, 2, \ldots, L$

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial L}{\partial W^{[l]}}$$
$$b^{[l]} = b^{[l]} - \alpha \frac{\partial L}{\partial b^{[l]}}$$

} update

Backpropagation mathematically is just a chain rule. Algorithmically, we calculate it in a way such that it is memory efficient and we reuse a lot of the computations when calculating.

Wouldn't all the neurons learn the same thing in the first layer? Aren't we learning multiple copies of same logistic regression.

* If we perform an initialization where entire network is initialized to 0. Then it will happen (Multiple copies).
* In order to avoid this problem we perform Random Initialization. It is a necessary step also called symmetry breaking.

Xavier / He Initialization

$$W_{ij}^{[l]} \sim N\left(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}\right)$$

OR

$$W_{ij}^{[l]} \sim \text{Unif}\,[-0.1, 0.1]$$

} Initialize W, b

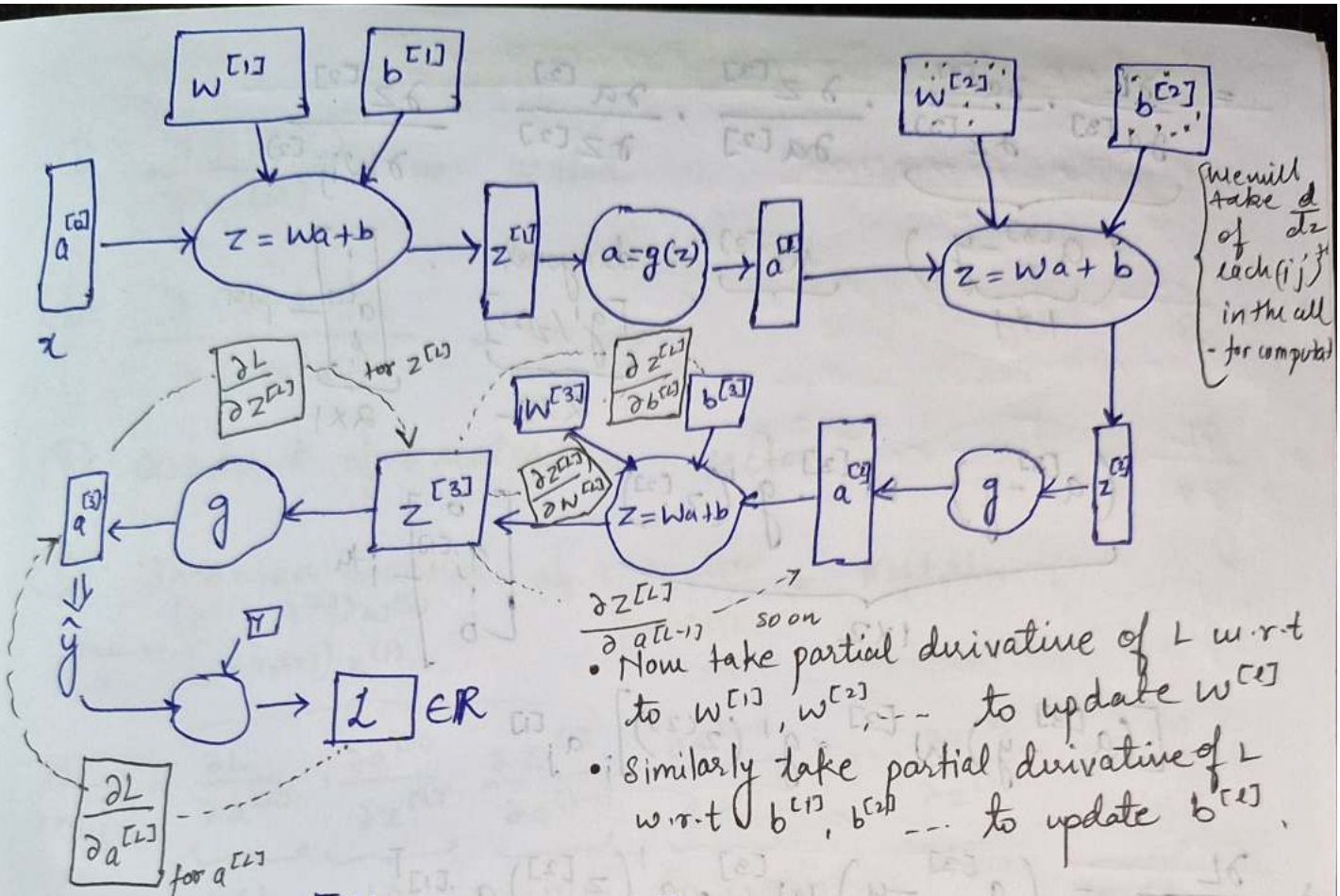
$$W_{ij}^{[l]} = W_{ij}^{[l]} - \alpha \frac{\partial L}{\partial W_{ij}^{[l]}}$$

For t in 1, ...

For l in 1, 2, ..., L

$$W^{[l]} = W^{[l]} - \alpha \frac{\partial L}{\partial W^{[l]}}$$

$$b^{[l]} = \text{.....}$$



- Now take partial derivative of L w.r.t to $w^{[1]}, w^{[2]}, \ldots$ to update $w^{[2]}$
- Similarly take partial derivative of L w.r.t $b^{[1]}, b^{[2]} \ldots$ to update $b^{[2]}$.

\# $\frac{\partial L}{\partial w^{[2]}} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}^{[2]}} & \cdots & \frac{\partial L}{\partial w_{13}^{[2]}} \\ \vdots & \ddots & \\ & & \frac{\partial L}{\partial w_{23}^{[2]}} \end{bmatrix}$

$$\frac{\partial L}{\partial z^{[3]}} = \frac{\partial}{\partial z^{[3]}} \left[-y \log \hat{y} - (1-y) \log (1-\hat{y}) \right]$$

$$= \frac{\partial}{\partial z^{[3]}} \left[-y \log \sigma(z^{[3]}) - (1-y) \log (1 - \sigma(z^{[3]})) \right]$$

$$= \cdots$$

$$= a^{[3]} - y$$

\# $\frac{\partial L}{\partial w_{ij}^{[2]}} = \frac{\partial L}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial w_{ij}^{[2]}}$

$$= \frac{\partial L}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial w_{ij}^{[2]}}$$

$$= \underbrace{\frac{\partial L}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}}}_{(a^{[3]} - y)} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z_i^{[2]}}{\partial w_{ij}^{[2]}}$$

$(a^{[3]} - y)$ is 1×1; $w^{[3]}$ is 1×2; diagonal $[g'(z^{[2]})]$ is 2×2; $\begin{bmatrix} 1 \\ a_i^{[1]} \\ 1 \end{bmatrix}$ ← i^{th} is 2×1

$$\underbrace{(a^{[3]} - y)\, w^{[3]} \cdot g'(z^{[2]})}_{1 \times 2} \quad \begin{bmatrix} 0 \\ a_j^{[1]} \\ 0 \end{bmatrix} \leftarrow j^{th}$$

$$\left[(a^{[3]} - y)\, w^{[3]} \cdot g'(z^{[2]})\right]_i a_j^{[1]}$$

$$\Rightarrow \frac{\partial L}{\partial W^{[2]}} = (a^{[3]} - y)\, w^{[3]} \cdot g'(z^{[2]})\, a_j^{[1]T}$$

Network View

$f: \mathbb{R}^n \to \mathbb{R}^m$

$\frac{\partial f}{\partial x} \leftarrow \mathbb{R}^{m \times n}$ (Jacobian)

The chain rule of calculus tell us that because all these functions are compositions where the output of one becomes the input of the other and so on. We can take the local derivatives of each of these and multiply them out. This is the M.V chain rule of calculus.

Cases

① $\frac{\partial (\text{scalar})}{\partial (\text{scalar})}$ = scalar

② $\frac{\partial (\text{vector})}{\partial (\text{scalar})}$ = vector

③ Gradient of a scalar = vector

④ Jacobian matrix of a vector function = matrix

$(y^{(i)} - a^{[L]})\, w^{[L]}$

$(y - h_\theta(x)) \cdot x^{(i)}$

$$\frac{\partial L}{\partial w_{ij}^{[2]}} = \underbrace{\frac{\partial L}{\partial a^{[L]}}}_{1\times 1} \cdot \underbrace{\frac{\partial a^{[L]}}{\partial z^{[L]}}}_{1\times 1} \cdot \underbrace{\frac{\partial z^{[L]}}{\partial a^{[L-1]}}}_{1\times m_{L-1}} \cdot \underbrace{\frac{\partial a^{[L-1]}}{\partial z^{[L-1]}}}_{m_{L-1}\times m_{L-2}} \cdots \underbrace{\frac{\partial a^{[2]}}{\partial z^{[2]}}}_{m\times m} \cdot \underbrace{\frac{\partial z^{[2]}}{\partial w_{ij}^{[2]}}}_{m\times 1}$$

1×1

$1\times m_{L-1}$

$1\times m \cdot m\times 1 = 1\times 1$

The chain of Jacobians will always condense to a scalar.

Backpropagation is an algorithm that tells us, what is the most optimal way to compute these gradients by reusing the intermediate computations as much as possible.

Lecture 11 : Deep Learning - II

Logistic regression

$\frac{\partial L}{\partial \theta} = [y - h_\theta(x)] \cdot x$

$\frac{\partial L}{\partial x} = [y - h_\theta(x)] \cdot \theta$

Therefore, derivative of loss w.r.t a will give us and derivative of loss w.r.t w then a will show up.

$$\frac{\partial a^{[l-1]}}{\partial z^{[l-1]}} = \begin{bmatrix} g'(\cdot) & & \\ & g'(\cdot) & \\ & & \ddots \\ & & & g'(\cdot) \end{bmatrix}$$

(a: $\uparrow$ $\downarrow$, z: $\longleftrightarrow$)

$$a_i = g(z_i)$$

$$\frac{\partial a_j}{\partial z_i} = 0 \text{ if and only if } i = j$$

$\frac{\partial z^{[l]}}{\partial a^{[l-1]}} = W^{[l]}$ $\Leftrightarrow$ $\frac{\partial z^{[2]}}{\partial W_{ij}^{[2]}} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ a_j \\ 0 \\ \vdots \end{bmatrix}$ $\leftarrow i^{th}$

$$[z_i] = [W_i] a_i$$

$$\boxed{z_i = \sum_j W_{ij} a_j + b_i}$$

$z_k \rightarrow$ the rest is 0.

$b_1 \quad \cdots \quad b_K$

$$[\ \cdots\ ] \begin{bmatrix} 0 \\ \vdots \\ a_j \\ 0 \end{bmatrix} i^{th} \text{ position} \Rightarrow b_i a_j$$

$$\frac{\partial L}{\partial W_{ij}^{[2]}} = \delta^{[2]} \cdot \begin{bmatrix} 0 \\ \vdots \\ a_j \\ \vdots \\ 0 \end{bmatrix}$$

$$\Rightarrow \boxed{\frac{\partial L}{\partial z^{[2]}} \cdot \begin{bmatrix} 0 \\ \vdots \\ a_j \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial z^{[2]}} \end{bmatrix}_i a_j}$$

$$\Rightarrow \boxed{\frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}} \left(a^{[1]}\right)^T}$$

$$\Rightarrow \boxed{W^{[1]} = W^{[1]} - \alpha \frac{\partial L}{\partial W^{[1]}}}$$

Training

\# We don't always do stochastic gradient descent. We perform minibatch gradient descent. In this instead of sampling one example at random, we sample a batch of examples at random.

$$J(a,b) \sum_{i=1}^{B} \mathcal{L}\left(y^{(i)}, \hat{y}^{(i)}\right)$$

B is the size of minibatch

$$\underbrace{Z^{[1]}}_{m} = \underbrace{\underbrace{W^{[1]}}_{m \times d} \underbrace{x^{(i)}}_{d \times 1}}_{m \times 1} + \underbrace{b^{[1]}}_{m \times 1}$$

$$= \underbrace{W^{[1]}}_{m \times d} \underbrace{\left[x^{(1)}, x^{(2)} \ldots x^{(B)}\right]}_{d \times B} + \underbrace{b^{[1]}}_{m \times 1}$$

$$\Rightarrow m \times B + m \times 1 \rightarrow m \times B$$

We are trying to add two matrix of different dimensions ($m \times B$ and $m \times 1$). But this addition is not possible. The softwares that we use for training ML (tensorflow, Numpy, Pytorch etc). All these work in this scenario of adding ($m \times B$ to $m \times 1$) with the help of Broadcasting.

$$(m \times B + (m \times 1) \longrightarrow m \times B)$$

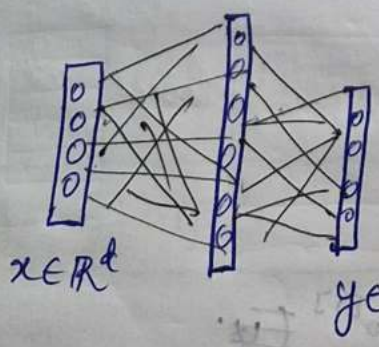
Every Neural network comes explicitely with a Kernel $\Rightarrow$ $Ker(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)})$

Universal Approximation theorem.

If you have a function with given region of interest of the inputs. For any smooth / continuous function f, there exist a neural network with a finite number of hidden layers that can approximate f to an arbitrary degree of precision.

$$y = f(x) \quad , \quad x \in \mathbb{R}^d, y \in \mathbb{R}^k$$

Exists a neural network with 1 hidden layer

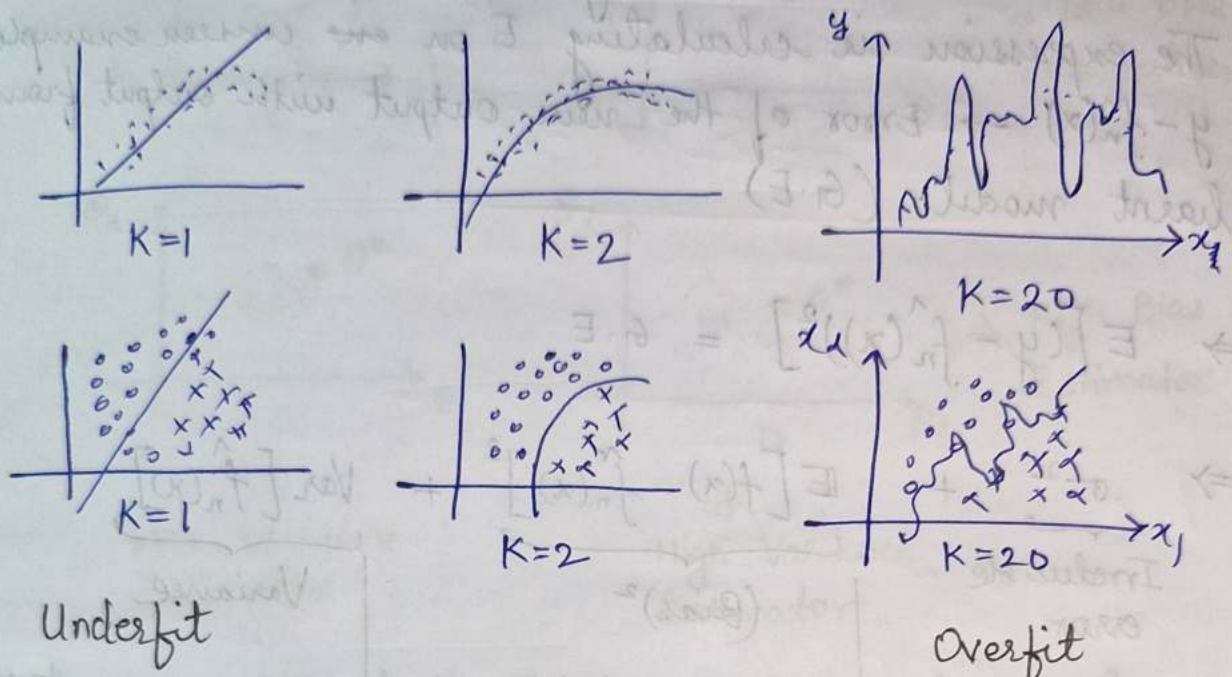


$\forall$ eg $\varepsilon = 10^{-6}$

$|\hat{f}(x) - f(x)| < \varepsilon$

However it doesn't tell you that how to recover that function.

Lecture 12 : Bias, Variance & Regularization



Generalization Error (G.E)

\# **Bias** : Component of G.E due to "Expressivity handicap".

\# **Variance** : Component of G.E due to finite sample of training set.

Squared error

$$y^{(i)} = f(x^{(i)}) + \varepsilon^{(i)} \quad , \quad E[\varepsilon] = 0, \quad V[\varepsilon] = \sigma^2$$

$$\text{Test error [G.E]} = \mathbb{E}\left[(y - \hat{f_n}(x))^2\right]$$

\# here x, y are pair from the test set (unseen example).

$\hat{f_n}(x)$ → It is the model we get by fitting on n training examples. n training examples were obtained using $y^{(i)} = f(x^{(i)}) + \varepsilon^{(i)}$.

→ Thus it is a Random variable.

$\mathbb{E}\left[(y - \hat{f}_n(x))^2\right]$: Expectation is over all ex. in training set and test examples.

* The expression is calculating E on unseen example.
$y - \hat{f}_n(x)$ → Error of the new output with output from learnt 'model' (G.E)

$$\Rightarrow \mathbb{E}\left[(y - \hat{f}_n(x))^2\right] = G.E$$

$$\Rightarrow \underbrace{\sigma^2}_{\text{Irreducible error}} + \underbrace{\mathbb{E}\left[f(x) - \hat{f}_n(x)\right]^2}_{(\text{Bias})^2} + \underbrace{\text{Var}\left[\hat{f}_n(x)\right]}_{\text{Variance}}$$

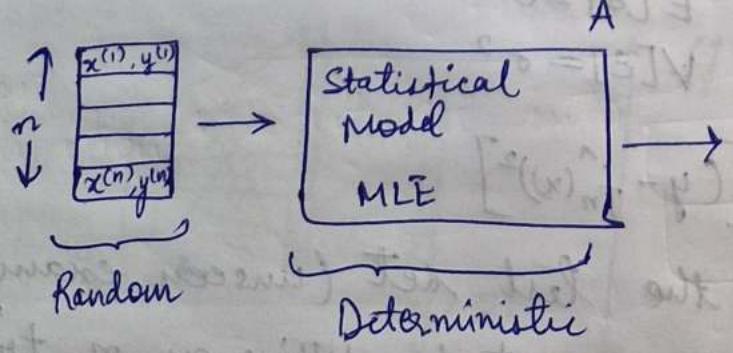
- Since the examples are noisy already

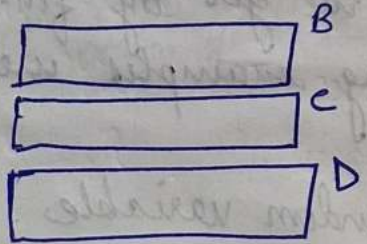
- A true underlying signal that we don't have access to. $f(x)$ will be different from the prediction made by our model
- Basically how systematically wrong we are

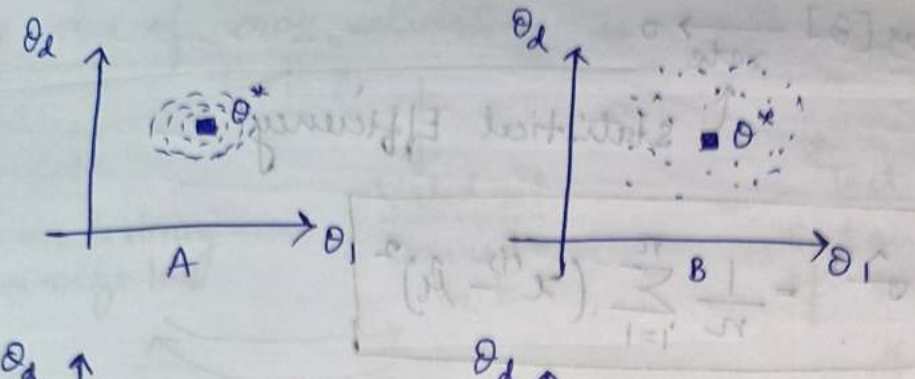
- If we were to repeat this example with new set of n examples. $\hat{f}_n(x)$ → diff value
- Sensitive to noise.

Bias - Variance

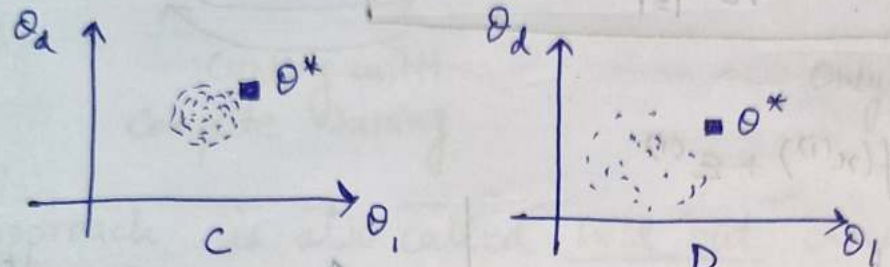
$(x, y) \sim \text{Dist}(\theta^*)$







⇒ Low Bias Estimation



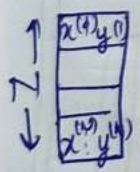
⇒ High Bias Estimator

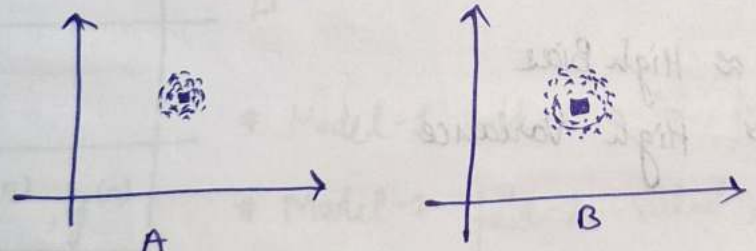
⇓
Low Variance Estimator

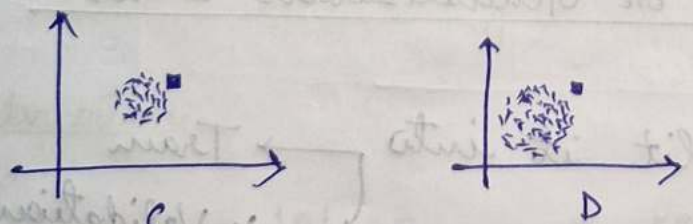
⇓
High Variance Estimator

This clearly suggest that Bias and Variance are not co-related, Knowing one doesn't give info about other.

Now with $n \to$ very high.







N →

$E[\hat{\theta} - \theta^*]$ = Bias

$Var[\hat{\theta}]$ = Variance

⇒ If Bias $\to 0$, $n \to \infty$ ⇒ Consistent Estimation

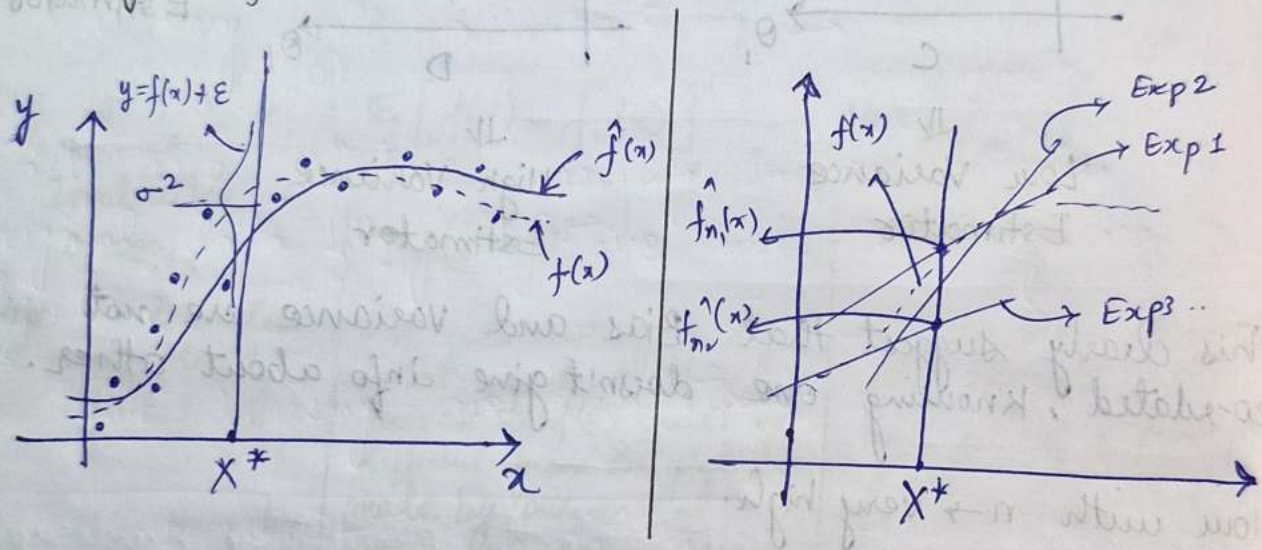
$\lim_{n\to\infty} \text{Var}[\hat{\theta}] \xrightarrow[\text{rate}]{} 0$

↑ Statistical Efficiency

MLE Biased

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \hat{\mu})^2$$

Now $y^{(i)} = f(x^{(i)}) + \varepsilon^{(i)}$



Underfitting ≈ High Bias

Overfitting ≈ High Variance

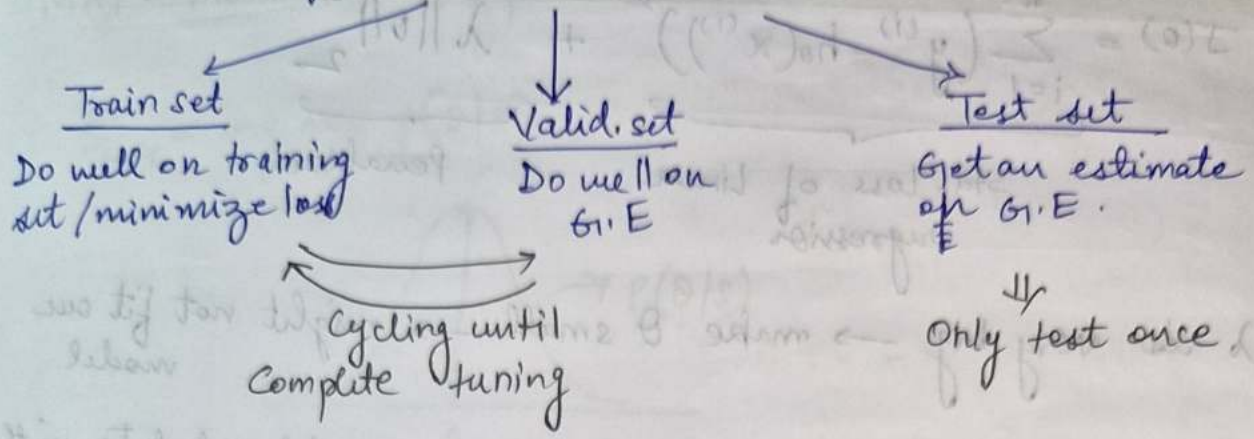
Goal: Do well on Generalization Error.

Cross Validation: split it into

- Train — e.g 70%
- Validation/dev — 20%
- Test — 20%

No matter what your hyperparameter decision is. The process of cycling between training and validation set is essential. Universal ML concept.

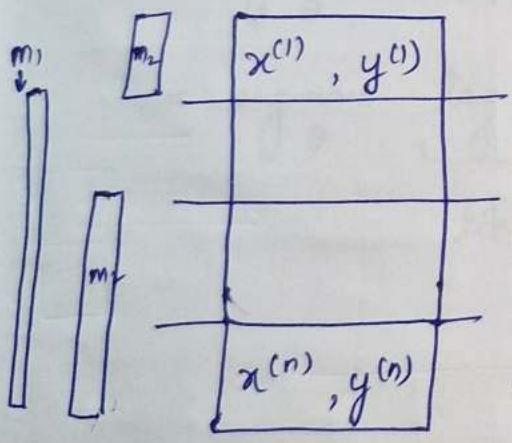
The idea of cross validation is



This approach is also called hold out cross validation technique. Since we have kept some data separate for test and val.

K-fold Cross Validation

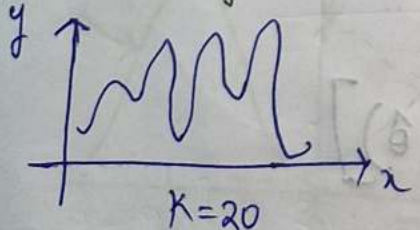
- Small Data sets.



Model-1 fold-1 valid, fold 2-n is training

Model-2 fold-2 valid, fold 3-n is training

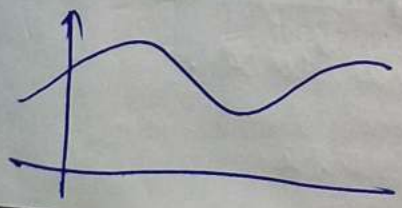
Regularization



$|\theta| >> 0$ Bad

Regularization is basically encouraging small $||\theta||$

Smooth curve



$$J(\theta) = \underbrace{\sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right)^2}_{\text{Std. loss of Linear regression}} + \underbrace{\lambda \|\theta\|_2^2}_{\text{Penalize}}$$

λ is very big → make θ small → might not fit our model

λ is very small (0) → make θ big → model fit data without any regularization.

λ acts as a relative weighing factor between the original data fitting objective and the Regularization objective

We tune λ using cross validation.

Bayesian Interpretation

$\theta \sim p(\theta)$ [Prior]

$S|\theta \sim$ [likelihood]

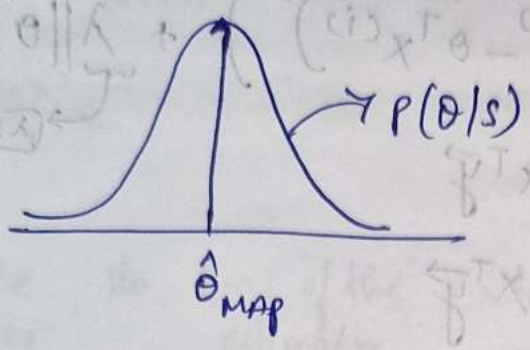
$\theta|S$ [Posterior]

$$P(\theta|S) = \frac{P(S|\theta) \cdot P(\theta)}{P(S)}$$

$$P(Y_*|X_*, X, Y) = E_{\hat{\theta} \sim P(\theta|S)}\left[P(y_*|X_*, \hat{\theta})\right]$$

MAP - Maximum - a-posteriori Parameter Estimate

$$\hat{\theta}_{MAP} = \arg\max_{\theta} p(\theta|S)$$



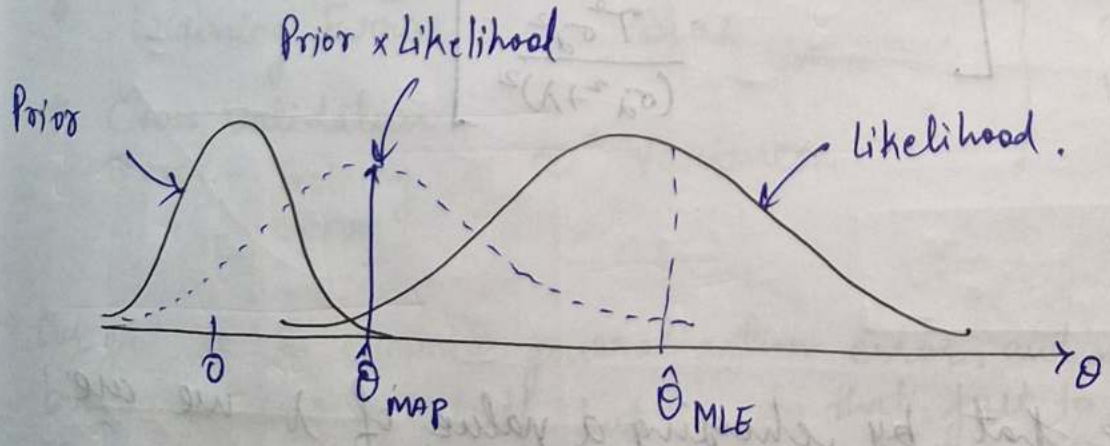
$$\hat{\theta} = \arg\max_{\theta} p(\theta|S)$$

$$p(\theta) \sim N(0, \sigma^2) = \arg\max_{\theta} \frac{p(S|\theta) \cdot p(\theta)}{p(S)}$$

$$= \arg\max_{\theta} p(S|\theta) \cdot p(\theta)$$

$$= \arg\max_{\theta} \underbrace{\log p(S|\theta)}_{\text{likelihood}} + \underbrace{\log p(\theta)}_{\lambda \|\theta\|^2}$$

$$p(\theta) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \exp\left\{-\frac{1}{2}\frac{(\theta)^2}{\sigma^2}\right\}$$



Lecture 13 : Statistical learning Uniform Convergence

L_2 - Regularized Linear Regression

$$J(\theta) = \left(\sum_{i=1}^{n} \left(y^{(i)} - \theta^T x^{(i)} \right)^2 \right) + \underbrace{\lambda \|\theta\|_2^2}_{\lambda > 0}$$

we know $\hat{\theta} = (X^T X)^{-1} X^T \vec{y}$

$$\hat{\theta}_n = (X^T X + \lambda I)^{-1} X^T \vec{y}$$

$$(X^T X + \lambda I)^{-1} = U \begin{bmatrix} (\sigma_1^2 + \lambda)^{-1} & & 0 \\ & \ddots & \\ 0 & & (\sigma_d^2 + \lambda)^{-1} \end{bmatrix} U^T$$

$$E[\hat{\theta}_n] = \left[U \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2 + \lambda} & & 0 \\ & \ddots & \\ 0 & & \frac{\sigma_d^2}{\sigma_d^2 + \lambda} \end{bmatrix} U^T \right] \theta^*$$

For Variance

$$Cov(\hat{\theta}_n) = U \begin{bmatrix} \frac{\tau^2 \sigma_1^2}{(\sigma_1^2 + \lambda)^2} & & \\ & \ddots & \\ & & \frac{\tau^2 \sigma_d^2}{(\sigma_d^2 + \lambda)^2} \end{bmatrix} U^T$$

$\varepsilon \sim N(0, \tau^2)$

$y = \theta^T x + \varepsilon$

We can see that by choosing a value of λ we are doing trade of between bias and variance.

$\lambda \uparrow \rightarrow$ Var $\downarrow$, Bias $\uparrow$ and vice versa

Bias Variance decomposition

$$MSE[\hat{f}_n] = \underbrace{\tau^2}_{\text{Irreducible error}} + \underbrace{E[\hat{f}_n(x^*) - f(x^*)]^2}_{\text{Bias}^2} + \underbrace{V[\hat{f}_n(x^*)]}_{\text{Variance}}$$

$$f(x) = \theta^{*T} x$$

Now

Bias of the predictor to Bias of the estimator relation

$$\begin{aligned} Bias(\hat{f}_n(x^*)) &= E[\hat{f}_n(x^*) - f(x^*)] \\ &= E[\hat{\theta}_n^T x^* - \theta^{*T} x^*] \\ &= E[\hat{\theta}_n - \theta^*]^T x^* \end{aligned}$$

$$\boxed{Bias(\hat{f}_n(x^*)) = Bias(\hat{\theta}_n)^T x^*}$$

$$\boxed{Var[\hat{f}_n] = x^T Cov[\hat{\theta}_n] x}$$

Heuristics for Bias and Variance

\# Training Error $\approx$ Bias

\# Cross Validation Error − Training error $\approx$ Variance.

$\approx$ Appox
$\neq$ Not equal

→ Our goal is to minimize generalization error, and not to go after bias or variance by itself but we do that just to minimize G.E. In order to do that, we have rough heuristic of bias and variance. Compare the two and go after the one which is bigger

→ We do not want to take any arbitrary step like get more data, add more regularization. Therefore we construct heuristic estimation.

To fight Bias

- Add more layers to Neural network (Make model larger)
- Reduce regularization
- Add more features.

To fight Variance

- Increase regularization
- Collect more data
- Simpler model

Statistical Learning theory

① Train and Test data ~ Same distribution (Assumption).

② Examples are sampled IID

Risk of hypothesis

$$\varepsilon(h) = \mathop{\mathbb{E}}_{(x,y)\sim D}[\text{loss}(y, h(x))]$$

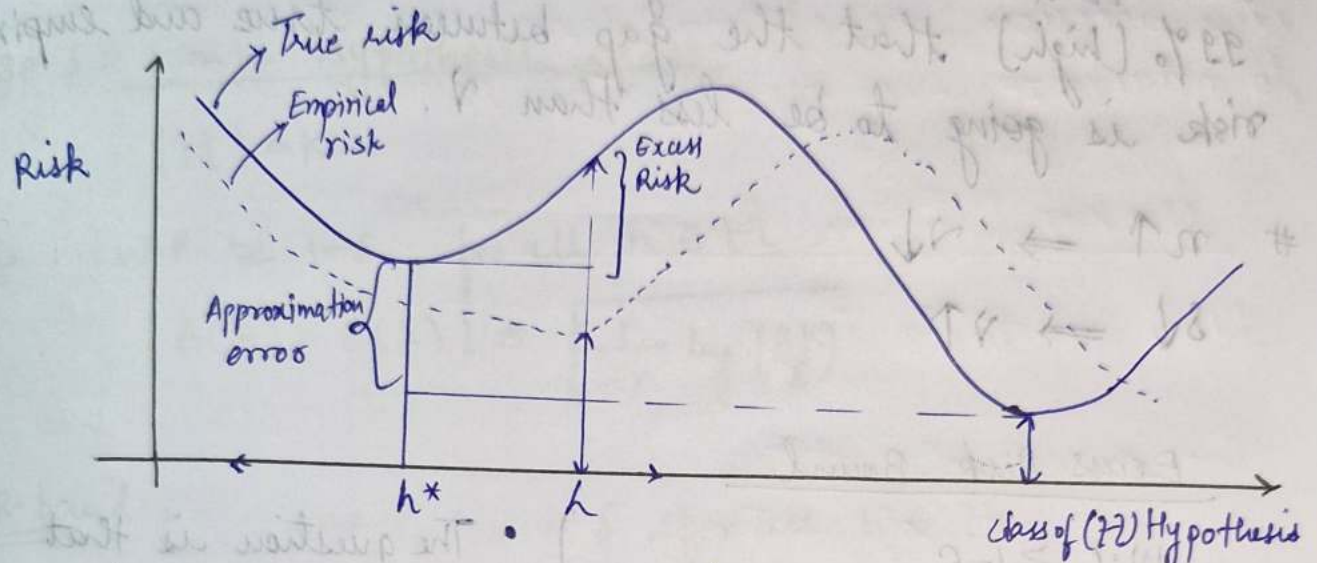
$$S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$$

Empirical Risk

$$\hat{\varepsilon}(h) = \frac{1}{|S|} \sum_{(x,y)\in S} \text{loss}(y, h(x))$$

Q// For a given h, what is the relation between $\hat{\varepsilon}(h)$ and $\varepsilon(h)$.

Q// How does the generalization error compare to the best possible generalization error.



- Hypothesis – Set of all hypothesis for all possible models.
- h^* – best in class hypothesis.

Step 1: Uniform Convergence.
Step 2: Excess Risk Bound.

Uniform Convergence: How the Empirical risk function as we increase the number of data points we have will probabilistically converge to true risk function uniformly across all points.

We want to minimize the gap uniformly across all hypothesis.

Statement: With probability (W.P) $\Rightarrow$ W.P $= 1-\delta$

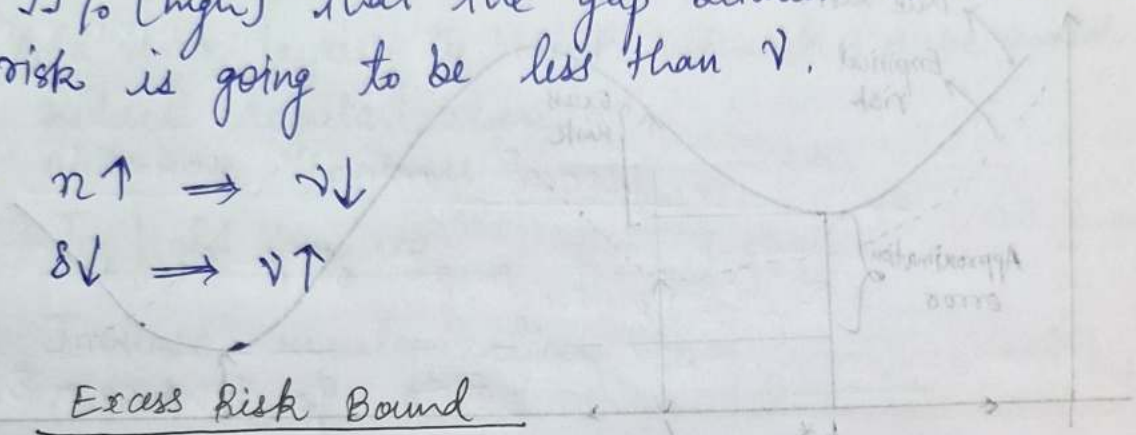
for all $h \in \mathcal{H}$

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \underbrace{\gamma}_{\text{Term}(n,\delta,\mathcal{H})}$$

* For all hypothesis on the H axis the gap of $|\varepsilon(h) - \hat{\varepsilon}(h)|$ is less than γ. If we are given a training set of size n, degree of confidence (δ) and H then we will get down γ with probability 99% (high) that the gap between true and empirical risk is going to be less than γ.

* $n \uparrow \Rightarrow \gamma \downarrow$

  $\delta \downarrow \Rightarrow \gamma \uparrow$



Excess Risk Bound

w.p $\geq 1-\delta$

$\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma$

Proof

$\varepsilon(\hat{h}) \leq \hat{\varepsilon}(\hat{h}) + \gamma$, $\left(\begin{smallmatrix}\text{U.C}\\ \text{at } \hat{h}\end{smallmatrix}\right)$

$\leq \hat{\varepsilon}(h^*) + \gamma$, [By definition of E.R.M]

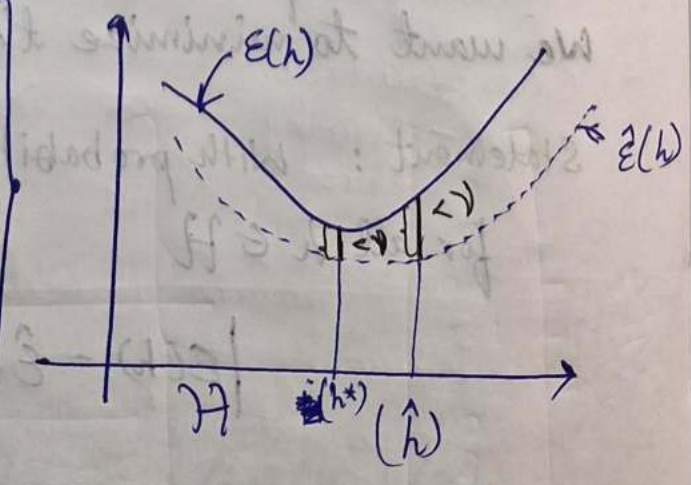
$\leq [\varepsilon(h^*) + \gamma] + \gamma$

, [U.C at h^*]

$\leq \varepsilon(h^*) + 2\gamma$

$$\boxed{\varepsilon(\hat{h}) \leq \varepsilon(h^*) + 2\gamma}$$

- The question is that uniform convergence gives the difference between $\varepsilon(h)$ and $\hat{\varepsilon}(h)$ i.e thick line (—) and dotted line (- - - -).
- But in Risk bound we need to find difference between both the points on $\varepsilon(h)$ i.e thick line (—)



This excess risk bound tells us how well our model is gonna perform in the real world relative to the best possible model that could have performed in the real world.

Case I: Finite Hypothesis class

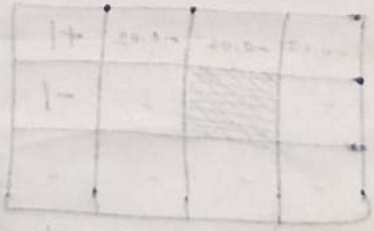
$$|\mathcal{H}| = K$$

U.C : $W.P \geq 1-\delta$, for all $h \in \mathcal{H}$

$$|\varepsilon(h) - \hat{\varepsilon}(h)| \leq \sqrt{\frac{1}{2n} \log\left(\frac{2K}{\delta}\right)}$$

E.R. bound : $W.P \geq 1-\delta$, for all $h \in \mathcal{H}$

$$\varepsilon(\hat{h}) \leq \underbrace{\varepsilon(h^*)}_{\min_{h \in H} \varepsilon(h)} + 2\sqrt{\frac{1}{2n} \log\left(\frac{2K}{\delta}\right)}$$



Lecture 14 : Reinforcement Learning - I

In reinforcement learning we are making multiple decisions, one decision at a time and with each decision, we obtain some reward. Our goal is to maximize this accumulated reward into the future.

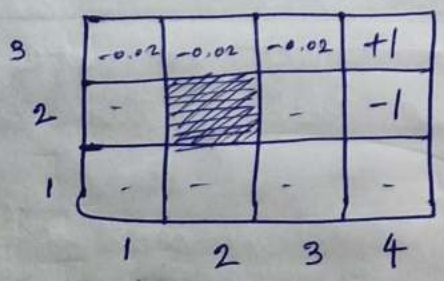
Markov Decision Process (MDP)

It is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$

S - a set of states
A - a set of actions
P_{sa} - Transition probabilities
$\gamma \in [0,1]$ - Discount factor

$R : S \times A \to \mathbb{R}$
or
$R : S \to \mathbb{R}$ - Reward function

Ex



$S = \{(1,1), (1,2), \ldots\ldots, \}$

$|S| = 11$

$A = \{N, S, W, E\}$

P_{sa} and γ

$P_{(1,3)N} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.1 \\ 0.1 \\ 0.8 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ — $(1,4)$, $(1,2)$, $(2,3)$

The reward function can be represented as a function of state

$$R(S) \longrightarrow R$$

or it can be modelled as function $R(s,a)$

$$R(s,a) \longrightarrow R$$

↳ The reason is because taking different actions we have different costs, so that is why both s, a becomes important.

Trial / Episode / Trajectory

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \gamma^3 R(s_3)$$

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots\cdots$$

$$s_1 \sim P_{s_0 a_0} \quad s_2 \sim P_{s_1 a_1}$$

We want to get larger rewards sooner and negative rewards later.

Policy

Policy $\pi : S \longrightarrow A$

Value $V^\pi : S \longrightarrow R$

$$V^\pi(s) = \mathbb{E}\left[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots \mid s_0 = s, \pi\right]$$

* Value can be defined as the long term reward we gain following a policy.

$s_0 \rightarrow$ not random

$s_1, s_2, \cdots \rightarrow$ Random

$$V^{\pi}(s) = R(s) + \gamma E\underbrace{[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \dots]}_{V^{\pi}(s_1)}$$

$$V^{\pi}(s) = \underbrace{R(s)}_{\text{Immediate reward}} + \underbrace{\gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')}_{\text{Discount factor times, expectation of future value of each particular state times the probability that we end up in that particular state by taking the action.}} \rightarrow \text{Bellman Equation}$$

The action itself is based on the policy. $\boxed{A = \pi(s)}$

$$\boxed{V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')}$$ Bellman Equation

Given Policy, how do we find Value

Policy	Value
$S \rightarrow A$	$S \rightarrow \mathbb{R}$

(I) $\pi \rightarrow V^{\pi}$

(II) $\pi \leftarrow V$

Case I: $\pi \rightarrow V^{\pi}$ (Policy to Value)

$$V^{\pi}(s) = \underbrace{R(s)}_{\text{Known}} + \underbrace{\gamma \sum_{s' \in S} P_{s\pi(s)}}_{\text{Known}} \underbrace{V^{\pi}(s')}_{\text{Solving for}}.$$

$$V^{\pi}(s_1) = R(s_1) + \gamma \sum_{s' \in S} P_{s\pi(s_1)}(s') V^{\pi}(s')$$

$$V^{\pi}(s_2) = R(s_2) + \gamma \sum_{s' \in S} P_{s\pi(s_2)}(s') V^{\pi}(s')$$

$$\vdots$$

$$V^{\pi}(s_{|S|}) = \text{- - - -}$$

$$\underbrace{V^{\pi}}_{|S|} = \underbrace{R}_{|S|} + \underbrace{\gamma P^{\pi}}_{|S| \times |S|} \underbrace{V^{\pi}}_{|S|}$$

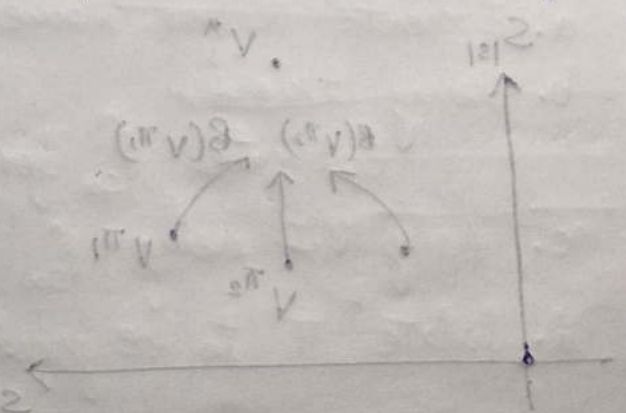
$$\Rightarrow \boxed{V^{\pi} = [I - \gamma P^{\pi}]^{-1} R}$$

Where

$$V^{\pi} = \begin{bmatrix} - \\ - \\ \vdots \\ - \end{bmatrix} \begin{matrix} s_1 \\ s_2 \\ \vdots \\ \end{matrix} \quad \begin{matrix} V^{\pi}(s_1) \\ V^{\pi}(s_2) \\ \vdots \end{matrix}$$

$$P^{\pi} = \underbrace{\begin{matrix} s_1 \\ \\ s_i \\ \\ \end{matrix} \begin{bmatrix} \text{—} P_{s_1\pi(s_1)} \text{—} \\ \\ \text{—} P_{s_i\pi(s_i)} \text{—} \\ \\ \end{bmatrix}}_{|S|} \Big\} |S|$$

\# This approach gives us a way of starting from a policy π and calculating the long term values for each of the states if we follow the policy at all times.



Optimal Value function

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

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

optimal policy

$$\pi^*(s) = \arg\max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

Algorithm used for computation

- **Value iteration algorithm:**

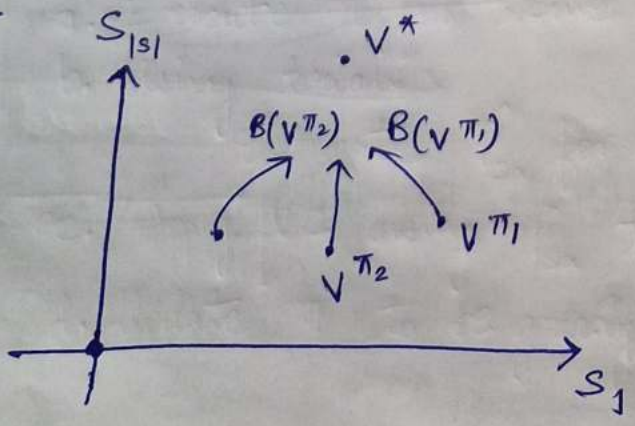
1. For each step/state s, initialize $V(s) = 0$
2. Repeat until convergence:

$O \rightarrow$ order $O(|S|^2|A|)$ { For every state s, update

$$V(s) = R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$$

(Bellman Backup operator).

Intuition



Bellman operator is called a contraction mapping because for any two inputs to the operator. The corresponding outputs will be closer to each other than the corresponding two inputs.

The point to which they all are converging to (V^*) is called a fixed point.

- **Policy iteration algorithm (Greedy Policy)**

1. Initialize π randomly
2. Repeat until convergence

   (a) Set $V = V^{\pi}$ $\leftarrow$ $O(|S|^3)$

   $O(|A| \cdot |S|)$ { (b) For each state s, set

$$\pi(s) = \arg\max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$$

For large state spaces, policy iteration can be much more expensive than value iteration. However, with policy iteration after a finite no. of steps, we get the exact optimal value. Whereas with value iteration we get closer and closer to V^* (optimal value) but we may never reach ~~each~~ it.

Lecture 15: Reinforcement Learning - II

P_{sa} is unknown

Model Based vs Model-free

Model = P_{sa}

Trials

$s_0^{(1)} \xrightarrow{a_0^{(1)}} s_1^{(1)} \xrightarrow{a_1^{(1)}} s_2^{(1)} \longrightarrow \dots\dots$

$s_0^{(2)} \xrightarrow{a_0^{(2)}} s_1^{(2)} \xrightarrow{a_1^{(2)}} \dots\dots$

} Learning Model!

$$\hat{P}_{sa}(s') = \frac{\text{No. of times we took action } a \text{ at state } s \text{ and got to state } s'}{\text{No. of times we took action } a \text{ at state } s.}$$

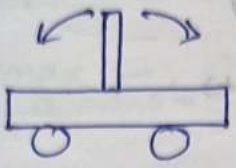
$$= \frac{0}{0} \Rightarrow \text{Uniform}$$

Algorithm (with Model learning)

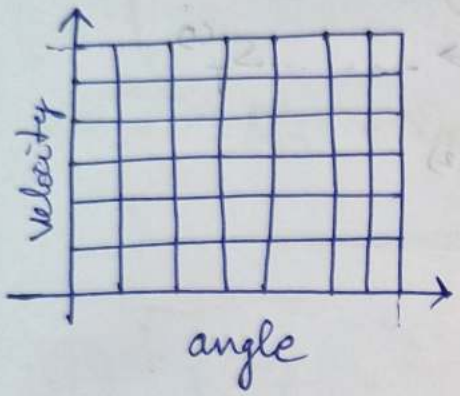
1. Initialize π randomly
2. Repeat {
   a. Execute π for some time
   b. Using trials
      Estimate $\hat{P}_{sa}$ for each s, a
   c. Apply V.I using $\hat{P}_{sa}$ to get updated $V^{(t)}$
   d. Update π to be greedy w.r.t $V^{(t)}$

Continuous States

$S \rightarrow$ Space



: Balancing a stick on a moving cart



- Discretization help us to solve continuous problems.
- But has limitations.

This is discrete now

Curse of Dimentionality

12 parameters = $\underbrace{x, y, z}_{\text{Coordinates}}$, $\underbrace{\theta, \phi, \psi}_{\text{angles}}$, $\underbrace{\dot{x}, \dot{y}, \dot{z}}_{\text{velocities}}$, $\underbrace{\dot{\theta}, \dot{\phi}, \dot{\psi}}_{\text{angular velocities}}$

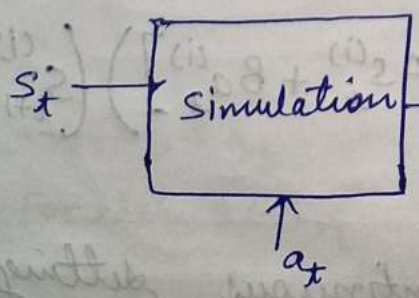
- Consider this as an example for flying a helicopter
- If we descritize it in each param. to 10 values. then

  Total parameters = 10^{12} (Massive state size)

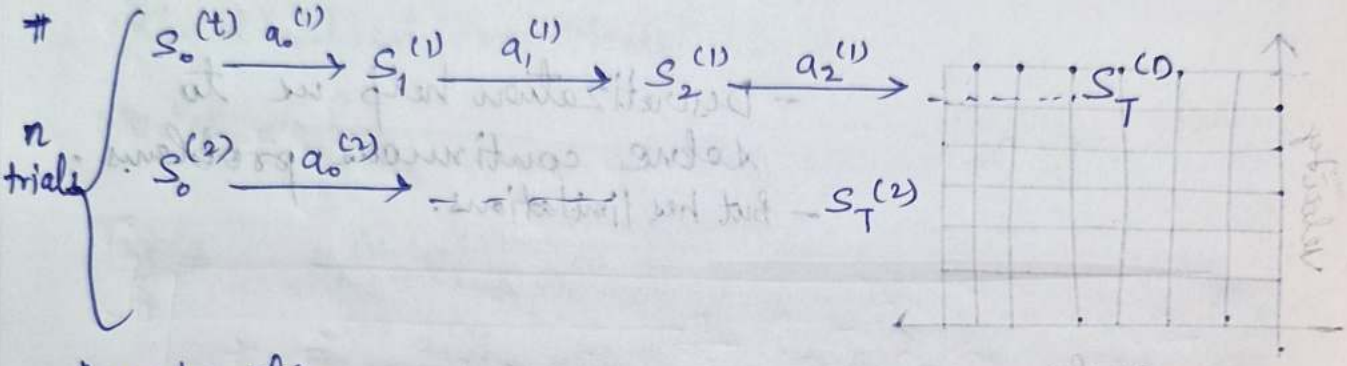
- This is a shortcomig of descritization. (Limitation)

Value function Approximation

S is continuous and we don't descritize.



$$s^{(t+1)} = f(s^{(t)}, a^{(t)})$$
$$= As^{(t)} + Ba^{(t)}$$



Loss function

$$J(A,B) = \sum_{i=1}^{n} \sum_{t=0}^{T-1} \left\| s_{t+1}^{(i)} - (As_t^{(i)} + Ba_t^{(i)}) \right\|_2^2$$

Now if we minimize it, then it will give our learnt model.

$$\hat{A}, \hat{B} = \underset{A,B}{\operatorname{argmin}} (J(B,A))$$

$$\Rightarrow s^{(t+1)} = \hat{A}s^{(t)} + \hat{B}a^{(t)} \rightarrow \text{Deterministic model}$$

$$s^{(t+1)} = \hat{A}s^{(t)} + \hat{B}a^{(t)} + \varepsilon^{(t)} \rightarrow \text{Stochastic}$$

$$\boxed{\varepsilon \sim N(0, \Sigma)}$$

where,

$$\hat{\Sigma} = \frac{1}{n} \frac{1}{T-1} \left[\left(s_{t+1}^{(i)} - \left[As_t^{(i)} + Ba_t^{(i)} \right] \right) \left(s_{t+1}^{(i)} - \left[As_t^{(i)} + Ba_t^{(i)} \right] \right)^T \right]$$

⇑

Learn a model in continuous setting

Now making the value function also continuous.

$$V(s) = R(s) + \gamma \max_a E_{s' \sim P_{sa}} [V(s')]$$

$$P_{sa} = N(\hat{A}_s + \hat{B}_a ; \hat{\Sigma})$$

$$V(s) = R(s) + \gamma \max_a \int \underbrace{P_{sa}(s')}_{\text{Gaussian Pdf}} \cdot V(s')\, ds'$$

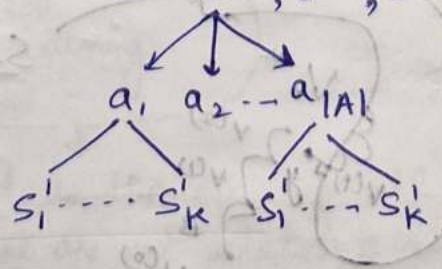
Value function approximation

$$V(s) = \theta^T \phi(s)$$

Algorithm for Continuous state model value function approximation:

$$s^{(t+1)} = A s^{(t)} + B a^{(t)} + \varepsilon$$
$$V(s) = \theta^T \phi(s)$$

1. Randomly samples n states $s^{(1)}, s^{(2)}, s^{(3)}, \ldots, s^{(n)} \in S$



2. Initialize $\theta = 0$
3. Repeat {
   For $i = 1, \ldots, n$ {
      For each $a \in A$ {

sample $s_1', \ldots, s_k' \sim P_{s^{(i)}a}$ (e.g. $= N(As^{(i)} + Ba, \hat{\Sigma})$)

Set $q(a) = \frac{1}{k} \sum_{j=1}^{k} R(s^{(i)}) + \gamma V(s_j)$ $\quad [V(s) = \theta^T \phi(s)]$

} J_c

Set $y^{(i)} = \max_a q(a)$ $\quad [\text{label } V(s^{(i)})]$

} J_b

Set $\hat{\theta} = \arg\min_\theta \sum_{i=1}^{n} \left(\theta^T \phi(s^{(i)}) - y^{(i)} \right)^2$

} J_a

$$q(s^{(i)}, a) \approx R(s^{(i)}) + \gamma E_{s' \sim P_{sa}} [V(s')]$$

$$y^{(i)} \approx R(s^{(i)}) + \max_a \gamma E_{s' \sim P_{sa}} [V(s')]$$ Bellman Backup operator

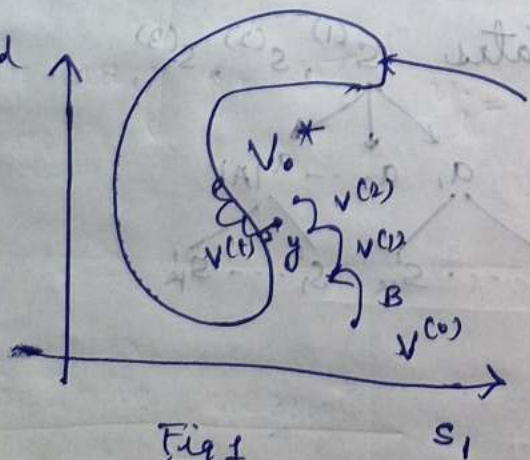


Fig 1

Set of all V s.t. $V(s) = \theta^T \phi(s)$

~~Value~~

In this setting we are learning the dynamics of A and B and using those dynamics we are performing value iteration using Bellman backup operator. Performing in such a way we use those fitted value function.

But now as can be seen in Fig 1. we are limiting ourselves in the space of value functions. Which means this whole algorithm is limited to some set of all V.

We don't know env. but learn it

	Value function based	Policy Based
Discrete finite	V.I	P.I
Infinite state space	Fitted V.I	

Model free

↳ Here we do not care how the environment works. and also we don't care about it

Q-function =

$Q(s,a) \approx$ Value

$V^*(s)$

Q-learning

What is the sum of all future rewards if we are at state s and take action a.

Lecture 16 - Unsupervised learning

K-means, GMM and EM.

In unsupervised learning, we are given the $\{x^{(1)}, x^{(2)}, \ldots x^{(n)}\}$ but not the corresponding y labels.

$x^{(i)} \in \mathbb{R}^d$

K-Means Algorithm

$S = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ $\quad x^{(i)} \in \mathbb{R}^d$

K- cluster (given to us)

Steps

1. Initialize cluster centroids

$$\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^d \text{ randomly}$$

2. Repeat until convergence:

Cluster identity - For every i, set $c^{(i)} = \arg\min_j \|x^{(i)} - \mu_j\|_2^2$

Recalculate μ_j (centroid) - For every j, set

$$\mu_j = \frac{\sum_{i=1}^{n} 1\{c^{(i)} = j\} \cdot x^{(i)}}{\sum_{i=1}^{n} 1\{c^{(i)} = j\}}$$

Co-ordinate descent

$$J(c, \mu) = \sum_{i=1}^{n} \|x^{(i)} - \mu_{c^{(i)}}\|_2^2$$

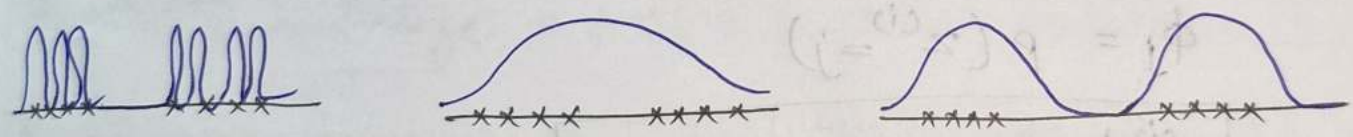
(Distortion function)

The step 1 is to minimize this function by holding μ fixed and optimizing c. and step 2 is to then minimizing J again by holding the c fixed and optimizing it with respect to μ.

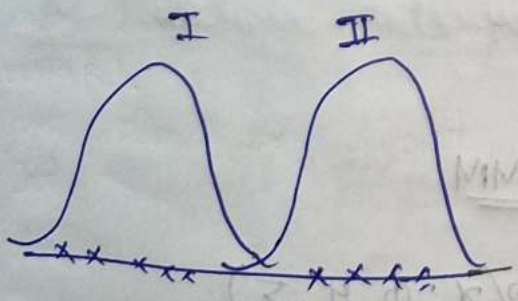
K-means is co-ordinate descent of the distortion function J.

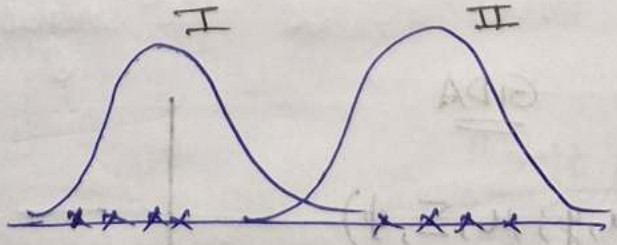
The μ and c we end up with can be different at different time, as it depends upon the random initialization. (Non convex)

Density Estimation



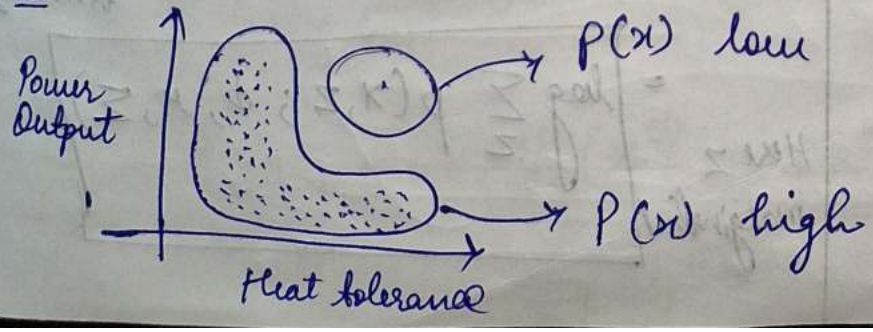
Gaussian Mixture Model





To come up with p(x).

Ex



Basically our objective is to construct an algorithm for mixture of gaussians and it is based purely on intuition. Then to describe this framework (EM) to rederive (GMM) and check if we end up with the same algorithm.

GMM

$S = \{x^{(1)}, \dots, x^{(m)}\}$ ⟶ Given training set

$z^{(i)} \sim$ Multinomial distribution (ϕ).

$$\phi_j \geq 0 \qquad \sum_{j=1}^{k} \phi_j = 1$$

$$\phi_j = p(z^{(i)} = j)$$

$$\boxed{x^{(i)} | z^{(i)} = j \sim N(\mu_j, \Sigma_j)}$$

$z^{(i)}$ – latent variable : Random variable that we have not observed

GDA

$$\log p(x, y; \mu, \Sigma, \phi)$$
$$= l(\mu, \Sigma, \phi)$$

GMM

$$\log p(x; \phi, \mu, \Sigma)$$
$$= l(\mu, \Sigma, \phi)$$

(Don't know z)

$$= \boxed{\log \sum_z p(x, z; \phi, \mu, \Sigma)}$$

Here z Marginalized

Notation for our study throughout from now.

$z \rightarrow$ Latent (unobserved)

$p(z) \rightarrow$ class prior / prior

$p(x,z) \rightarrow$ model

$p(z|x) \rightarrow$ posterior

$p(x) \rightarrow$ Evidence.

Our goal is to maximize the likelihood using the evidence.

[Inspired by K-means]

Repeat until convergence.

Randomly initialize μ, ϕ, Σ

E-Step For each i, j set:

$$w_j^{(i)} = p(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma)$$

Probability estimation

$$p(z|x) = \frac{p(x|z) \cdot p(z)}{\sum_z p(x|z) \cdot p(z)}$$

M-Step Update parameters

$$\phi_j = \frac{1}{n} \sum_{i=1}^{n} w_j^{(i)}$$

"Soft K-means"

$$\mu_j = \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)}}$$

e.g. $K=3$

$P(z^{(i)}=j \mid x^{(i)})$ $\begin{bmatrix} 0.1 \\ 0.7 \\ 0.2 \end{bmatrix}$ $K=1$ $K=2$ $K=3$ $\begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

EM Algorithm (Expectation Maximization)

It is an algorithm where we perform MLE in the presence of latent (unknown) variables.

True model : $p(x, z; \theta)$

when z is unobserved

Maximization $l(\theta) = \log p(x; \theta)$ { z is marginalised }

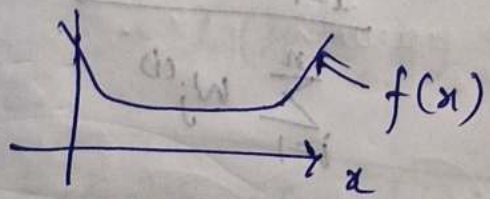
The EM gives us a framework for achieving this. (Indirect way)

EM algorithm is one of the key algorithm for modern Deep learning / Deep generative models. It is used across all algos.

Jensen's Inequality

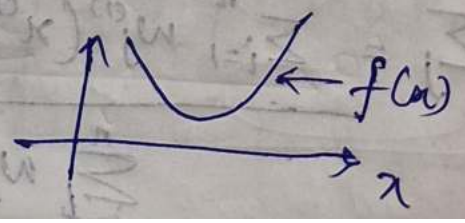
f to be convex

$f''(x) \geq 0$ for all x



f is strictly convex

$f''(x) > 0$ for all x

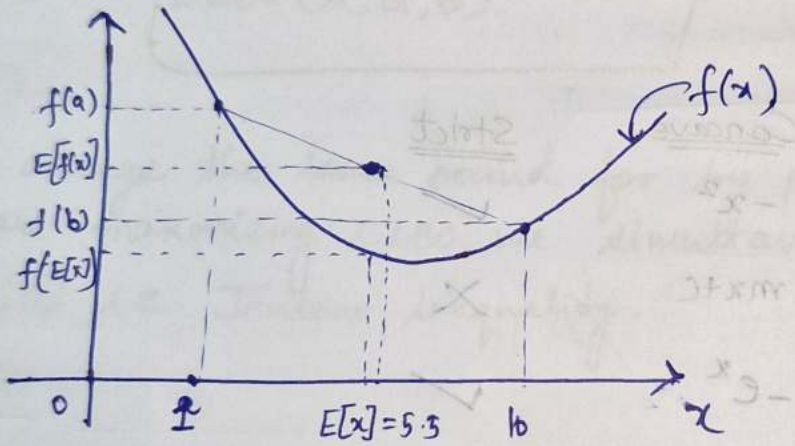


XXXX

I $\boxed{E[f(x)] \geq f(E[x])}$ where f is convex, $x \rightarrow$ any random variable

If f is strictly convex, then

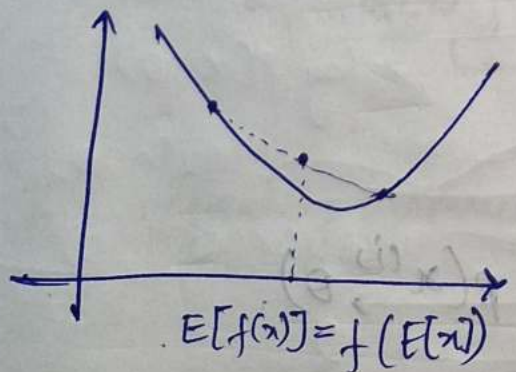
II If $E[f(x)] = f(E[x])$ then $X = E[X]$ w.p 1. → Means x is constant



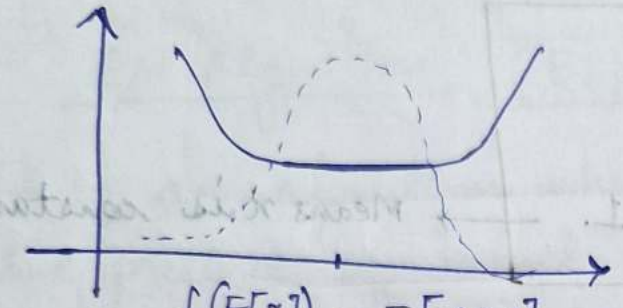
$E[x] = \int x \cdot p(x)\, dx$

$E[f(x)] = \int f(x) \cdot p(x)\, dx$

Strictly convex



The reason why we require f to be strictly convex is because if f were not strictly convex, we could have a case where $f(x)$ is flat and



$f(E[x]) = E[f(x)]$ even though x is not constant

Ex

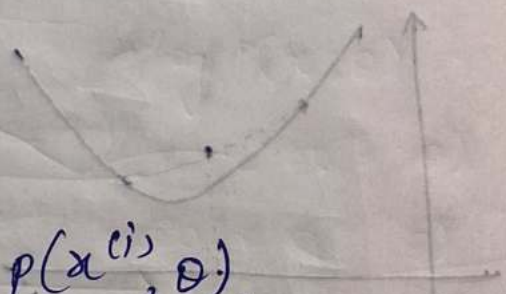
Convex	Concave	Strict
$f(x) = x^2$	$-x^2$	✓
$f(x) = mx+c$	$mx+c$	✗
e^x	$-e^x$	✓
$-\log(x)$	$\log(x)$	✓

Jensen's inequality for concave function. just the signs gets reversed.

If f is concave

eg $f(x) = \log(x)$

$$E[\log(X)] \leq \log E[X]$$



Goal is to maximize $\sum_{i=1}^{n} \log p(x^{(i)}; \theta)$.

Basically Maximize $\log p(x; \theta)$

$$\Rightarrow \log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

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

$\begin{cases} Q(z) > 0 \text{ for all } z \\ Q(z) \to \text{some arbitrarily prob. fnc. on } z \end{cases}$

$$\Rightarrow \log p(x;\theta) = \log \underset{z\sim Q}{E}\left[\frac{p(x,z;\theta)}{Q(z)}\right]$$

$$\geq \underset{z\sim Q}{E}\left[\log \frac{p(x,z;\theta)}{Q(z)}\right]$$

$$\log p(x;\theta) \geq \text{ELBO}(x;Q,\theta)$$

\# ELBO is always the lower bound for $\log p(x;\theta)$.
If we are maximizing ELBO, we simultaneously maximize $\log p(x;\theta)$ i.e Jensens inequality.

Are there cases when

$$\log p(x;\theta) = \text{ELBO}(x;\theta,\theta) \longrightarrow \text{Yes}$$

If f is strictly convex and going back to II part of Jensen's inequality.

$$\Rightarrow \underset{z\sim Q}{E}\left[\log \frac{p(x,z;\theta)}{Q(z)}\right]$$

$\frac{p(x,z;\theta)}{Q(z)} \longrightarrow$ Constant

\# The inequality becomes equality if $\frac{p(x,z;\theta)}{Q(z)} = \text{contt.}$ (↓ constant)

$$Q(z) = \frac{1}{c} p(x, z; \theta)$$

$$Q(z) \propto p(x, z; \theta)$$

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

$$= \frac{p(x, z; \theta)}{p(x; \theta)}$$

$$\boxed{Q(z) = p(z|x; \theta)}$$

EM Algorithm:

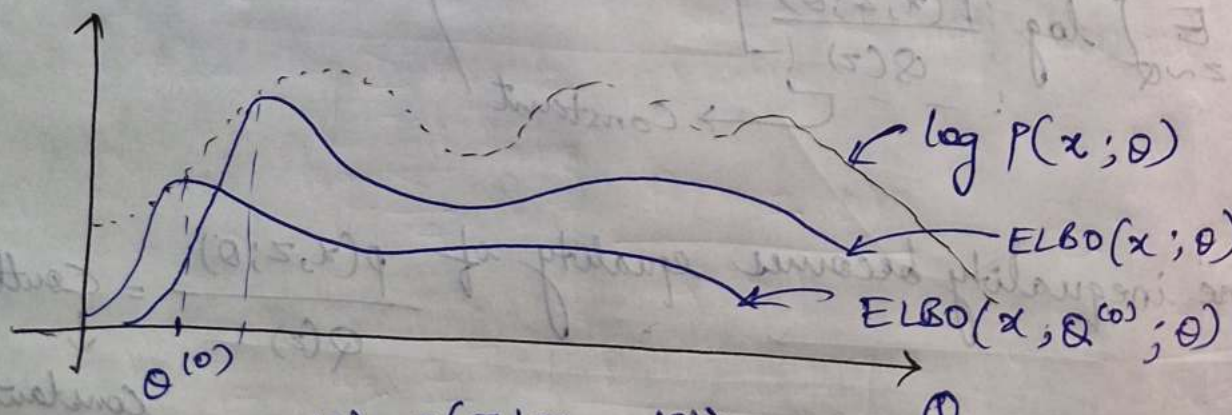
E-Step

For each i, set

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

M-Step

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



$$\Rightarrow \theta^{(0)} = p(z|x; \theta^{(0)})$$

Lecture 17 - Factor Analysis & ELBO

Recap

① K-MEANS

Given: K, $\{x^{(i)}\}_{i=1}^{n}$

Randomly init $\boxed{\mu_j \in \mathbb{R}^d}$, $1 \le j \le k$

Loop till convergence

"E-Step": For each i

Set $\boxed{c^{(i)} = \arg\min_j \|x^{(i)} - \mu_j\|_2^2}$

"M-Step": For each j $\quad \dfrac{\sum_{i:c^{(i)}=j} x^{(i)}}{\sum_{i:c^{(i)}=j} 1} = \mu_j$

$$\boxed{J(c,\mu) = \sum_{i=1}^{n} \left\| x^{(i)} - \mu_{c^{(i)}} \right\|_2^2}$$

② Gaussian Mixture Model [Soft K-means]

Given: K, $\{x^{(i)}\}_{i=1}^{n}$

Model: $z^{(i)} \sim \text{Multinomial}(\phi)$

$x^{(i)} | z^{(i)} \sim N(\mu_{z^{(i)}}, \Sigma_{z^{(i)}})$

Randomly Initialize $\boxed{\phi, \mu, \Sigma}$

Loop till convergence:

"E-Step" For each:

Set $\boxed{w^{(i)} = p(z^{(i)} | x^{(i)}; \phi, \mu, \Sigma)}$ {Use Bayes Rule}

"M-Step": For each j

$$\# \quad \phi_j = \frac{1}{n} \sum_{i=1}^{n} w_j^{(i)}$$

$$\# \quad \mu_j = \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}}$$

$$\# \quad \Sigma_j = \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)}}$$

③ E·M Algo

($P(x, z; \theta)$ if z was observed but z is not observed here)

$$\log p(x; \theta) = \log \sum_z p(x, z; \theta)$$

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

$$\left(\log P(x; \theta)\right) = \log \mathop{\mathbb{E}}_{z \sim Q} \left[\frac{P(x, z; \theta)}{Q(z)}\right]$$

$\mathop{\mathbb{E}}_{z \sim Q}[g(z)] = \sum_z Q(z) g(z)$

$g(z) = \frac{P(x, z; \theta)}{Q(z)}$

$$\log P(x;\theta) \geq \mathop{\mathbb{E}}_{z\sim Q}\left[\log \frac{P(x,z;\theta)}{Q(z)}\right]$$

$$= ELBO(x;Q,\theta)$$

Corollary

$$\boxed{\log P(x;\theta) = ELBO(x;Q,\theta) \quad \text{iff} \quad Q(z) = P(z|x;\theta)}$$

EM Algorithm [General form]

Randomly init θ

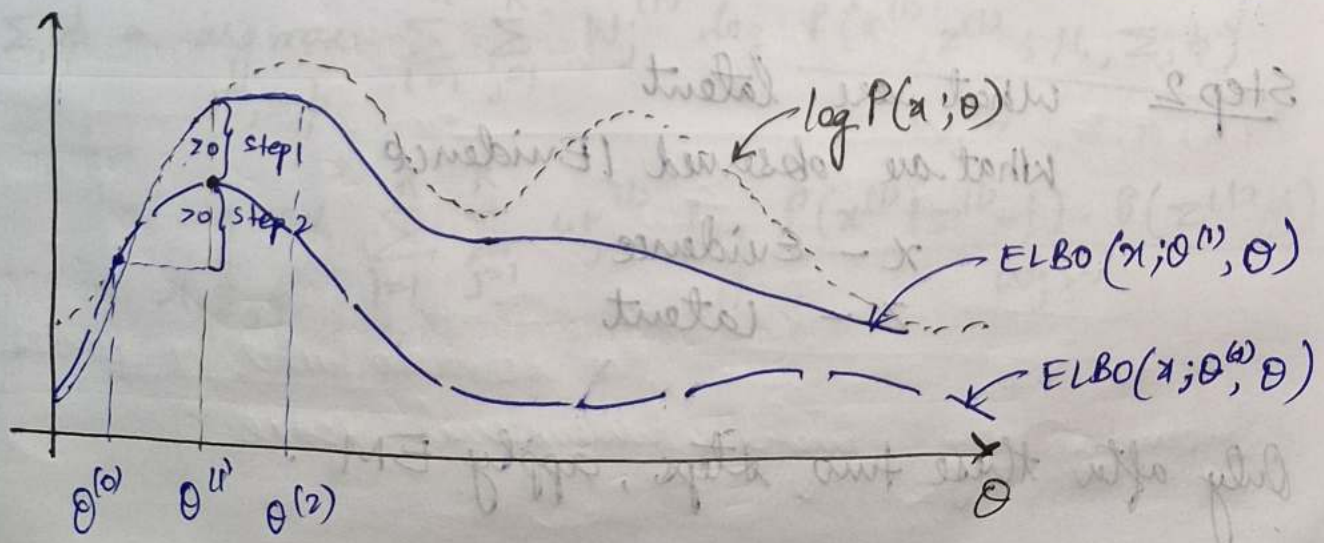
Loop till convergence

"E-Step": For each i,

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

: "M-Step"

$$\boxed{\begin{aligned} Q &= \arg\max_{\theta} \sum_{i=1}^{n} ELBO(x^{(i)};\theta) \\ &= \arg\max_{\theta} \sum_{i=1}^{n} \sum_{z} Q_i(z^{(i)}) \log \frac{P(x,z;\theta)}{Q(z^{(i)})} \end{aligned}}$$



Proof of EM Convergence

For every t

$$l(\theta^{(t+1)}) \geq l(\theta^{(t)}) \qquad \{ l(\theta) = \log P(x;\theta) \}$$

$$l(\theta^{(t+1)}) \geq ELBO(x; Q^{(t)}, \theta^{(t+1)}) \qquad [\text{Jensen's Inequality}]$$

$$\geq ELBO(x; Q^{(t)}, \theta^{(t)}) \qquad [\text{Due to M-step}]$$

$$l(\theta^{(t+1)}) = l(\theta^{(t)}) \qquad [\text{corollary of Jensen's inequality}]$$

GMM via EM

Step 1 Data generating model information (Must be known)

Write out $p(x, z; \theta)$

$$z \sim \text{Multinomial}$$
$$x|z \sim N(\mu_z, \Sigma_z)$$
$$p(z) \cdot p(x|z) = p(x, z)$$
$$= p(x, z; \phi, \mu, \Sigma)$$

Step 2 What are latent

What are observed / Evidence

x – Evidence

z – Latent

Only after these two steps, apply EM.

There are two types of unknown here

→ ① Parameters (μ, ϕ, Σ) → global unknown (M step)

→ ② x, z → specific to each example (E step)

E Step

For each i

$$Q_i(z^{(i)} = j) = P(z^{(i)} | x^{(i)}; \phi, \mu, \Sigma)$$

$$= \frac{P(x|z) \cdot P(z=j)}{P(x)}$$

$$= \frac{P(x|z=j) \cdot P(z=j)}{\sum_z P(x|z=j) \cdot P(z=j)}$$

$$= \frac{\frac{1}{(2\pi)^{d/2} |\Sigma_j|^{1/2}} \exp\left\{\frac{-1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)\right\} \phi_j}{\sum_j (\text{same})}$$

M-Step

$$\mu, \Sigma, \phi = \arg\max_{\mu, \Sigma, \phi} \sum_{i=1}^{n} \sum_{j=1}^{k} w_j^{(i)} \log \frac{P(x^{(i)}, z^{(i)}; \mu, \Sigma, \phi)}{w_j^{(i)}}$$

$$= \arg\max_{\mu, \Sigma, \phi} \sum_{i=1}^{n} \sum_{j=1}^{k} w_j^{(i)} \log \frac{P(x^{(i)} | z^{(i)} = j) \cdot P(z^{(i)} = j)}{w_j^{(i)}}$$

$\hat{\phi} = \frac{1}{n} \sum_{i=1}^{n} w_j^{(i)}$

$\hat{\mu}_j = \frac{\sum_{i=1}^{n} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{n} w_j^{(i)}}$

$\hat{\Sigma}_j = \frac{\sum_{i=1}^{n} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{n} w_j^{(i)}}$

Factor Analysis

$x^{(i)} \in \mathbb{R}^d$

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

dimensions >>> n (examples) { usually examples are much larger the dimensions }

$\Sigma \in \mathbb{R}^{d \times d}$

$\Sigma = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$

$\Sigma_{rank} = n \times 1 = n$ rank

$P(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\{\ \}$

We are here considering the scenario when x comes from a very high dimensional space.

Now we assume $z^{(i)} \sim N(0, I)$, I - K dimensional

$z^{(i)} \in \mathbb{R}^K$ (K lower dimensional subspace) → z resides in $K < n$

$\Rightarrow x^{(i)} | z^{(i)} \sim N(\mu + Lz, \Psi)$

$L \in \mathbb{R}^{d \times K}$

$\Psi \in \mathbb{R}^{d \times d}$ diagonal

There is this low dimensional (K) subspace where $K < n$, and the latent variable z that resides in the K dimensional subspace, which get mapped on to higher dimensional space (x). Using L (can be called an uplifting Matrix as it takes from $K \to d$), Lz: d dimensions, shifted by some offset μ and random diagonal noise Ψ.

Here z is continuous, not separate for each example. In factor analysis we are essentially trying to find the subspace of x. The finding lower dimensional subspace (z) and assuming that x that we observe has corresponding latent variable z. The way x is generated from z is through the relation $N(\mu + Lz, \Psi)$.

Goal: $\log p(x; \mu, L, \Psi)$.

probabilistic model

We want to learn a model eg $p(x)$ so that we can make systems like for anamaly detection.

Using the set of assumptions, we want to maximize / learn $\log p(x; \mu, z, \mu)$ → This will be done by EM.

Model

$$\boxed{\begin{aligned} z &\sim N(0, I) \\ x|z &\sim N(\mu + Lz, \Psi) \end{aligned}} = \boxed{\begin{aligned} z &\sim N(0, I) \\ \varepsilon &\sim N(0, \Psi) \\ x &= \mu + Lz + \varepsilon \end{aligned}}$$

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim N\left(\mu_{zx}, \Sigma\right)$$

$$\mu_{zx} = \begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \Sigma_z & \Sigma_{xz} \\ \Sigma_{xz}^T & \Sigma_x \end{bmatrix} = \begin{bmatrix} I & L^T \\ L & LL^T + \Psi \end{bmatrix}$$

Model detail

$$P(x, z)$$

$$\Rightarrow \begin{bmatrix} z \\ x \end{bmatrix} \sim N\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & L^T \\ L & LL^T + \Psi \end{bmatrix}\right)$$

Parameters: μ, L, Ψ

Observe – x

Latent – z

Flow of our approach

E step: $p(z|x)$

M step: $\mu, L, E = \text{argmax}\ ELBO$

Aside

$$\ell(\mu, L, \psi) = \log p(x; \mu, \Sigma)$$

$$x \sim \mathcal{N}(\mu, LL^T + \psi)$$

$$\log p(x) = \frac{1}{(2\pi)^{d/2} |LL^T + \psi|^{1/2}} \exp\left\{ -\frac{1}{2} (x-\mu)^T [LL^T + \psi]^{-1} (x-\mu) \right\}$$

\# There is No close form solution for this in L and therefore we come up with factor analysis i.e Expectation maximization. Estep, Mstep.

E Step

Calculate $p(z|x)$

Using previous proofs i.e. $\begin{bmatrix} z \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ \mu \end{bmatrix}, \begin{bmatrix} I & L^T \\ L & LL^T + \psi \end{bmatrix} \right)$

$$z|x \sim \mathcal{N}\left[\underbrace{L^T (LL^T + \psi)^{-1} (x - \mu)}, \underbrace{I - L^T (LL^T + \psi)^{-1} L} \right]$$

$$Q_i(z^{(i)}) = \mathcal{N}(\text{Mean}, \text{Covariance})$$

M Step

$$\mu, L, \psi = \arg\max_{\mu, L, \psi} \sum_{i=1}^{n} \int_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}, \mu, L, \psi)}{Q_i(z^{(i)})}$$

$$\Rightarrow \arg\max_{\mu, L, \psi} \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim Q_i} \left[\log P(x^{(i)}|z^{(i)}) + \cancel{\log p(z^{(i)})} - \cancel{\log Q_i(z^{(i)})} \right]$$

Because it does not have $z^{(i)}$ which we are optimizing

$$\Rightarrow \underset{\mu, L, \Psi}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim Q_i} \left[\log \frac{1}{(2\pi)^{d/2} |\Psi|^{1/2}} \exp \left\{ \frac{-1}{2} (x^{(i)} - \mu - Lz^{(i)}) \Psi^{-1} (x^{(i)} - \mu - Lz^{(i)}) \right\} \right]$$

$$\sum_{i=1}^{n} \mathbb{E}_{z^{(i)} \sim Q_i} \left[\left(\log \frac{1}{(2\pi)^{d/2}} \right) - \frac{1}{2} \log |\Psi| - \frac{1}{2} (x^{(i)} - \mu - Lz^{(i)}) \Psi^{-1} (x^{(i)} - \mu - Lz^{(i)}) \right]$$

$$\Rightarrow \hat{L} = \left(\sum_{i=1}^{n} (x^{(i)} - \mu) \mu_{z|x}^{T} \right) \left(\sum_{i=1}^{n} \mu_{z|x} \mu_{z|x}^{T} + \Sigma_{z|x} \right)$$

$$\Rightarrow \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x^{(i)}$$

$$\Rightarrow \Phi = \frac{1}{n} \sum_{i=1}^{n} x^{(i)} x^{(i)T} - x^{(i)} \mu_{z|x}^{T} L^{T} - L \mu_{z|x} x^{(i)T} + L (\mu_{z|x} \mu_{z|x}^{T} + \Sigma_{z|x}) L^{T}$$

$$\Rightarrow \Psi_{ii} = \Phi_{ii}$$

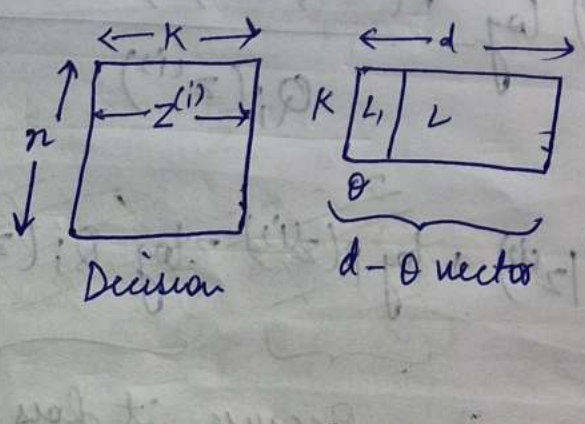
Factor analysis sum up

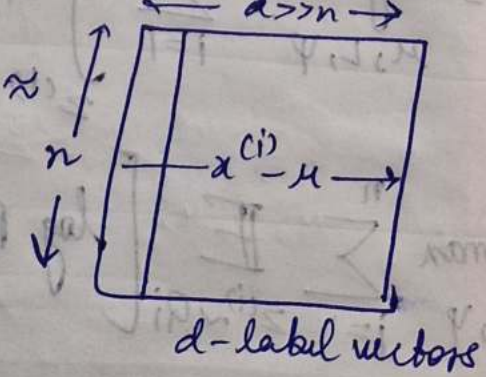
Model: $z \sim N(0, I) \in \mathbb{R}^k$

$x|z \sim N(\mu + Lz, \Psi) \in \mathbb{R}^d$

$k < n << d$

Parameters: $\mu \in \mathbb{R}^d$, $L \in \mathbb{R}^{d \times k}$, $\Psi \in \mathbb{R}^{d \times d}$





Lecture 18: Principal & Independent CA

On the basis of our previous derivations / proof we can make some best guesses about z and x.

$$\Rightarrow \mu_{z^{(i)}|x^{(i)}} \approx \hat{z}^{(i)} \quad \left(\text{Best estimate of } \hat{z}^{(i)} \text{ given } x\right)$$

$$\Rightarrow L\,\mu_{z^{(i)}|x^{(i)}} = L\hat{z}^{(i)} \approx x^{(i)} - \mu = \hat{x}^{(i)} - \mu$$

(It can be seen as reconstruction of x.)

Now writing L

$$L = \left[\sum_{i=1}^{n} y^{(i)} z^{(i)T}\right]\left[z^T z + \Sigma\right]^{-1}$$

$$L^T = \left(z^T z + \Sigma\right)^{-1} z^T y$$

For Ψ

$$\Psi_{ii} \approx \sum_{i=1}^{n} x^{(i)} x^{(i)T} - x^{(i)} \hat{x}^{(i)T} - \hat{x}^{(i)} x^{(i)T} + \hat{x}^{(i)} \hat{x}^{(i)T} + L \sum_{z^{(i)}|x^{(i)}} L^T$$

$$\Psi_{ii} \approx \left[\sum_{i=1}^{n} (x^{(i)} - \hat{x}^{(i)})(x^{(i)} - \hat{x}^{(i)})^T + L \sum_{z^{(i)}|x^{(i)}} L^T\right]$$

Factor analysis can be thought of as a problem where we are given just the labels for d different problems and we are assuming a linear modelling of some hidden covariates. Don't know the design matrix, don't know the parameters, we are estimating both of these things given just the labels.

Just knowing labels $\xrightarrow{\text{Find}}$ x and parameters (μ, Σ, L)

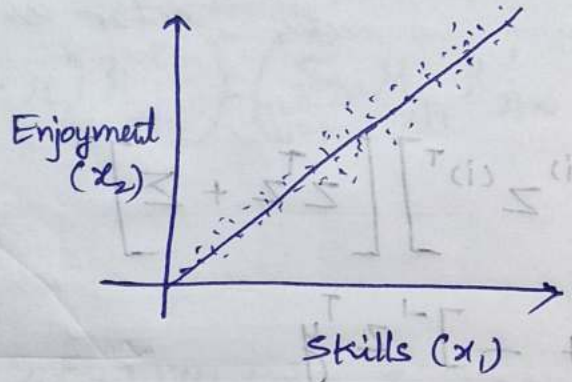
Principal Component Analysis (PCA)

$$\{x^{(i)}\}_{i=1}^{n} \quad x^{(i)} \in \mathbb{R}^d$$

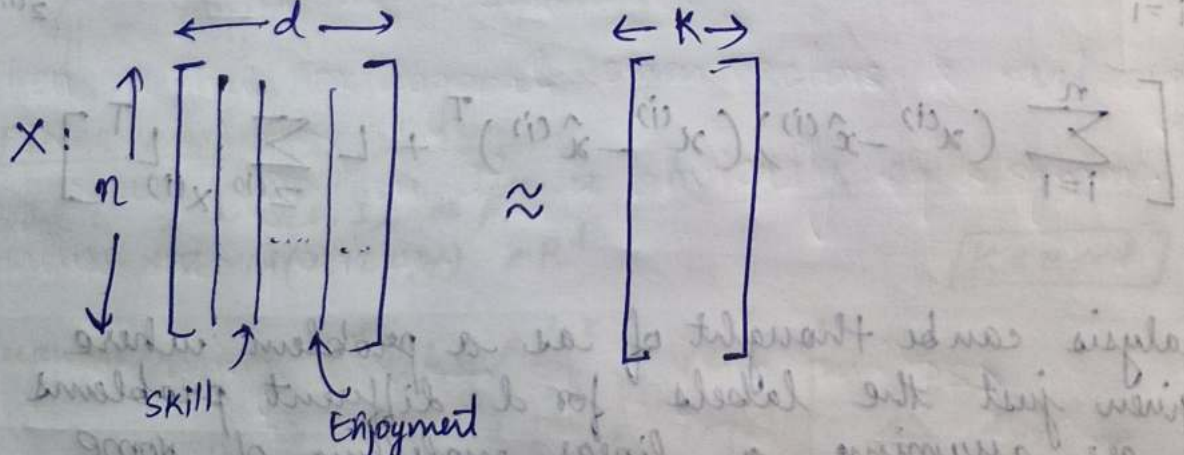
Here we are back to our general case where n is much larger than d. Much more number of data points than the dimension of the space.

Ex

Flying helicopter →



Basically from the given feature, relations can be co-related with each other, which will eventually help to reduce dimensions. ($d \to K$)

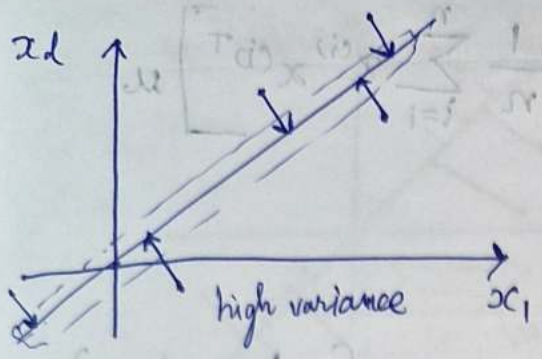


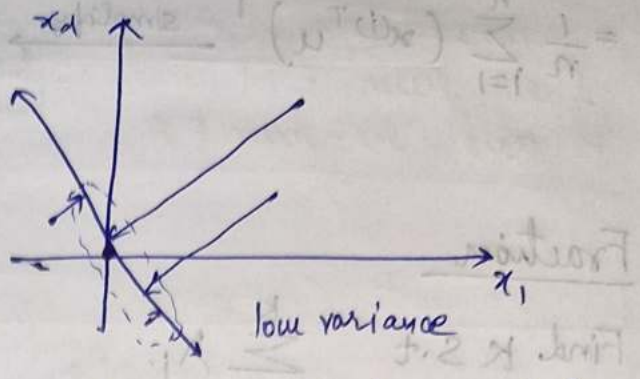
Step 1: Standardize

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

$$\mu_j = \frac{1}{n} \sum_{i=1}^{n} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{n} \sum_{i=1}^{n} (x_j^{(i)} - \mu_j)^2$$





PCA says that the subspace onto which we project our data should be the one where the variance of the projected points is maximized.

⇒ $u \in \mathbb{R}^d$ unit length (Here u is the basis vector)

⇒ $\text{Proj}(u) \cdot \vec{x} = \frac{uu^T}{u^T u} \cdot \vec{x} = (\vec{x^{(i)}} \cdot u)\,\vec{u}$

Our goal is to find the u, such that the variance across the projected points is maximum.

Basically find u such that $\frac{1}{n}\sum_{i=1}^{n} \|\text{Proj}(u)\, x\|^2$ is max.

$$u = \arg\max_u = \frac{1}{n}\sum_{i=1}^{n} \|x^{(i)T} u\|_2^2$$

$$= \frac{1}{n}\sum_{i=1}^{n} \left(x^{(i)T} u\right)^2$$

$$= u^T \underbrace{\left(\frac{1}{n}\sum_{i=1}^{n} x^{(i)} x^{(i)T}\right)}_{\text{Simple covariance Matrix}} u$$

$$u = \arg\max_u \; u^T \left[\frac{1}{n}\sum_{i=1}^{n} x^{(i)} x^{(i)T}\right] u$$

Using prev. knowledge $\arg\max_u u^T A u$

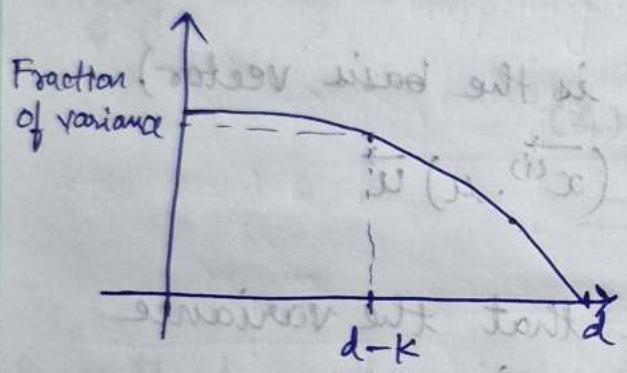
⇒ Eigen vector value of A. (Longer)

$$= \frac{1}{n} \sum_{i=1}^{n} \left(x^{(i)T} u\right)^T \xrightarrow{\text{simplify}} u^T \left[\frac{1}{n} \sum_{i=1}^{n} x^{(i)} x^{(i)T}\right] u$$

Fraction

Find k s.t $\dfrac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \geq 95\%$

{ $\lambda \rightarrow$ eigen value }



Value decomposition

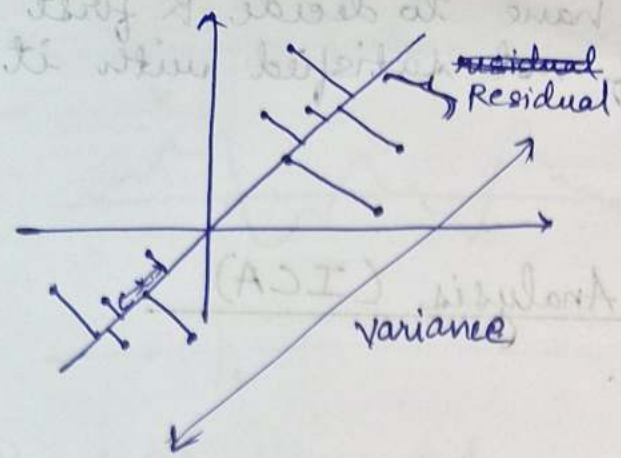
$X \rightarrow$ Square and symmetric.
then X has orthogonal eigen vectors & real eigen values

If X is PSD then it is obviously sq. & symm. Eigen values are positive.

We are using $X^T X \rightarrow$ sq, symm and PSD

⇒ **Eigen decomposition on $X^T X$ = SVD (singular value decomposition) on X**

SVD is applied for any matrix.



If we rephrase the problem as finding the subspace where the projected points are as close as possible to the original points i.e minimizing Residual.

Maximizing the variance and minimizing residual are equivalent

So far...

	Non Probabilistic	Probabilistic	
Clustering	K-means	GMM	Loosely speaking, it can be thought of as the counterpart of classification
Subspace	PCA	Factor Analysis	Regression

$U = [\,|\,|\,|\,]$ ($\leftarrow k \rightarrow$, $\updownarrow d$), $n > d$

XU^T = Projection X on lower dimension

$U : x \rightarrow z$

E·M

$d >> n$

z was subspace

$L : z \rightarrow x$

In PCA we first perform PCA to get the set of all eigenvalues & vectors and then decide what is the appropriate k for the problem.

Whereas in FA, we have to decide K first and then run EM and if not satisfied with it, change K and re-run FA.

Independent Component Analysis (ICA)

Lets consider an example.

$d \rightarrow$ speakers

$d \rightarrow$ microphones placed in the room

$s \in \mathbb{R}^d \rightarrow$ One instantaneous recording of the speeches happening.

$x \in \mathbb{R}^d \rightarrow$ Recordings in microphones

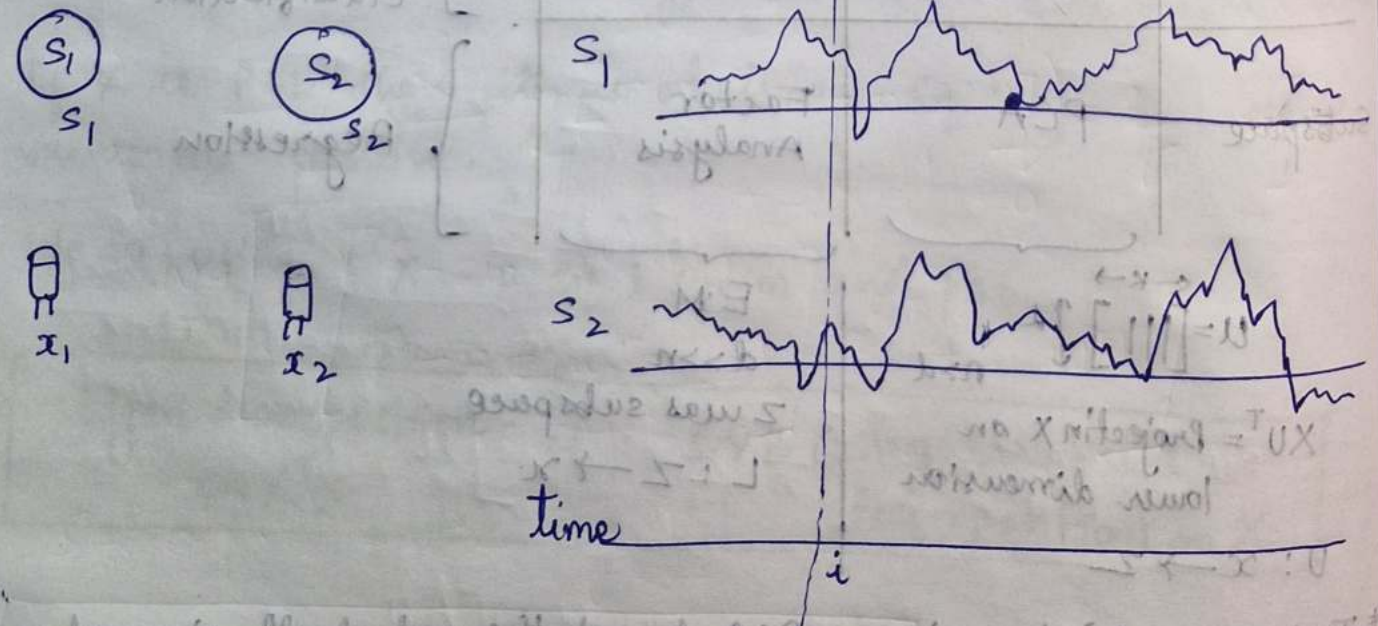
$\Rightarrow x = A \cdot s$

$A \rightarrow$ Mixing matrix

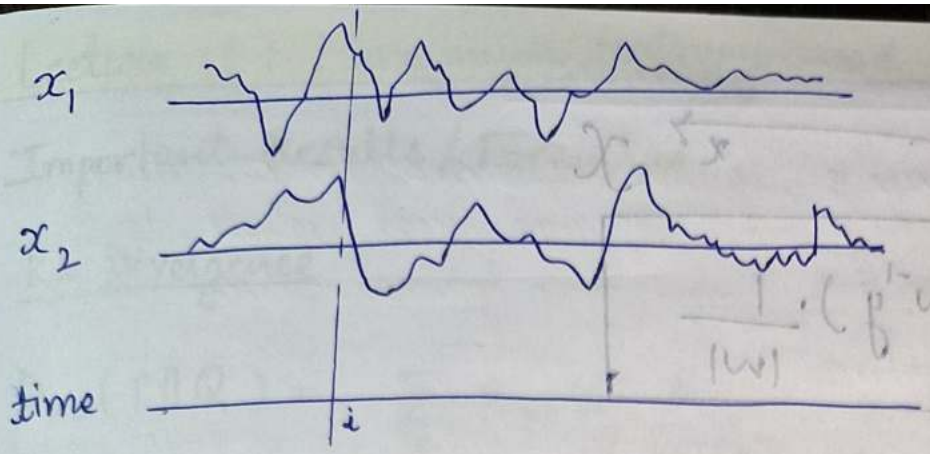
s_j^i: $i \rightarrow$ time, $j \rightarrow$ speaker identity

and

x_j^i: $i \rightarrow$ time, $j \rightarrow$ microphone



$$s^{(i)} = (s_1^{(i)}, s_2^{(i)}) \in \mathbb{R}^d \qquad d=2$$



$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}) \in \mathbb{R}^d \quad d=2$$

$$\Rightarrow x^{(i)} = As^{(i)}$$

$W = A^{-1}$ [unmixing matrix]

$$\Rightarrow \boxed{s^{(i)} = Wx^{(i)}} \rightarrow \text{extraction of source from mixed audio.}$$

Assumptions

No. of s = No. of x.

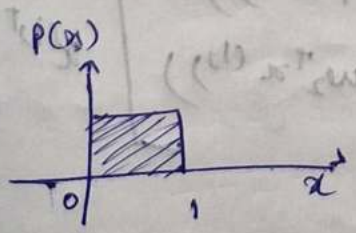
Linear combination : $s = Wx$

The sources are independent of each other : $s_j \perp s_k \; ; j \neq k$

Gaussian: s_j is NOT Gaussian (Due to the problem of rotational invariance. Therefore we use Laplace, logistic etc.)

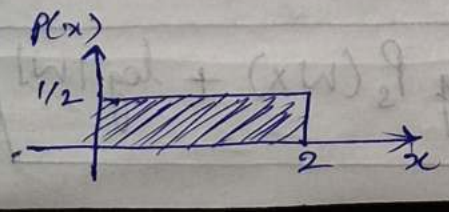
Proof

$x \sim \text{Unif}[0,1]$



$y = 2x$

$P_y(y) = P_x(y/2) \cdot \frac{1}{2}$



{Area under the PDF must be 1}

$y = Wx$

$\mathbb{R}^d \quad \mathbb{R}^{d \times d} \quad \mathbb{R}^d$

$x \rightarrow \mathcal{X}$

$$\boxed{P_y(y) = P_x(W^{-1}y) \cdot \frac{1}{|W|}}$$

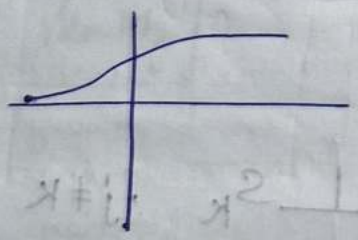
$$\Rightarrow P(x) = \prod_{j=1}^{d} P_s(w_j^T x) \cdot |W|$$

j^{th} source

$$[s_j] W = \begin{bmatrix} - w_j - \end{bmatrix} \begin{bmatrix} x \end{bmatrix}$$

unmixing matrix

$P_s \sim$ logistic distribution



CDF of $\log F(x) = \frac{1}{1+e^{-x}} = \sigma(x)$

PDF of $f(x) = \sigma(x) \cdot (1 - \sigma(x))$

$$\ell(W) = \sum_{i=1}^{n} \left[\left(\sum_{j=1}^{d} \log \left[\sigma(x^{(i)}) \cdot (1 - \sigma(x^{(i)})) \right] \right) + \log |W| \right]$$

$$W = W + \left[2 \begin{bmatrix} (1 - 2\sigma(w_1^T x^{(i)})) \\ (1 - 2\sigma(w_2^T x^{(i)})) \\ \vdots \end{bmatrix} x^{(i)T} + (W^T)^{-1} \right]$$

$S = Wx.$

$$\boxed{\hat{W} = \arg\max_W \log P_s(Wx) + \log |W|}$$

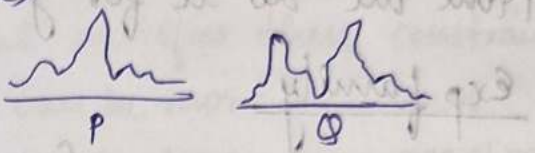
Lecture 19 : Maximum Entropy and Calibration

Important Results / Formulas.

KL Divergence.

$D_{KL}(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} = \mathbb{E}_P\left[\log \frac{P}{Q}\right]$

- $P, Q \rightarrow$ Probability distributions



- This is not symmetric. $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$

ENTROPY $H(P) = -\sum_x P(x) \log P(x) = \mathbb{E}_P\left[\log \frac{1}{P}\right]$

CROSS-ENTROPY $H(P,Q) = -\sum_x P(x) \log Q(x) = \mathbb{E}_P\left[\log \frac{1}{Q}\right]$

Relative Entropy $\Rightarrow D_{KL}(P\|Q) = H(P,Q) - H(P)$.
= KL Div.

Exponential Family

$$P(y;\eta) = b(y) \exp\{\eta \cdot T(y) - a(\eta)\}$$

$\Rightarrow$ Mean $(\mu) = a'(\eta)$

$\Rightarrow \eta = (a')^{-1}(\mu)$

MLE of Exponential families

Previously exp. family, y was assumed to be always a scalar like in regression it was real values, logistic regression → binary (0,1), saw some generalization with softmax (some values).

Now we do it for generalized case. ($y \rightarrow$ anything)

Exp family

$$P(y;\eta) = b(y)\exp\{\eta \cdot T(y) - a(\eta)\}$$

$$S = \{y^{(1)}, \ldots, y^{(n)}\} \quad \because \text{i.i.d assumption}$$

$$\hat{\eta} = \boxed{\arg\max_{\eta} \log \prod_{i=1}^{n} P(y^{(i)};\eta)}$$

$$= \arg\max_{\eta} \sum_{i=1}^{n} \left[\log b(y^{(i)}) + \eta \cdot T(y^{(i)}) - a(\eta)\right]$$

$g = a' =$ canonical resp fn

$g = (a')^{-1} =$ canonical link fn

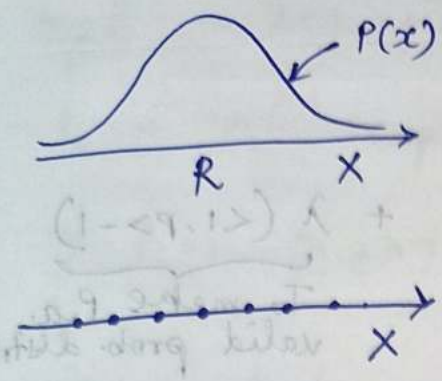
$$\nabla_{\eta} = \sum_{i=1}^{n} \left(0 + T(y^{(i)}) - a'(\eta)\right) = 0$$

$$\Rightarrow \underbrace{n}_{\text{Total no. of } n}\, a'(\eta) = \sum_{i=1}^{n} T(y^{(i)})$$

$$\Rightarrow a'(\eta) = \frac{1}{n}\sum_{i=1}^{n} T(y^{(i)})$$

General form

$$\boxed{\hat{\eta}_{MLE} = (a')^{-1}\left(\frac{1}{n}\sum_{i=1}^{n} T(y^{(i)})\right)}$$



Constraints

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$$

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} (x^{(i)} - \hat{\mu})^2$$

* Any probabability distribution that satisfies these constraints is a feasable distribution. It can be morethan one. Out of all these choose the one which has maximum entropy.

$$P^* = \arg\max_{P} H(P)$$

~~$s.t \quad \sum_i T_j(P_i) = C_j$~~ $\quad \sum_i T(y_i) P_i = C_j$ $\quad j = 1, \ldots, m$.

$T_j : Y \to R \quad , C_j$ for $j = 1, \ldots, m$

$\boxed{T(y) = y}$ constraint for mean

$$\sum_{i=1}^{N} T(y) \cdot P(y) = C \quad \#$$

$$= \sum_{i=1}^{N} y \cdot P(y) = \mathbb{E}[y]$$

Support Y is finite

N is the $|Y|$

M constraints (It means M such T functions)

$$c_j = \frac{1}{n} \sum_{i=1}^{n} T(y^{(i)})_j$$

$\mathcal{L}(p, \eta, \lambda) = \underbrace{H(p)}_{\text{Entropy}} + \underbrace{\langle \eta, T_p - c \rangle}_{\text{Constraints (From data)}} + \underbrace{\lambda (\langle 1, p \rangle - 1)}_{\text{To make } p \text{ a valid prob. distribution}}$

$$T_1 \cdots \begin{bmatrix} T_1(y_1) & T_1(y_2) & \cdots & T_1(y_N) \\ & & \vdots & \\ T_m(y_1) & T_m(y_2) & \cdots & T_m(y_N) \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{bmatrix} \downarrow N = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

$$p_i \geq 0, \quad \sum_{i=1}^{N} p_i = 1, \quad p_2 = P(y_2)$$

$$\begin{bmatrix} E_p[T_1(y)] \\ E_p[T_2(y)] \\ \vdots \\ E_p[T_n(y)] \end{bmatrix} = \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}$$

Solving Lagrangian ($\mathcal{L}$)

$$\frac{\partial}{\partial p_i} \mathcal{L} = \frac{\partial}{\partial p_i} \left[H(p) + \langle \eta, T_p - c \rangle + \lambda (\langle 1, p \rangle - 1) \right]$$

$$= \frac{\partial}{\partial p_i} \left[-p_i \log p_i + \langle \eta, T_p - c \rangle + \lambda (\langle 1, p \rangle - 1) \right]$$

$$= \left[-(\log p_i + 1) + \langle \eta, T(y_i) \rangle + \lambda \right] = 0$$

$$\Rightarrow \log p_i = \lambda - 1 + \langle \eta, T(y_i) \rangle$$

$$\Rightarrow p_i = e^{\lambda - 1} \cdot \exp\{\eta \cdot T(y_i)\}$$

$$\sum_{i=1}^{n} p_i = 1 \quad \sum e^{\lambda - 1} \cdot \exp\{\eta \cdot T(y_i)\} = 1$$

$$e^{\lambda - 1} = 1 / \sum \exp\{\eta \cdot T(y)\} \rightarrow z(\eta)$$

$$\Rightarrow \boxed{p(y) = \frac{\exp\{\eta \cdot T(y)\}}{z(\eta)}}$$

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

$$a = \log z$$

$$\Rightarrow \nabla_\eta \mathcal{L} = \nabla_\eta \left[-\sum_y p(y) \log p(y) + \eta \cdot T_p - \eta \cdot c + \lambda(\ ) \right]$$

$$\Rightarrow \nabla_\eta \left[-\sum_y p(y) \{\eta \cdot T(y) - a(\eta)\} + \eta \cdot E_p[T(y)] \cdot \eta c + \lambda(\ ) \right]$$

$$\Rightarrow \nabla_\eta \left[-\eta E[T(y)] + a(\eta) - \eta \cdot E_p[T(y)] - \eta \cdot c + \lambda(\ ) \right]$$

$$\Rightarrow \nabla_\eta [a(\eta) - \eta \cdot c + \lambda]$$

$$\Rightarrow a'(\eta) - c = 0$$

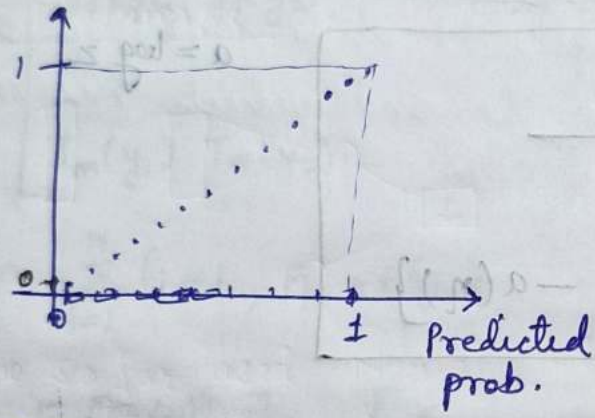
$$a'(\eta) = c$$

$$\boxed{\hat{\eta} = (a')^{-1}(c) = (a')^{-1}\left[\frac{1}{n}\sum_{i=1}^{n} T(y^{(i)})\right]}$$

$$\hat{y} = \frac{1}{1+\exp(-\theta^T x)} \in (0,1)$$

Calibration

It tells us about quality of $\hat{y}$ being an actual probability



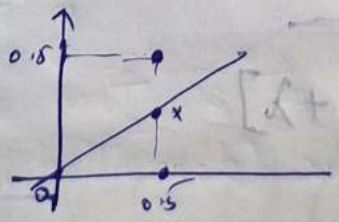
Estimated prob. values matching with observed frequency of occurences is called calibration.

predicted $\hat{y}^{(i)}$ $y^{(i)}$ → Ground truth

$(0,1)$ $\{0,1\}$

Calibration $\rightleftharpoons$ Accuracy (NO, both ways)

Eg $P(\text{correct}) = 0.51$
$P(\text{wrong}) = 0.01$



Entropy vs MLE

Max. Entropy ⇒ M.L.E

$\text{loss}_\theta = -\log P_\theta(x)$

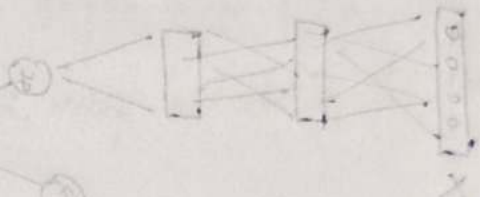
* Conventional way $\Rightarrow$ loss $(y, \hat{y})$

* Here Scoring Rule $[\hat{P}, y]$
  - $\hat{P}$: Prob. distr. over all possible outcomes.
  - y: The actual observed outcome.

Proper scoring rule, $f: (P, y) \rightarrow R$

$$E_{y \sim Q}[f(Q, y)] \leq E_{y \sim Q}[f(P, y)] \quad \forall P, Q$$

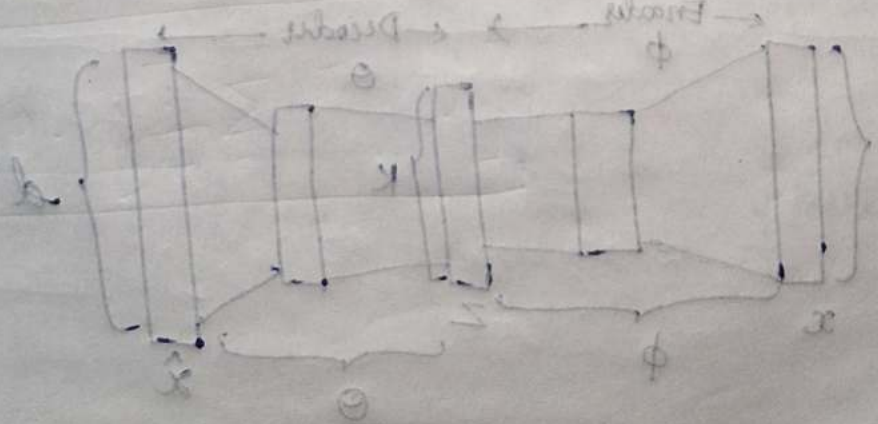
Strictly Proper scoring rule $\boxed{P = Q}$



$\Rightarrow$ Basically f will assign lower values for better predictions.

$$\boxed{\text{Loss} = E_{y \sim Q}\left[-\log \frac{1}{P}\right]}$$

$$E_{y \sim Q}[-\log Q(y)] \leq E_{y \sim Q}[-\log P(y)] = 0 \leq E_{y \sim 0}\left[\log \frac{Q(y)}{P(y)}\right]$$

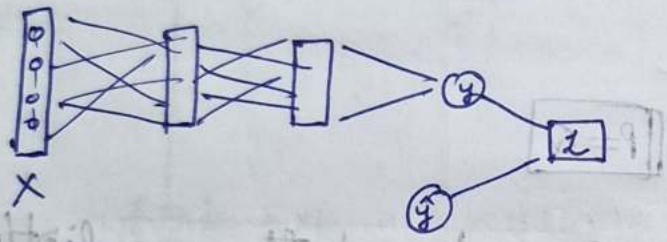


Lecture 20 - Variational Autoencoder

Variational autoencoders are the simplest Deep Generative models. VAE was an early method which made good progress in Deep gen. AI.

① Autoencoders

Neural Networks

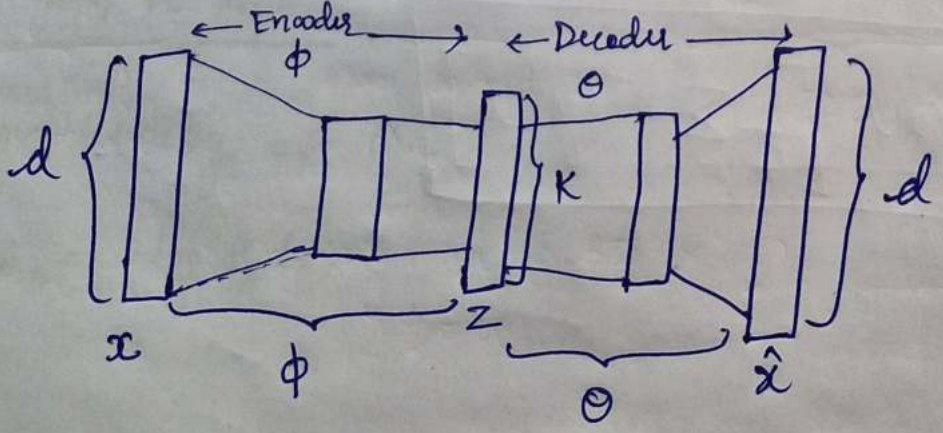


We have so far used Neural networks in supervised setting, where (X, Y) are given. And we use Backpropagation to update weights and biases.

Now for Autoencoders NN:

Now in unsupervised learning we are only given X. $S = \{x^{(1)}, \ldots, x^{(n)}\}$ and not y.

The goal of auto encoder is that is to learn a way in which we obtain a z (bottleneck) and reconstruct the original data.



* Loss $= \|\hat{x} - x\|_2^2$

$$\underset{(\theta,\phi)}{\text{Loss}} = \sum_{i=1}^{n} \Big\| x^{(i)} - \underbrace{\underset{\theta}{\text{Dec.}}\Big[\underbrace{\text{Enc}[x^{(i)}]}_{z^{(i)}}\Big]}_{\hat{x}^{(i)}} \Big\|_2^2$$

⇒ Here we take the data transformation from a bottleneck z ($d \to K$). It helps the model to learn some low dimensional representation of high dimensional input data (d). The (K) starting from the low.dim data we map it back to high dim. data itself.

⇒ Then minimize the loss ($\|\hat{x}-x\|_2^2$). Basically we have learned compression into a latent state / hidden state.

⇒ Weight & biases of Encoder are (ϕ) and of Decoder are (θ). These weight & biases (model parameters) are updated similarly that of backpropagation using Loss calculated.

Extra

⇒ There is also a Denoising-autoencoder where we learn how to denoise the input data. We provide the noisy input ($\tilde{x}$) and recover original (x) using this approach.

② MCMC : EM

E Step : For all i

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

M Step :

$$\theta^{(t+1)} = \arg\max_{\theta} \sum_{i=1}^{n} \sum_{z} Q_i^{(t)}(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i^{(t)}(z)}$$

$$= \arg\max_{\theta} \sum_{i=1}^{n} E_{z^{(i)} \sim Q_i^{(t)}} \left[\log P(x^{(i)}, z^{(i)}; \theta) \right]$$

$$\approx \arg\max_{\theta} \sum_{i=1}^{n} \frac{1}{M} \sum_{m=1}^{M} \log P(x^{(i)}, z_i^{(m)}; \theta)$$ ← Monte carlo estimate

$z_i^{(m)} \sim Q_i^{(t)}$ (sampled)

In original EM we constructed the convergence proof where with every E,M step the likelihood was only increasing. But here the guarantee does not hold any more because this is an approximation of lower bound and not the exact lower bound.

Variational Inference

$$\log P(x) \geq ELBO(x; Q)$$

$$\log P(x) = ELBO(x; Q) + \text{?}$$

$$D_{KL}(Q \| P_{z|x}) = \log P(x) - ELBO(x; Q)$$

$$\underbrace{\log P(x)}_{\text{Constant w.r.t } Q} = \underbrace{ELBO(x; Q)}_{\text{Max w.r.t } Q} + \underbrace{D_{KL}(Q \| P_{z|x})}_{\geq 0}$$

$$P_{z|x} \approx \arg\max_{q \in Q} ELBO(x; q)$$ ← Variational Inference

This approach where we maximize the lower bound (ELBO) w.r.t to Q and obtain a distribution which we approximate to z/x is variational inference

In the Monte carlo technique, we do sampling while in variational inference we do optimization.

In Monte carlo we never know how good our estimate is all we know is that we eventually reach the exact solution. while in variation inference will converge and we know when to stop (The soln. is approximate).

Q. In variational inference most of the times the question is how do we choose the family Q from which we want to perform optimization.

$$Q(z) = Q_1(z_1) \cdot Q_2(z_2) \cdot Q_3(z_3) \cdots Q_K(z_K)$$

$$z \in \mathbb{R}^K$$

This assumption is - Mean field assumption. VI

Variational Autoencoders

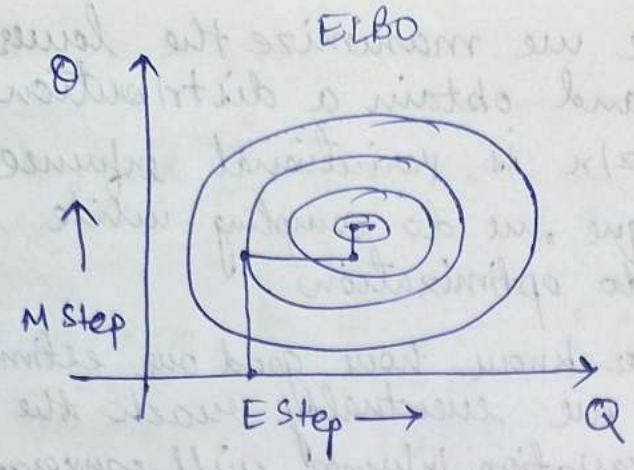
ELBO (Q, θ)

In EM we construct this ELBO, in E Step we find the best possible Q (posterior) & in the M Step we would update θ and while performing these steps we keep the other constant.

E Step : Calculate Q, θ fixed.

M Step : Calculate θ, Q fixed

This technique is also called Coordinate Ascent



Coordinate Ascent

Can we do gradient ascent?
In VAE we max. ELBO using gradient ascent.

Prior → $Z \sim N(0, I_{K \times K})$

Likelihood → $X|Z \sim N(g(z;\theta), \sigma^2 I)$

g is a neural network with param θ.

Posterior $p(z|x)$

Q family for V.I

$Q_i(z) = N(\underbrace{q(x^{(i)};\phi)}_{R^K}, \text{diag}\underbrace{(V(x^{(i)};\psi))^2}_{R^{K \times K}})$

↳ Amortized Inference

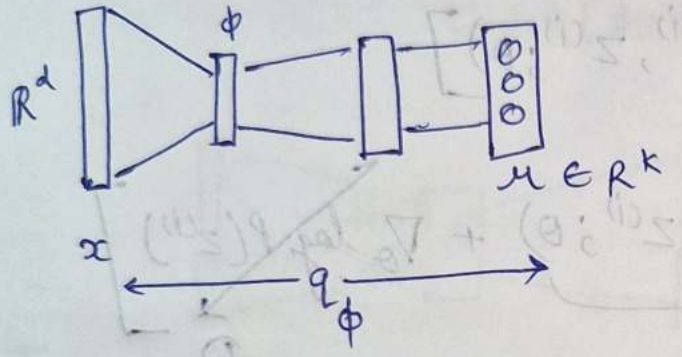
The Difference b/w EM and VAE is that in EM we were separately calculating the parameters of the Q distribution for each example independently.

In Amortized Inference, we do not compute separately but we assume that all Q comes from Normal distribution N and the mean and variance of each Q_i is a function

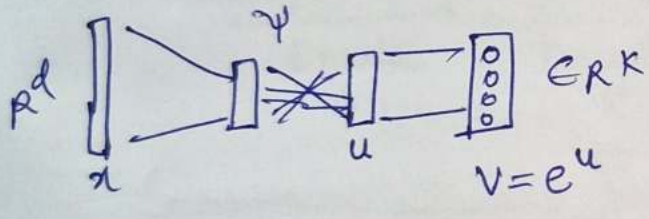
of the x's. These functions are Neural Networks.

* Feed x as the input and the NN will output two scalars per input i.e Mean and Variance.

Note: This is not autoencoder (No bottleneck).



$q_\phi(x^{(i)}) = \mu^{(i)}$
Mean



$k \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}$

We are making mean field assumption.

Feed x in input ⟶ Learned parameter ϕ, ψ ⟶ Estimate Q from ϕ, ψ. (E Step construction)

Basically

$$ELBO(\phi, \psi, \theta) = \sum_{i=1}^{n} E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

Where $Q_i = N\left(q(x^{(i)}; \phi), \text{diag}(v(x^{(i)}; \psi))\right)$

$$\theta = \theta + \eta \nabla_\theta ELBO(\phi, \psi, \theta)$$
$$\phi = \phi + \eta \nabla_\phi (\text{"} \quad)$$
$$\psi = \psi + \eta \nabla_\psi (\text{"} \quad)$$

⟶ Gradient updates until Convergence.

For θ

$$\nabla_\theta \text{ELBO}(\phi, \psi, \theta)$$

{ θ independent of Q }

$$\Rightarrow \nabla_\theta \sum_{i=1}^{n} E_{z^{(i)} \sim Q} \left[\log \frac{P(x, z; \theta)}{Q_i(z^{(i)})} \right]$$

$$\Rightarrow \sum_{i=1}^{n} E_{z^{(i)} \sim Q_i} \left[\nabla_\theta \log P(x^{(i)}, z^{(i)}; \theta) \right]$$

$$\Rightarrow \sum_{i=1}^{n} E_{z^{(i)} \sim Q_i} \left[\nabla_\theta \underbrace{\log P(x^{(i)} | z^{(i)}; \theta)}_{\text{~~Encoder~~ Decoder}} + \nabla_\theta \log P(z^{(i)}) \right]$$

θ

For ϕ

$$\nabla_\phi \text{ELBO}(\phi, \psi, \theta)$$

$$\Rightarrow \nabla_\phi \sum_{i=1}^{n} E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x, z; \theta)}{Q(z^{(i)})} \right]$$

{ ϕ is parameter of Q }

$$z \sim N(\mu, \sigma)$$

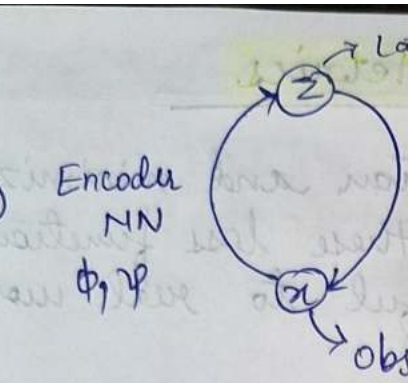
$$z = \epsilon \cdot \sigma + \mu \; ; \; \epsilon \sim N(0, 1)$$

$$\nabla_\phi \sum_{i=1}^{n} E_{\epsilon^{(i)} \sim N(0,1)} \left[\log \frac{P(x, \epsilon^{(i)} \cdot \Sigma^{(i)} + \mu^{(i)}; \theta)}{Q(\epsilon^{(i)} \cdot \Sigma^{(i)} + \mu^{(i)})} \right]$$

where $\mu^{(i)} = q(x^{(i)}; \phi)$

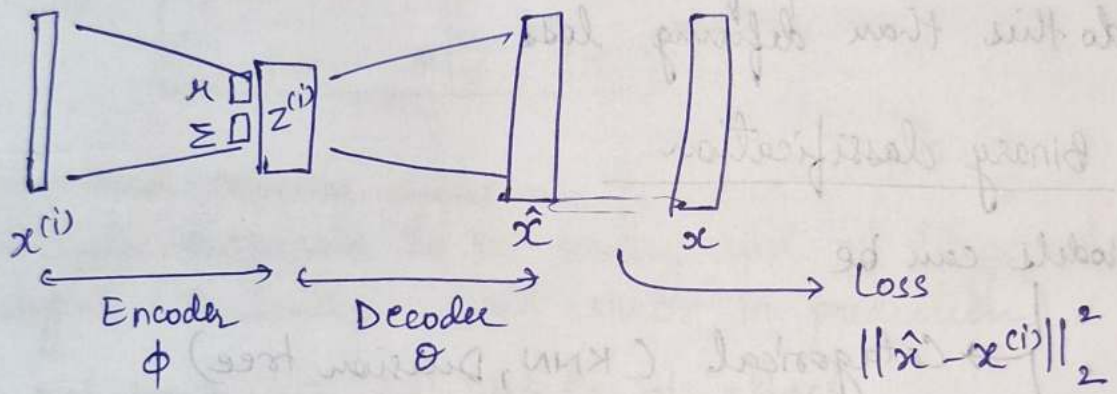
$\Sigma^{(i)} = \text{diag}(V(x^{(i)}; \psi))$

Now we can take gradient inside



ELBO (ϕ, Ψ, θ)

#



Lecture 21 : Evaluation Metrics

We often calculate a loss function and minimize it w.r.t the parameters. However these loss function is not clear if they are meaningful to real world case. (Eg. Buisness goal).

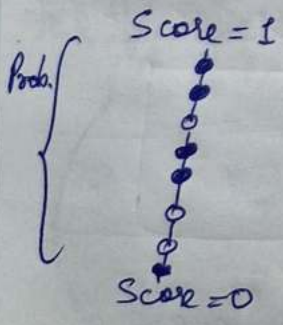
Evaluation metrics help to quantify the gap between the desired performance and baseline. It is easier to do this than defining loss.

Binary classification

Models can be

- Catagorical (KNN, Decision tree)
- Real value score (SVM, Logistic regression)

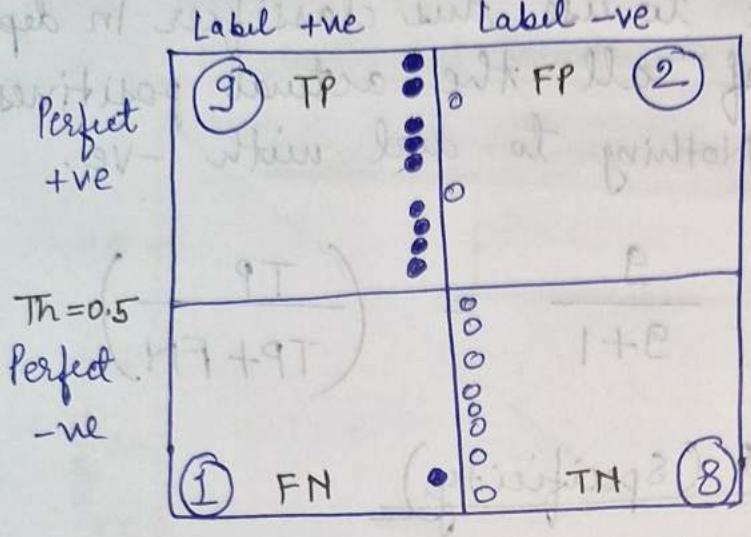
Score based models



● +ve labelled example
○ -ve labelled example

Prevalence = $\frac{\#\ +ve}{(\#\ +ve) + (\#\ -ve)}$

Point Metrics : Confusion Matrix



Threshold = 0.5

* Total sum and column sums are fixed.
* We want the diagonals to be heavy and off diagonals to be light (Basically least error in prediction).
* FP and FN have very diff. kinds of effect

Th	TP	TN	FP	FN	Acc.	Pr	Recall	Spec	F1
0.5	9	8	2	1	0.85	0.81	0.90	08	0.85

* **Accuracy** $= \dfrac{\text{Predicted correct}}{\text{Total}} = \dfrac{(9+8)}{(9+8+2+1)} = \dfrac{(TP+TN)}{(TP+TN+FP+FN)}$

  → Fraction of example we got right.

* **Precision** = PPV (Positive predictive value)

  → Fraction of predicted examples actually positive

$$\Rightarrow \left(\frac{TP}{TP+FP}\right) = \frac{(9)}{(9+2)}$$

Recall (Sensitivity) or Positive recall:

Imagine if we were to use this classifier in deployment, what fraction of all the actual positives are we gonna recover. Nothing to deal with -ve.

$$\text{Recall (Sensitivity)} = \frac{9}{9+1} = \left(\frac{TP}{TP+FN}\right)$$

Negative Recall (Specificity)

What fraction of the actual examples (negatives only) did the model correctly classify as negative.

$$\text{Neg. Recall (Specificity)} = \frac{TN}{TN+FP} = \frac{8}{8+2}$$

F1 score

F1 score is the harmonic mean of precision and recall

$$\boxed{\frac{1}{F_1} = \frac{1}{P_r} + \frac{1}{R_c}} \Leftrightarrow \boxed{F_1 = \frac{P_r \cdot R_c}{P_R + R_c}}$$

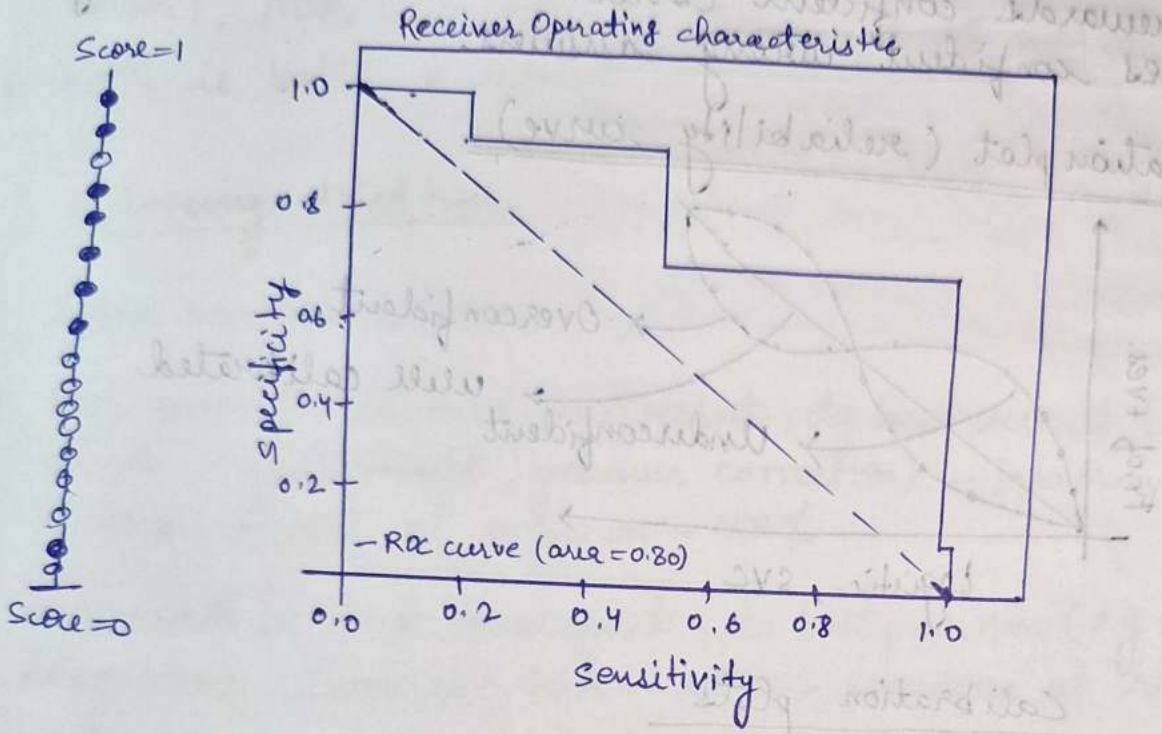
G score

Geometric mean

$$\boxed{G_1 = \sqrt{P_r \cdot R_c}} \Leftrightarrow \boxed{2\log G_1 = \log P_r + \log R_c}$$

As we change the threshold we get different values of all these param. Total thresholds = (1+ Total examples).

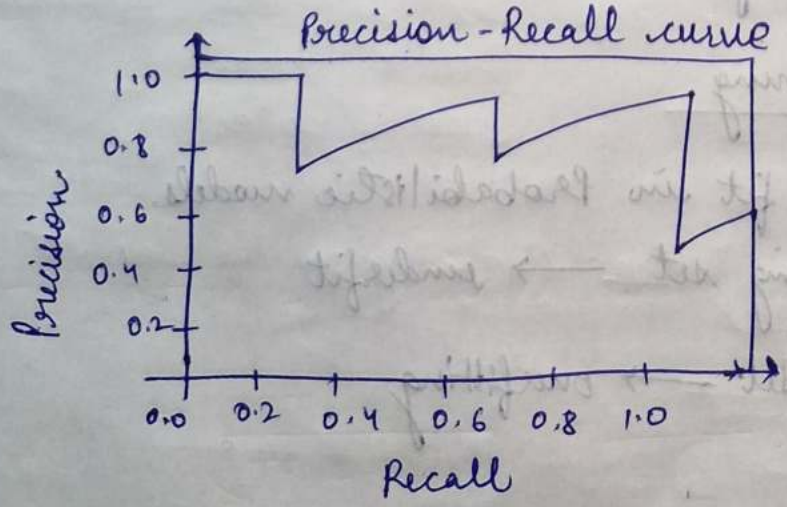
ROC curve



Area under the ROC curve will be high if most of the +ve examples are the ones we encounter first as we start scaning down.

Another intuition is that what is the probability that a random +ve example is ranked higher than a random -ve example. This probability is exactly equal to the area under the ROC curve.

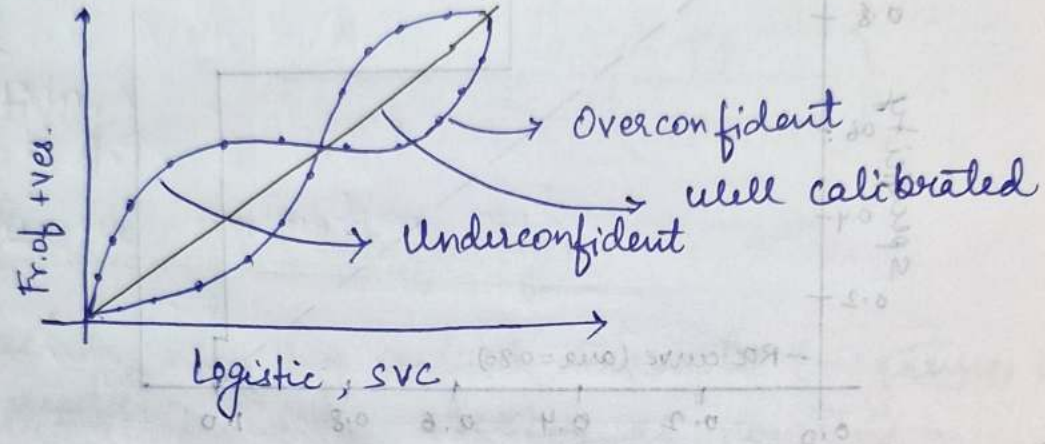
PRC curve



Log loss

Log loss rewards confident correct answers and heavily penalises confident wrong answers.

Calibration plot (reliability curve)



Calibration plots

Loss

$$\text{Brier} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{p}^{(i)} - y^{(i)} \right)^2$$

$\hat{p}$ → Predicted probability

Mean square error

Calibration : How meaningful the assigned probabilities are
Discrimination : How well the model can separate +ve & -ve in a ranking sense.

Unsupervised Learning.

$\log P(x)$ is a measure of fit in Probabilistic models

High $\log P(x)$ on training set ⟶ underfit

· low $\log P(x)$ on test set ⟶ overfitting

In a class imbalance scenario our least preference should be accuracy (it doesn't tell any reasonable value), $AUROC$ can also be fooled / wrong intuition.

$AUPRC$ is better & robust

Choosing Metrics

Some common Patterns

High precision is hard constraint, do best recall (e.g search engine result, grammer correction) -- intolerant to FP. Metric: Recall at precision = XX%

High recall is hard constraint, do best precision (e.g medical diagnosis). Intolerant to FN. Metric Precision at recall = 100%

Capacity constrained (by K). Metric: Precision in top K.

Lecture 22 - Course Recap and Practical Tips

ML in Practice vs Research

1. Start collecting a Dev set
2. Dev set Distribution match the production scenario.
3. Collect data & label
4. Define an evaluation matric
5. Training data
   - → As close as possible to Dev set
   - → Automated / noisy labelling

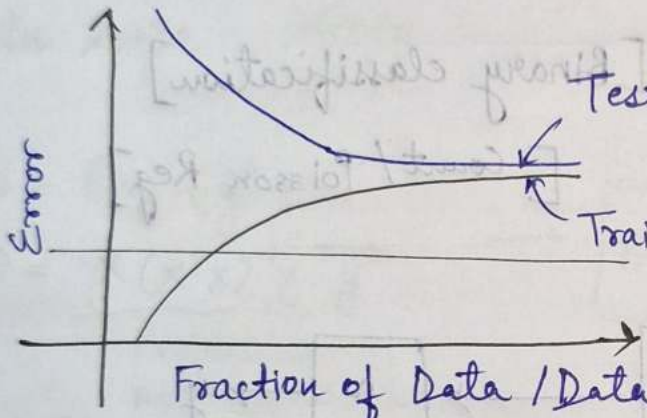
Breakdown of Performance

- **Human Level Performance** – Proxy for irreducible error
- **Training Performance** — Gap: Proxy to Bias
- **Train - Dev Performance** — Gap: Variance of the Model
- **Dev set Performance** — Gap: Dist^n mismatch between Dev & Train [cost of collecting]
  - Manually labelled
- **Deployment Performance** — Gap: Overfitting on the Dev set

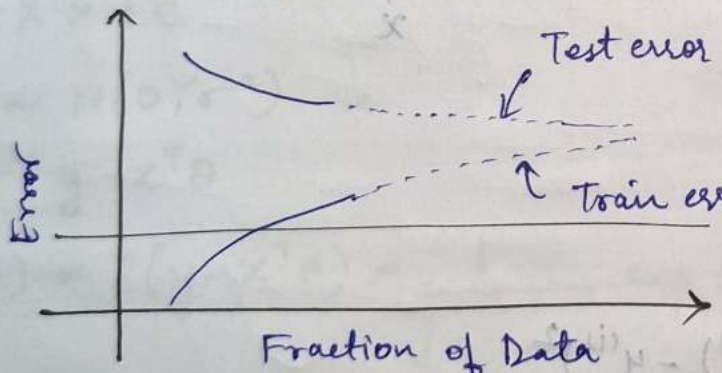
If the model has high bias, then these steps will help to fight bias

- add feature
- add depth
- Reduce regularization
- Complex kernels

→ Do the opposite of all these to fight variance
⊕ Get more data



- High Bias Scenario
- Collecting more data won't help.



- Yes more data will help to reduce (Test - Train) error

Our always focus should be on deployment performance

SUPERVISED LEARNING

$X \rightarrow Y$

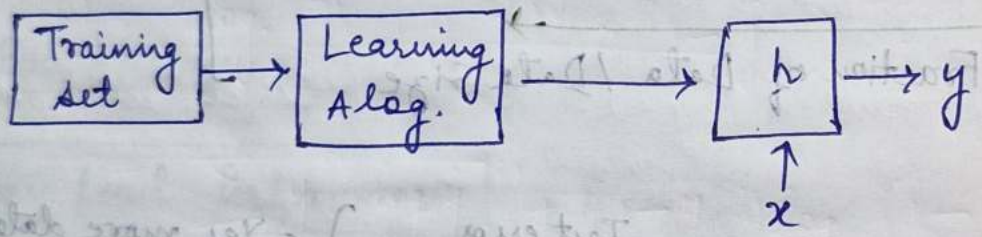
$S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$

input $x^{(i)} \in \mathbb{R}^d$

output $y^{(i)} \in \mathbb{R}$ [Regression]

$y^{(i)} \in \{0, 1\}$ [Binary classification]

$y^{(i)} \in 1, 2, 3, \ldots$ [Count / Poisson Reg]



① linear regression

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

$\theta \in \mathbb{R}^{d+1}$

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

$$= \| X\theta - \vec{y} \|_2^2$$

$$\hat{\theta} = \arg\min_{\theta} J(\theta)$$

② Gradient Descent

Repeat

$$\theta = \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - h_\theta(x^{(i)})) \cdot x^{(i)}$$

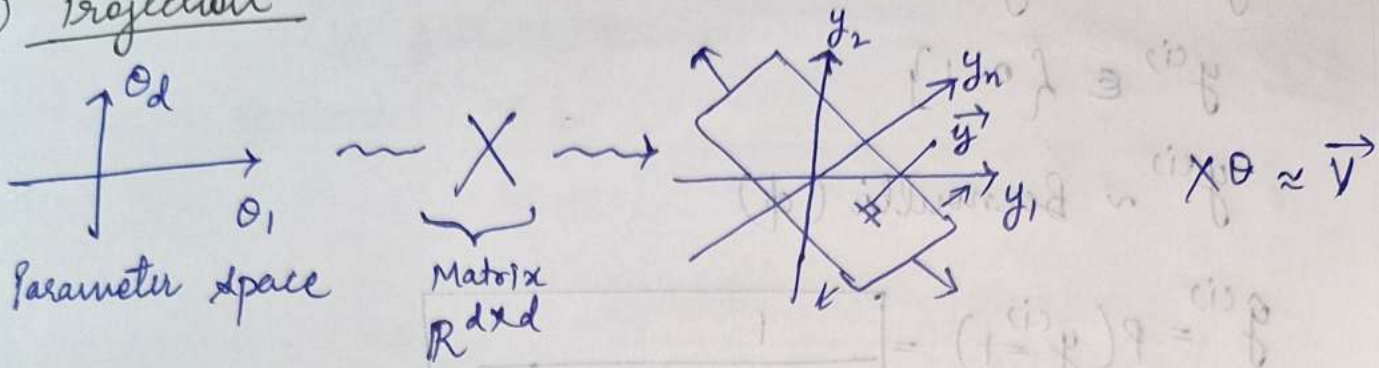
SGD: Repeat $\theta = \theta + \alpha (y^{(k)} - h_\theta(x^{(k)})) \cdot x^{(k)}$

$k \sim \text{Unif}[1, n]$

③ Normal Equation

$$X^T X\theta = X^T y \Rightarrow \boxed{\hat{\theta} = (X^T X)^{-1} X^T y}$$

④ Projection



$$X\theta = \text{Proj}(\vec{y}; X)$$

$$X\theta = \underbrace{X(X^T X)^{-1} X^T}_{\text{Projection Matrix}} \vec{y} \longrightarrow \boxed{\theta = (X^T X)^{-1} X^T \vec{y}}$$

⑤ Probabilistic Interpretation

$$y = X^T\theta + \epsilon$$

$$\epsilon \sim N(0, \sigma^2)$$

$$\epsilon = y - x^T\theta$$

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

Now we do MLE which is to basically find the parameter θ so that it maximizes this likelihood term.

$$L(\theta) = \log \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ \frac{-(y^{(i)} - x^{(i)T}\theta)^2}{2\sigma^2} \right\}$$

$$= \sum_{i=1}^{n} C + \left\{ \frac{-(y^{(i)} - x^{(i)T}\theta)^2}{2\sigma^2} \right\}$$

$$= C' + \frac{1}{2\sigma^2} \left(\sum_{i=1}^{n} (y^{(i)} - x^{(i)T}\theta)^2 \right\}$$

$$\arg\max_\theta \ell(\theta) = \arg\min_\theta \sum_{i=1}^{n} \left(y^{(i)} - x^{(i)T}\theta\right)^2$$

Logistic Regression

$y^{(i)} \in \{0, 1\}$

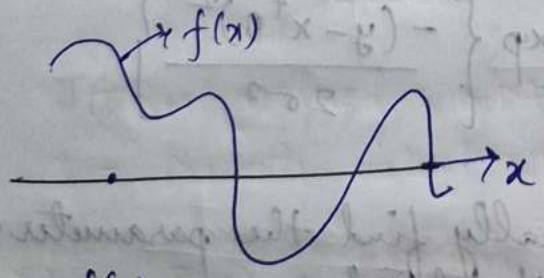
$y^{(i)} \sim \text{Bernoulli}(\phi)$

$$\hat{y}^{(i)} = P(y^{(i)} = 1) = \boxed{\frac{1}{1 + \exp\{-\theta^T x^{(i)}\}}}$$

$$\ell(\theta) = \log \prod_{i=1}^{n} (\hat{y}^{(i)})^{y^{(i)}} \cdot (1 - \hat{y}^{(i)})^{(1-y^{(i)})}$$

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

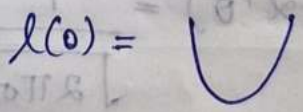
Newton's Method (scalar)

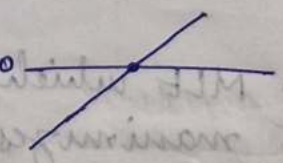


$f(x) = 0$

$f = \ell'$

$$\theta = \theta + \frac{1}{\ell''(\theta)} \ell'(\theta)$$

$\ell(\theta) =$ 

$\ell'(\theta) =$ 

$\ell(\theta) = \cap \Leftrightarrow \ell'(\theta)$ ~~$\Leftrightarrow$~~

Newton Raphson (Vector)

$$\boxed{\theta = \theta + H^{-1}(\theta)\, \ell(\theta)}$$

where $H(\theta) = \nabla_\theta^2 \ell(\theta)$

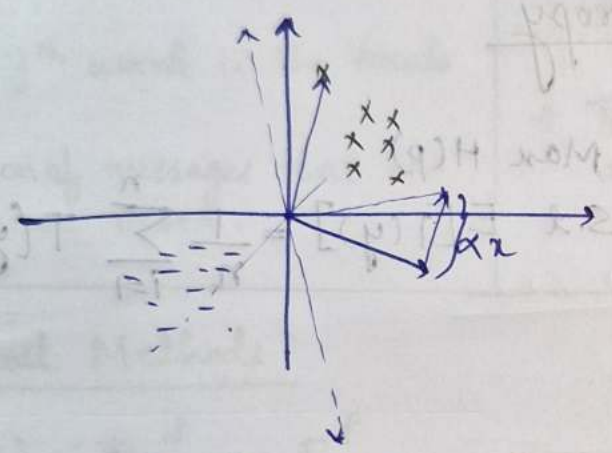
Lecture 23- Course Recap & Wrap Up

Perceptron algorithm

Encounter one example at a time

$$\theta = \theta + \alpha [y - g(\theta^T x)] \cdot x$$

$$y(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$



$$(\theta + \alpha x)^T x \geq \theta^T x$$

***

If a separating hyperplane does exist, then no matter in what order you present the examples to the learning algo. It will eventually find some separating hyperplane.

Exp Family

$$P(y;\eta) = b(y) \exp\{\eta^T T(y) - a(\eta)\}$$

y - variable

$T(y)$ - Sufficient statistics $[T(y) = y]$

η - Natural Parameter

$b(y)$ -- Base measure

$a(\eta)$ - log partition function

GLM

$\eta = \underbrace{\theta^T x}_{\text{learnable parameter}}$

Properties

$E[T(y)] = E[y] = a'(\eta)$
$Var[y] = a''(\eta)$

} Exp fam.

GLM

$E[y|x;\theta] \cdot x = a'(\theta^T x)$

$a''(\eta) = \underbrace{Var[y|x;\theta]}_{\geq 0} \cdot \underbrace{xx^T}_{PSD}$

Exp family & Max Entropy

$\{y^{(i)}\}_{i=1}^{n}$ MLE $\Leftrightarrow$ Max $H(p)$ s.t $E[T(y)] = \frac{1}{n}\sum_{i=1}^{n} T(y^{(i)})$

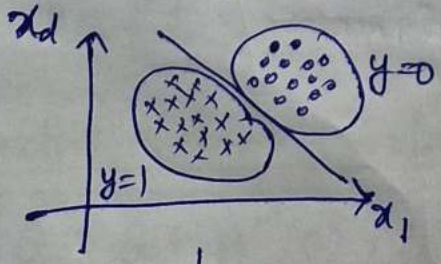
Generative Models

Discriminative – $P(y|x)$

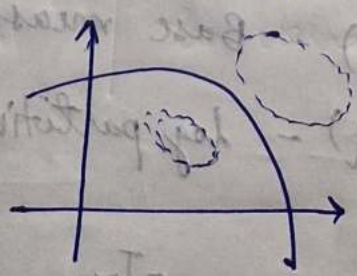
Generative $\underbrace{P(y,x)}_{\text{Joint}} = \underbrace{P(x|y)}_{\text{Likelihood}} \cdot \underbrace{P(y)}_{\substack{\text{Prior} \\ \text{[Class Prior]}}}$

x – Real valued GDA
x – Discrete valued NB

GDA – $P(y) \sim$ Bernoulli (ϕ).
$P(x|y) \sim N(\mu_y, \Sigma)$ $N(\mu_y, \Sigma_y)$



$P(y|x) = \frac{1}{1+\exp\{\dots\}}$



Naive Bayes

$x_i \perp x_j \mid y \neq x_i \perp x_j$

Bernoulli Event Model

$P(y) = \phi_y$ class Prior

$P(x_j \mid y) = \phi_{j|y}$

Bernoulli Dist^n

$x_j \rightarrow j^{th}$ word in the Vocab

Ex spam

Fraction of messages that has the spam word.

Multinomial Event Model

$P(y) = \phi_y$ class Prior

$P(x_j \mid y) = \phi_{j|y}$ Multinomial Dist^n

$x_j \rightarrow j^{th}$ word in a message

Total no. of times the word appear in the set.

Kernel Methods

$\phi(x) : \mathbb{R}^d \longrightarrow \mathbb{R}^p$

$x \in \mathbb{R}^d$

p – Dim of feature space p can be ∞

$K(x, x') = \phi(x)^T \phi(x')$

Properties

K is symmetric

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

$$\begin{bmatrix} K(x^{(1)}, x^{(1)}) & \cdots \\ \vdots & \ddots \\ \cdots & K(x^{(m)}, x^{(m)}) \end{bmatrix} \rightarrow \text{PSD}$$

(m × m)

For func. K to be a kernel it is necessary & sufficient for the matrix to be symm and PSD

Mercer's Theorem

This method allows us to have ∞ dimensional feature maps.

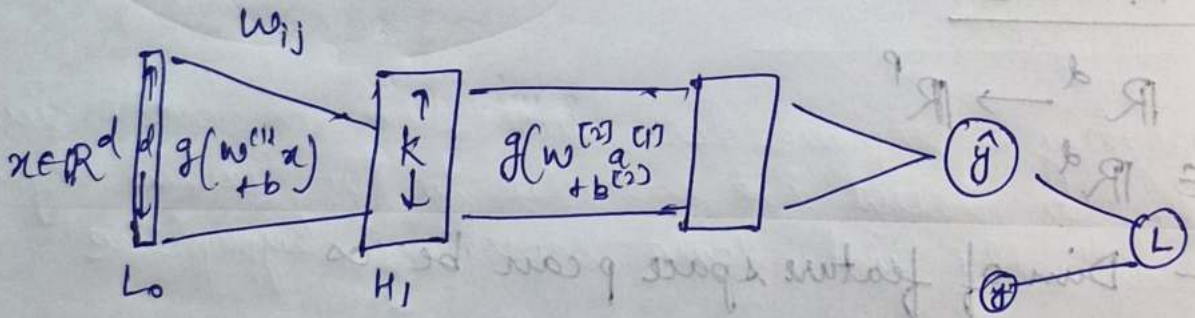
Gaussian Processes

Kernel method for regression.

$$\begin{bmatrix} | \\ | \end{bmatrix} \sim N\left[(\mu), (\Sigma)\right]$$

$$\updownarrow \sim GP\left(\updownarrow, k\right)$$

Fully connected Networks



$W \in \mathbb{R}^{k \times d}$

$W_x \in \mathbb{R}^k$

MSE (test) = Irreducible error + Bias^2 + Variance

Irreducible error: Noise in Test example

Bias^2: Inflexibility or Low Capacity of Model class

Variance: Training data noise contributes to variance.

$$loss = \left(\sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})^2 \right) + \lambda \|\theta\|_2^2 \right. \quad \text{- MAP w/ Gaussian}$$

$$\left. + \lambda \|\theta\|_1 \right. \quad \text{- MAP w/ Laplace}$$

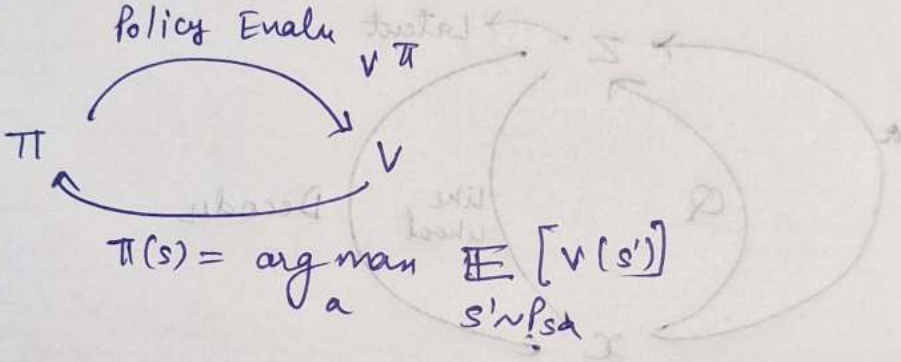
RL

MDP - $(S, A, \{P_{sa}\}, \gamma, R)$

Value : $V^\pi(s)$

Policy : $\pi(s) \rightarrow A$

Policy Iteration



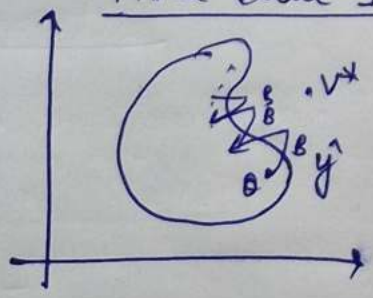
$$\pi(s) = \arg\max_a \mathbb{E}_{s' \sim P_{sa}} [V(s')]$$

Bellman Equation

Optimal $V^*(s) = \max_\pi V^\pi(s)$

$$V(s) = R(s) + \gamma \max_a \mathbb{E}_{s' \sim P_{sa}} V(s')$$

Fitted value Iteration



EM

Model $P(x, z; \theta)$

Parameter θ

Evidence x

Latent variable z

E-Step

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

M Step

$$\theta = \underset{\theta}{\text{argmax}} \; \underset{z \sim Q_i}{E} \left[\log \frac{P(x, z; \theta)}{Q_i(z)} \right]$$

ELBO

VAE