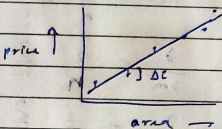
ML

Date
Page

Linear Regression :

$y = mx + c$

price = $m \times$ area $+ c$



Model finds m and c

by minimizing $\sum_{i=1}^{n} (\delta_i)^2$ (best fit line)

Given a house with some area, predict the price.

Multivariate linear regression : (multiple variables)

$y = m_1x_1 + m_2x_2 + m_3x_3 + b$

price = $m_1 \times$ area $+ m_2 \times$ bedrooms $+ m_3 \times$ age $+ b$

dependent var

independent variables (features)

m_1, m_2, m_3 → coefficients

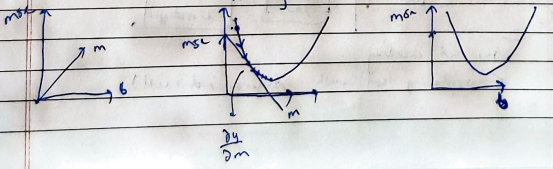
b → intercept

Mean squared error :

$$mse = \frac{1}{n} \sum_{i=1}^{n} (y_i - \underbrace{(mx_i + b)}_{y_{predicted}})^2$$

cost function

Gradient descent : to find best fit line for given training data set.



$\frac{\partial y}{\partial m}$

$$mse = \frac{1}{n} \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

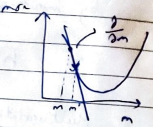
$$\frac{\partial}{\partial m} = \frac{2}{n} \sum_{i=1}^{n} -x_i (y_i - (mx_i + b))$$

$$\frac{\partial}{\partial b} = \frac{2}{n} \sum_{i=1}^{n} -(y_i - (mx_i + b))$$

steps

$$m' = m - \text{learning rate} \times \frac{\partial}{\partial m}$$

$$b' = b - \text{learning rate} \times \frac{\partial}{\partial b}$$

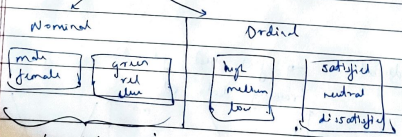


For example, start with $m = 0$, $b = 0$, learning rate = 0.001
Iterations = 1000
After all iterations we get the desired m and b values (best fit line)

→ After training, save the model using pickel or joblib. (No need to train everytime)

* One hot Encoding :

→ Categorical variables :



One hot encoding
(replacing text with integers not ideal)

eg:

town	area	price	A	B	C
A	—	—	1	0	0
A	—	—	1	0	0
B	—	—	0	1	0
B	—	—	0	1	0
B	—	—	0	1	0
C	—	—	0	0	1

↳ Now drop this column

→ Then drop one of the dummy variable column (either A, B or C)

eg: if we drop column C,

X = area, A, B

Y = price

so the dropped column can be derived from the other columns

→ train_test_split : eg: training data : 80%. testing data : 20%.

X_train, X_test, y_train, y_test = train_test_split (X, y, test_size=0.2)

(randomly selects rows)

Logistic Regression :

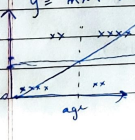
→ Predicted value is categorical. eg: Yes or No

→ Classification problem

Binary classification	Multiclass classification
eg: Yes, No	eg: Democratic, republican, independent

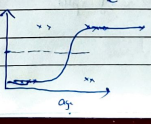
$y = sig(z) = \frac{1}{1+e^{-z}}$ $\quad y \in [0, 1]$

$y = mx + b$



1 → buy insurance.
0 → not buy insurance.

$y = \frac{1}{1 + e^{-(mx+b)}}$



if $y > 0.5$ Yes
else No

* Naive Bayes Classifier:

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

Naive because of assumption that features are independent of each other

Types: Bernoulli, Multinomial, Gaussian

features are: binary, discrete, continuous

Hyper-parameter Tuning:

→ Use k-fold cross validation score for different set of parameters & select the params with the best score.

→ Library : i) GridSearchCV

(does the same thing in one line instead of multiple for loops)

ii) RandomizedSearchCV :

∵ No. of permutation of parameter can be very high. selecting a fixed no. of random (parameter set) and then select the one with best score

L1 & L2 regularization :

L1 → LASSO regularization

L2 → RIDGE regularization

underfit

$y = \theta_0 + \theta_1 x$

overfit

$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots$

balanced fit

$y = \theta_0 + \theta_1 x + \theta_2 x^2$

Approach : make θ_3 and θ_4 close to 0

L2 regularization

$$mse = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - h_\theta(x_i)\right)^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

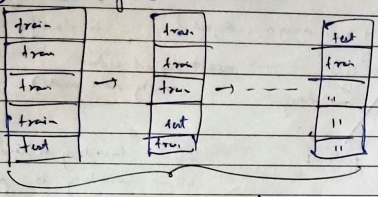
new term added that penalizes higher-θ values

$$mae = \frac{1}{n} \sum_{i=1}^{n} \left|y_i - h_\theta(x_i)\right| + \lambda \sum_{i=1}^{n} |\theta_i|$$

→ L1 regularization

K-fold cross-validation :

divided into 5 folds

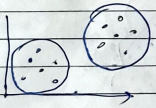


take the average of the score

* ML → supervised learning (i.e. know target variable / class) (↳ every algo studied so far (regression, classifier))
  → unsupervised "
  → reinforcement "

K-means clustering : (Unsupervised learning)

(no output)



K=2 ⇒ find 2 clusters

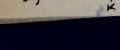
Approach :- i) Take K random centroids. The points closest to a centroid become part of that cluster.

ii) Readjust each centroid by finding the centroid of all the points of that cluster.

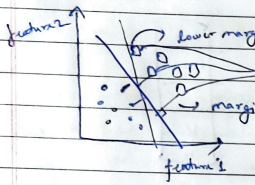
iii) Now repeating step 1 will give new set of points belonging to that cluster.

iv) Repeat the above process until no new points added to any of the cluster. Finally we get K diff. centroids

* To find K, plot (sum of squared errors) Vs K and the elbow point is desired K



Support Vector Machine :



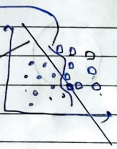
Maximize the margin to classify

2D → boundary is line
3D → boundary is plane.

Support vector machine draws a hyperplane in 'n' dimensional space : maximizes the margin btw classification groups.

→ High Gamma : consider only the nearby points for margin
low gamma : " all the points

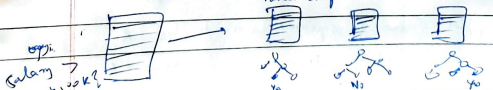
→ High regularization(C):
(overfitting)
low regularization(C):
(more error but that might be ok)



Random forest :

→ randomly sample the training data into multiple training sets & make a decision tree of each set.

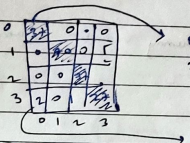
→ Then to predict, take the output from each of the decision tree & select the one with majority vote



→ Multiclass example : X: image of digit (represented in 2D array)
y: 0, 1, 2, 3, ..., 9

→ Confusion matrix:

cm = confusion_matrix (y_test, y_predicted)



Of all the test data, 37 times the model predicted the image as 0 when the image was actually 0.

2 times, the model predicted the image as 0 when the image was actually 3.

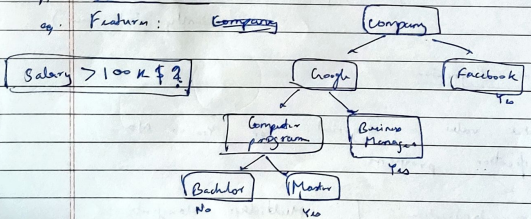
Truth (y_test)

Predicted (y_predicted)

Diagonal → Correct predictions

Decision Tree :

eg. Features: ~~Company~~



How to select ordering of features:

eg: on basis of company

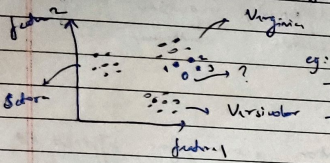
Facebook : 6 yes, 0 No — 6/0 (low entropy)

Google : 3 No, 1 yes

High Information gain ✓

→ Criteria of split : Gini / Entropy
(choose one of the two)

k nearest neighbour Classification :



eg: k = 3

→ find nearest k points [euclidian distance]

→ find the class having the maximum data points near to the test point.

→ Finding K → trial & error (eg GridSearchCV) (typically = 5)

PCA : principal component Analysis :

→ dimensionality reduction technique

(~~Reg~~ select the features having the maximum impact)

(eg: 100 features to 6 features ($PC_1, PC_2 \ldots, PC_6$)

⇒ the features are completely new (not a subset of the columns)

* Bias : how accurately a model can capture a pattern in training data. high train error ⇒ high bias

Variance : high test error ⇒ high variance

Overfitting : high variance, low bias

Balanced : low variance, low bias ✓

Basis → set of vectors in R^n that are LI and can represent any point in R^n

→ Orthonormal matrix : rows are mutually orthonormal
columns " " "

$A^T A = A A^T = I \Rightarrow A^{-1} = A^T$ eg: $A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$

Eigen decomposition :

$Av = \lambda v$ (vector v is scaled) — right eigen vector ?

square matrix, eigen vector, scalar called eigen value

eg: $A = \begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix}$ (symmetric)

i) $|A - \lambda I| = 0$ (determinant)

we get λ_1, λ_2

$\lambda_1 = 2, \lambda_2 = 0.5$

$\begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$\begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}$

$1.25 x_1 + 0.75 x_2 = \lambda x_1$

$1.25 x_1 + 0.75 x_2 = 2x_1$

$0.75 x_1 + 1.25 x_2 = \lambda x_2$

$0.75 x_1 = 0.75 x_2 \Rightarrow x_1 = x_2$

$x_1^2 + x_2^2 = 1$ (for unit eigen vector)

We get 2 eqn for $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$v_1 = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}$, $\lambda_1 = 2$ (scaled more)

$v_2 = \begin{bmatrix} -0.707 \\ 0.707 \end{bmatrix}$, $\lambda_2 = 0.5$ (scaled less)

Graph: $\|x\| = 1$ find Ax

We know two such x : v_1 & v_2

$Av_1 = \begin{bmatrix} 1.414 \\ 1.414 \end{bmatrix}$

$Av_2 = \begin{bmatrix} -0.354 \\ 0.354 \end{bmatrix}$

(0.707, 0.707) (1.414, 1.414)

Ellipse

→ Set of vectors is linearly independent if no vector in the set is a linear comb^n of other vectors

→ If column space is R^m ⇒ exactly m linearly independent columns

→ Square matrix with LI columns is called 'singular'

Norms:

→ intuitive meaning → distance of x from origin

$$L^p = \|x\|_p = \left(\sum_i |x_i|^p\right)^{\frac{1}{p}} \qquad p \in R,\ p \geq 1$$

$$L^2 = \left(\sum x_i^2\right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \dots x_n^2}$$

(analogous to magnitude of the vector)

Square $L^2_{norm} = x^T x = \sum x_i^2$

i.e. $\begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

→ $L^1 = \|x\| = \sum_i |x_i|$ (mod is used)

→ Frobenius norm $\|A\|_F = \sqrt{\sum_{i,j} A_{i,j}^2}$ (like L^2 norm) (for matrix, not vector)

→ Diagonal matrix: $D_{ij} = 0$ where $i \neq j$

Distance matrix: $A_{ij} = A_{ji}$

→ Unit vector: $\|x\|_2 = 1$

→ In R^n, atmost n vectors can be mutually orthogonal (90°) with non-zero norm.

eg: for R^3: $\hat{i}, \hat{j}, \hat{k}$ are mutually orthogonal

→ Vectors orthogonal & having unit norm ⇒ orthonormal

$\frac{a_1}{a_2} = \frac{b_1}{b_2} = \frac{c_1}{c_2}$ → infinite

$\neq \frac{c_1}{c_2}$ → no sol^n

→ Exactly one sol^n $\frac{a_1}{a_2} \neq \frac{b_1}{b_2}$

⇒ Span is entire 2D plane

(linear combination of the column vectors spans the entire plane)

$Ax = b \Rightarrow x = A^{-1}b$

→ Matrix non-invertible if not a square matrix or columns are linearly dependent (low rank)

Ax → linear comb^n of column vectors of A

$Ax = b$ → test whether b is in the span of A or column space or range

→ For 3D: If all 3 vectors in same plane i.e they are linearly dependent (any vector can be obtained by linear combination of other 2), then span is a plane.

Else span is entire 3D space

→ $Ax = b$ to have solution $\forall\, b \in R^m$, column space of A must be R^m

$A: m \times n$

A must have atleast m columns i.e $n \geq m$

eg: $A: 3 \times 2$ (here $n < m$) $\begin{bmatrix} - & - \\ - & - \\ - & - \end{bmatrix}$ $x: 2 \times 1$ $\begin{bmatrix} - \\ - \end{bmatrix}$ $b: 3 \times 1$ $\begin{bmatrix} - \\ - \\ - \end{bmatrix}$

$Ax \rightarrow$ 2D plane ; sol^n only when b lies in that plane.

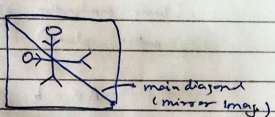
∴ ~~$n > m$~~ $n \geq m$ is necessary to have sol^n $\forall\, b$

Linear Algebra

→ 0D : scalar 2D : matrix
1D : vector
tensor : 0D or 1D or 2D or 3D . . .

→ **Matrix transpose**

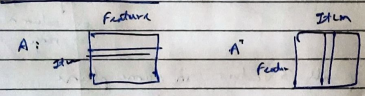
$B_{ij} = A_{ji}$
$(AB)^T = B^T A^T$



→ **Matrix multiplication:**

$A \cdot B = A_i \cdot B_j$ (dot product of vectors)

→ **Covariance matrix:** $A \cdot A^T$



$A \cdot A^T = C_{ij}$: how much similar i item is to j item

$A^T \cdot A = D_{ij}$: ——— feature i and feature j

→ **System of Equations :** $Ax = B$

$-4x_1 - 5x_2 = -13$
$-2x_1 + 3x_2 = 9$

if $\frac{a_1}{a_2} = \frac{b_1}{b_2}$
i.e. discriminant = 0,
then span or column space is a straight line
⇒ No soln or infinite solns.

$\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} = A = \begin{bmatrix} -4 & -5 \\ -2 & 3 \end{bmatrix}$ $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}$

$x_1 \begin{bmatrix} -4 \\ -2 \end{bmatrix} + x_2 \begin{bmatrix} -5 \\ 3 \end{bmatrix} = \begin{bmatrix} -13 \\ 9 \end{bmatrix}$

Feature Engineering :

PCA → Unsupervised

Linear regression :

↳ find $f: X_1 \times X_2 \dots \times X_n \longrightarrow Y$

Goal : minimize error / loss function

$\hat{y} = w^T x$

$\hat{y}$: predicted scalar

w^T : weighted values (coefficients)

x : $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ (given value of features to predict)

w : $\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$

$y = w_1 x_1 + w_2 x_2 \dots w_n x_n$

Objective : to minimize MSE train

$\nabla_w \text{MSE}_{train} = 0$ (Check other $= 0$)

$\nabla_w \frac{1}{m} \| \hat{y}^{(train)} - y^{(train)} \|_2^2 = 0$

$\hat{y}^{(train)} = X^{(train)} w$

$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ x_{21} & & x_{2n} \\ \vdots & & \\ x_{m1} & & x_{mn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$ → 1st training sample

n : # features, m : # training samples

$(m \times 1) \quad (m \times n) \quad (n \times 1)$

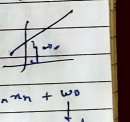
$(m \times 1)$

$w = \left(X^{(train)T} X^{(train)} \right)^{-1} X^{(train)T} y^{(train)}$

$(n \times 1) \quad (n \times m) \quad (m \times n) \quad (m \times 1)$

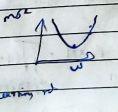
Finally $\hat{y}$ (predicted) $= [w^T \; w_0] [x \; 1]^T$

$= w_1 x_1 + w_2 x_2 \dots + w_n x_n + w_0$ ← bias term



* Gradient descent :

eg: $y = wx$



x_0 ← $x_0 - m \frac{dJ}{dx_0}$; α : learning rate

$x_{new} = x_0 - m\alpha$

(Reducing best fit line that minimize mse)

Probability, Bayes theorem problems, Gaussian Distribution

Date
Page

$P = [P_1\ P_2 \cdots P_n]$ (n×n)

P_1 → new basis

The rows are along vectors $P_1, P_2, \ldots P_n$ (orthonormal)

$[x_1\ x_2 \cdots x_n] \begin{bmatrix} | & | & & | \\ P_1 & P_2 & \cdots & P_n \\ | & | & & | \end{bmatrix}$

product give coordinate along P_1

$\hat{A} = AP$ (m×n)

($\hat{A}$ → matrix of transformed points)

$\frac{1}{m}\hat{A}^T\hat{A} = P^T \Sigma P = D$

(D → diagonal matrix)

$\frac{1}{m}\hat{A}^T\hat{A}$ → covariance matrix of transformed data

Σ → Covariance matrix of original data

Covariance matrix

let A = m×n

C = n×n :

$C_{ij} = \frac{1}{m}\sum_{i=1}^{m}(x_{i:} - \mu_i)(x_{i:} - \mu_j)$

If μ's are 0

$C_{ij} = \frac{1}{m}\sum_{i=1}^{m} x_{ki} \cdot x_{kj}$

dot product of the i^{th} column & j^{th} col

$= \frac{1}{m}(X^T X)_{ij}$

$\therefore P^T \Sigma P = D$

a) Which orthogonal matrix P diagonalizes Σ?

ans Matrix P whose columns are eigen vectors of $\Sigma = A^T A$

$\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{1n} \end{bmatrix} = \lambda \begin{bmatrix} P_{11} \\ P_{12} \\ \vdots \\ P_{1n} \end{bmatrix}$

$A^T A$ AP

Determinant :

$\det(A)$ → only for square matrix

= product of all eigen value of matrix

→ Matrix calculus ?

$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ $\quad \frac{d}{dx} Ax = A$

$Ax = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$f_1(x_1, x_2)$

$= \begin{bmatrix} x_1 + 2x_2 \\ 3x_1 + 4x_2 \end{bmatrix}$

$f_2(x_1, x_2)$

Principal component Analysis :

i) $\frac{dAx}{dx} = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = A$

new matrix with # row = # function

column = # variable

Ax vector (2×1) $(n\times1)$

x - vector (2×1) $(n\times1)$

$\frac{Ax}{dx} = 2x^T$ $(n\times n)$

ii) $\frac{d(x^T A x)}{dx}$ → assume symmetric

$= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11} & a \\ a & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

$= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11}x_1 + ax_2 \\ ax_1 + a_{22}x_2 \end{bmatrix}$

$= a_{11}x_1^2 + ax_2x_1 + ax_1x_2 + a_{22}x_2^2$

$= a_{11}x_1^2 + 2ax_1x_2 + a_{22}x_2^2$

$= f(x_1, x_2)$

$\frac{d(x^T A x)}{dx} = \begin{bmatrix} \frac{df}{dx_1} & \frac{df}{dx_2} \end{bmatrix} = \begin{bmatrix} 2a_{11}x_1 + 2ax_2 & 2a_{22}x_2 + 2ax_1 \end{bmatrix}$

$= 2 \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11} & a \\ a & a_{22} \end{bmatrix}$

$= 2x^T A$

$\boxed{\frac{d(x^T Ax)}{dx^T} = 2Ax}$

* If $y = Ax$ then $\frac{dy}{dz} = A\frac{dx}{dz}$

Th¹ → Eigen vectors of $A \in R^{n \times n}$ having distinct eigen values are LI

Th² → eigen vectors of square symmetric matrix are orthogonal.

$Q = V^T V$: $Q_{ij} = v_i^T v_j$ $= 0$ if $i \neq j$ (if vectors are normalized) $= 1$ if $i = j$ $\Rightarrow V^T V = I \Rightarrow \boxed{V^{-1} = V^T}$

→ Matrix is said singular if any of eigen value $= 0$

$f(x) = x^T A x$ when $\|x\|_2 = 1$

When x is eigen vector of A, & takes corresponding eigen value

$x^T A x = x^T \lambda x = \lambda x^T x = \lambda (\|x_i\|^2) = \lambda$

→ Eigen decomposition

Let A has n independent eigen vector $\{v^1, v^2, \ldots v^n\}$ with corresp. eigen val $\{\lambda_1, \lambda_2, \ldots \lambda_n\}$

Let $V = [v^1, v^2, \ldots v^n]$ ← matrix

$\lambda = [\lambda_1, \lambda_2, \ldots \lambda_n]$

V^T exists when $v^1, v^2 \ldots v^n$ have distinct eigen values

$AV = V \, \text{diag}(\lambda) \Rightarrow \boxed{A = V \, \text{diag}(\lambda) V^{-1}}$ ← eigen value decomp.

(if A is square symmetric, then $A = V \, \text{diag}(\lambda) V^T$)

eg: $\begin{bmatrix} 1.25 & 0.75 \\ 0.75 & 1.25 \end{bmatrix} \begin{bmatrix} 1.414 & -0.354 \\ 1.414 & 0.354 \end{bmatrix} = \begin{bmatrix} 1.414 & -0.354 \\ 1.414 & 0.354 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$

$(A) \qquad (V) \qquad (V) \qquad (\text{diag}(\lambda))$

$\boxed{\text{diag}(\lambda) = V^{-1} A V}$ ← diagonal matrix of A

Trace Operator :

→ $Tr(A) = \sum_i A_{i,i}$ (sum of main diagonal elements)

→ $\|A\|_F = \sqrt{\sum_{ij} (A_{ij})^2} = \sqrt{Tr(AA^T)}$ $\quad A \cdot A^T = \begin{bmatrix} - & - & - \\ - & - & - \end{bmatrix} \begin{bmatrix} | & | \\ - & - \\ | & | \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$

→ $Tr(A) = Tr(A^T)$

$Tr(a) = a$ $\qquad Tr(AA^T) = \sum A_{ij}^2$

→ $Tr(ABC(D)) = Tr(DABC) = Tr(CDAB) = Tr(BCDA)$

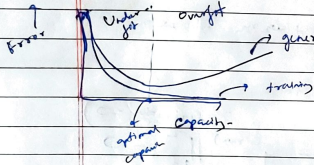
→ Training error / Generalization error
(error on training set) (error on testing set) (unseen data)

Overfitting → low training error, high generalization error

Capacity:

→ High capacity ⇒ overfitting (eg: $ax+b \rightarrow ax^2+bx+c$)

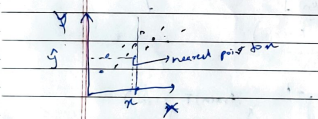
→ learning algo reduces the training error (doesn't find the best function)



* Nearest Neighbour regression:

→ Non-parametric models (no parameter vector)

→ For a given x, $\hat{y} = y_i$ where $i = \text{argmin} \|X_i - x\|_2^2$ (Euclidean dist²)



Logistic Regression:

$$p(x) = \frac{1}{1+e^{-(w_0+w_1 x)}}$$

$$\frac{p(x)}{1-p(x)} = e^{w_0+w_1 x} \rightarrow \text{odds}$$

Maximum likelihood;

Maximum $L(w_0, w_1) = \prod_{x_i: y_i=1} p(x_i) \cdot \prod_{x_j: y_j=0} (1-p(x_j))$

taking log (log likelihood)

$$= \prod p(x_i)^{y_i} \cdot (1-p(x_i))^{1-y_i}$$

$$\sum_{i=1}^{n} y_i \log(p(x_i)) + (1-y_i)\log(1-p(x_i))$$

NPTEL: Sequence → RNN, LSTM
Images → CNN

0 → 9

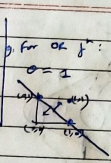
Date
Page

McCulloch Pitts Neuron:

$y = 1$ if $\sum_{i=0}^{n} x_i \geq \theta$ (threshold)

$\Rightarrow y$ if $\sum_{i=0}^{n} x_i < \theta$

for OR f: $\theta = 1$



Artificial neuron
$y \in \{0,1\}$
x_1 (inputs)

Perceptron: (weighted adder)

$y = 1$ if $\sum_{i=0}^{n} w_i x_i \geq \theta$ ⇒ $\sum_{i=0}^{n} w_i x_i \geq 0$

$= 0$ if $\sum_{i=0}^{n} w_i x_i < \theta$

($x_0 = 1$, $w_0 = -\theta$)

→ real valued inputs allowed

→ threshold θ can be learned

→ Single perceptron: can not classify if the function is not linearly separable.

Perceptron learning Algo:

Goal: that divides input in two halves $\quad w^T x = 0$

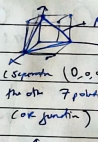
$x_1 + x_2 - 1 = 0$

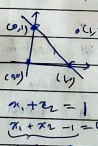
Initialize with random weights $\quad w_0 = 1,\ w_1 = 1,\ w_2 = 1$

for each point x:

if $x \in P$ & $w \cdot x < 0$, $w = w + x$

if $x \in N$ & $w \cdot x \geq 0$, $w = w - x$





$x_1 + x_2 = 1$

$x_1 + x_2 - 1 = 0$

$f(x) \geq 0 \rightarrow y = 1$

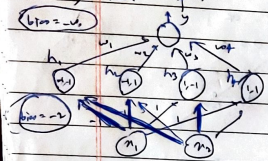
$< 0 \rightarrow y = 0$

Issue: Not linearly separable eg: XOR

→ Multilayered network of perceptrons (MLP) can represent any boolean f^n (may be linearly inseparable)

* Any boolean f^n of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons & 1 output layer with 1 perceptron

Eg: XOR $\hat{f}$:



x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$ with
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_1 ✓
1	0	1	0	0	1	0	w_1 ✓
1	1	0	0	0	0	1	w_4

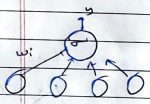
Let 'bias' be w_0 : $\sum_{i=1}^{4} w_i h_i \geq w_0$

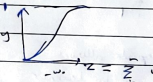
Then $w_1 < w_0$ (not firing) (output neuron), $w_2 \geq w_0$ (fired), $w_3 \geq w_0$, $w_4 < w_0$

— : -1

— : 1

Sigmoid Neuron :





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

smooth, continuous, differentiable

$y = \frac{1}{1+e^{-(\sum_{i=0} w_i x_i)}} = \frac{1}{1+e^{-(w^T x)}}$

Input : x_i — real values $x \in R^n$ (We are moving from boolean function to real valued $\hat{f}$)

Output : y → real between 0 and 1

Machine Learning setup : Components :

i) Data : $\{x_i, y_i\}_{i=1}^{n}$

ii) Model : Approximation of relation b/w x and y

eg: $\hat{y} = \frac{1}{1+e^{-(w^T x)}}$ → logistic regression

or $\hat{y} = w^T x$ → linear regression

or $\hat{y} = w^T w x$ → quadratic

iii) Parameter : 'w' is a parameter that needs to be learnt

iv) Learning algo : algo to learn w (eg: perceptron learning algo, gradient descent)

v) Objective / Loss / Error Function :

Learning algo should minimize the loss $\hat{f}$.

eg: $L(w) = \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$ (squared sum error)

→ At the output layer

$f(x) = h_L(x) = O(a_L(x))$

↳ output activation fn (eg softmax, linear, etc)

Parameters: (for all supervised learning algos).

i) Data : $\{x_i, y_i\}_{i=1}^{N}$ (N training examples)

ii) ~~Parameter~~ Model = $\hat{y}_i = f(x_i)$

$= O(W^3 g(W^2 g(W^1 x + b_1) + b_2) + b_3)$

iii) Parameter: $\theta = W_1, \ldots W_L, b_1, b_2 \ldots, b_L$ $(L=3)$

iv) Algorithm: Gradient descent with back propogation

v) Objective / loss function:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} (\hat{y}_{ij} - y_{ij})^2$$

i.e minimize $L(\theta)$.

$$\theta = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_L \\ b_1 \\ b_2 \\ \vdots \\ b_L \end{bmatrix} \qquad \nabla\theta = \begin{bmatrix} \frac{\partial L(\theta)}{\partial W_1} \\ \vdots \\ \frac{\partial L(\theta)}{\partial b_L} \end{bmatrix}$$

$$\theta_{t+1} = \theta_t - \eta \nabla\theta_t$$

$\nabla\theta$ → ~~total~~ contains derivatives wrt all weights

eg: $\frac{\partial L(\theta)}{\partial W_{11}} \ldots \frac{\partial L(\theta)}{\partial W_{1m}}, \frac{\partial L(\theta)}{\partial W_{21}} \ldots \frac{\partial L(\theta)}{\partial W_{nm}}, \ldots$

Total derivatives = $(L-1)(n \times n) + 1 \times (n \times k)$ + $(L-1) \times n + 1 \times (K \times 1)$

b

$y = f(x)$

$f(x) = \frac{1}{1 + e^{-(wx+b)}}$ $\quad L = \frac{1}{2}(f(x) - y)^2$

$\nabla_w = \frac{1}{2} \cdot 2 (f(x) - y) \frac{\partial f}{\partial w}$

$\nabla_b = (f(x) - y) \cdot f(x) \cdot (1 - f(x)) \cdot x$

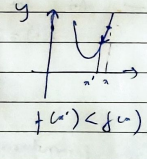
$(f(x) - y) \; f(x) \; (1 - f(x))$

$\sigma'(x) = \sigma(1 - \dots$

Gradient descent :

$$x' = x - \epsilon \nabla_x f(x)$$

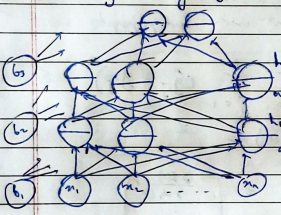
new point — current point — learning rate — gradient vector



$f(x') < f(x)$

eg: $\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \epsilon \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$

Multilayered feedforward neural network:



→ Output layer : k neurons (corresponding to k classes)

(L-1) hidden layers

h_i → activation

a_i → pre-activation

→ Input layer n-D vector ($L=0$)

→ Hidden layers & output layers :

pre-activation : aggregation (Σ)

activation : non-linearity

→ $W_i \in \mathbb{R}^{n \times n}$ $\quad (i < 0 < L)$ & $b_i \in \mathbb{R}^n$

$W_L \in \mathbb{R}^{n \times K}$ and $b_L \in \mathbb{R}^K$ (for output layer)

→ $h_L = \hat{y} = f(x)$

⇛ $a_i(x) = b_i + W_i h_{i-1}(x)$

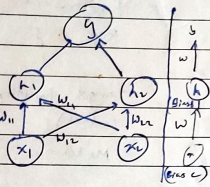
$b_i \in \mathbb{R}^n$, $W_i \in \mathbb{R}^{n \times n}$, $h_{i-1}(x) \in \mathbb{R}^n$, $a_i(x) \in \mathbb{R}^n$

$h_i(x) = g(a_i(x))$

(activation fn g applied on every element of vector)

Simple FFN with hidden layer:

$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



(2×2) — # neurons in input layer; # neurons in hidden layer

$$W^T x + c$$

$$= \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

bias

$$h = g(W^T x + c)$$

activation function (g) (g: sigmoid, tanh, ReLU)

$$\therefore \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = g\left(\begin{bmatrix} w_{11} x_1 + w_{21} x_2 + c_1 \\ w_{12} x_1 + w_{22} x_2 + c_2 \end{bmatrix} \right)$$

Let $g(z) = \max\{0, z\}$ (ReLU activation f^n)

Now, $y = [w_1 \; w_2] \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} + b$

$$y = w^T \max\{0, W^T x + c\} + b$$

$\Rightarrow f(x; W, c, w, b)$

Example: A solution for XOR problem:

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b = 0$$

Let $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\max(2, 0) = 2$, $\max(1, 2) = 1$

$$W^T x = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \quad W^T x + c = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad g(W^T x + c) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

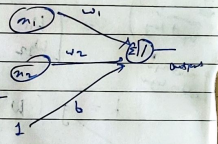
$$y = [1 \; -2] \begin{bmatrix} 2 \\ 1 \end{bmatrix} + 0 = 0 + 0 = 0 \qquad (\because 1 \wedge 1 = 0)$$

Feedforward neural network example:

XOR $f^* : X = \{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \}$

Target function (actual) : $y = f^*(x)$

Model : $y = f(x; \theta)$



let cost function = MSE

$$J(\theta) = \frac{1}{4} \sum_{x \in X} (f^*(x) - f(x;\theta))^2$$

let we consider a linear model

$$y = f(x;\theta) = f(x;w,b) = x^T w + b$$

$$J(\theta) = \frac{1}{4} \left[\left(0 - \underbrace{(w_1 x_1 + w_2 x_2 + b)}_{x_1=0,\, x_2=0} \right)^2 + \left(1 - \underbrace{(w_1 x_1 + w_2 x_2 + b)}_{x_1=0,\, x_2=1} \right)^2 + \left(1 - \underbrace{(\quad)}_{x_1=1,\, x_2=0} \right)^2 + \left(0 - \underbrace{(\quad)}_{x_1=1,\, x_2=1} \right)^2 \right]$$

$$= \frac{1}{4} \left[b^2 + (1 - w_2 - b)^2 + (1 - w_1 - b)^2 + (1 - w_1 - w_2 - b)^2 \right]$$

$\frac{\partial J}{\partial w_1} = 0 \quad \frac{\partial J}{\partial w_2} = 0 \quad \frac{\partial J}{\partial b} = 0$ } solve for w_1, w_2, b

$-2(1 - w_1 - b) - 2(1 - w_1 - w_2 - b) = 0$

$2 - 2w_1 - w_2 - 2b = 0$

$2 - 2w_2 - w_1 - 2b = 0$

$w_1 = w_2$

$2 - 2w_2 - w_2 = 0$

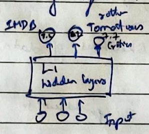
$2 = 3w_2$

Date ______
Page ______

	Outputs	
	Real Values	Probabilities
Output Activation	Linear	Softmax
Loss function	Squared error	Cross-entropy

→ Output are real values (eg. regression), then activation f^n is linear (bounded if required)

eg:

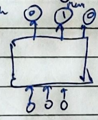


(output) $O(\ldots) \to 0$ to 10

Here generally cost funct^n used is squared error $= \left(= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{2} (\hat{y}_{ij} - y_{ij})^2 \right)$

→ When output are probabilities :

eg: classification :



$\sum p_i = 1$

Activat^n f^n : softmax

$$\hat{y}_j = O(a_L)_j = \frac{e^{a_{L,j}}}{\sum_{i=1}^{K} e^{a_{L,i}}}$$

$f(x) = h_L = O(a_L)$
$= W_0 a_L + b_0$
(sigmoid not used as we don't want value to be b/w 0 and 1)

$\sum \hat{y}_j = 1$ & each value is b/w 0 and 1

→ Information content : $I(A) = -\log_2(P(A))$

(less the probability more the info content)

→ Entropy = $\sum p(i)\, I(i)$
$= -\sum p_i \log(p_i)$

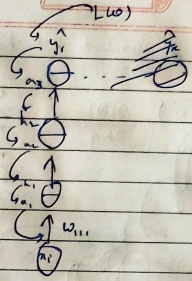
→ Cross-entropy : $\boxed{-\sum p_i \log(q_i)}$

p → true distribution
q → predicted distribution

(minimized when true distribution is as close to predicted distribution.

Backpropagation in FFNN:

$$\frac{\partial L(\Theta)}{\partial w_{11}} = \frac{\partial L(\Theta)}{\partial \hat{y}_1} \cdot \frac{\partial \hat{y}_1}{\partial a_3} \cdot \frac{\partial a_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial a_2} \cdot \frac{\partial a_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_{111}}$$



$g: L(\Theta) = -\log \hat{y}_k$ (k = true class)

(i) $\Rightarrow \frac{\partial (L(\Theta))}{\partial \hat{y}_i} = -\frac{\partial}{\partial \hat{y}_i}(-\log \hat{y}_k) = \frac{-1}{\hat{y}_k}$ if $i = k$

$= 0$ otherwise

$\Rightarrow \frac{\partial (L(\Theta))}{\partial \hat{y}_i} = \frac{-1_{(i=k)}}{\hat{y}_k}$

Gradient wrt output

$$\nabla_{\hat{y}} L(\Theta) = \begin{bmatrix} \frac{\partial L}{\partial \hat{y}_1} \\ \vdots \\ \frac{\partial L}{\partial \hat{y}_K} \end{bmatrix} = \frac{-1}{\hat{y}_k} \begin{bmatrix} 1_{k=1} \\ 1_{k=2} \\ \vdots \\ 1_{k=K} \end{bmatrix}$$

(ii) $\frac{\partial L}{\partial a_{Li}} = \frac{\partial L}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial a_{Li}}$

$= \frac{-\partial(-\log \hat{y}_k)}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial a_i}$ → depends on a_{Li} as softmax used

$= -\frac{1}{\hat{y}_k} \cdot \frac{\partial}{\partial a_i} \text{softmax}(a_L)_k$

$\hat{y}_k = \frac{e^{a_k}}{\sum e^{a_i}}$

$= \frac{-1}{\hat{y}_k}(1_{k=i} \hat{y}_k - \hat{y}_k \hat{y}_i)$

$= -(1_{k=i} - \hat{y}_i)$

$$\nabla_{a_L} = \begin{bmatrix} \frac{\partial L}{\partial a_{L1}} \\ \vdots \\ \frac{\partial L}{\partial a_{LK}} \end{bmatrix} = \begin{bmatrix} -(1_{k=1} - \hat{y}_1) \\ -(1_{k=2} - \hat{y}_2) \\ \vdots \\ -(1_{k=K} - \hat{y}_K) \end{bmatrix}$$

* Representative power of MLP:

→ Single hidden layer can be used to represent any boolean fn precisely (no errors)

→ Hidden layer neurons = 2^n (n = input $\in R^n$)

* Representative power of multilayered network of sigmoid neurons:

→ A single hidden layer can be used to approximate any continuous function to any desired precision.

i.e for any f^n $f(x): R^n \to R^m$, we can find neural network with enough neurons in 1 hidden layer whose output $g(x)$ satisfies $|g(x) - f(x)| < \epsilon$

y output

→ more the no of towers f^n, better approximation

→ $f(x)$ = sum of tower functions

tower f^n x

aggregate (h_{21}) (output) of

Neural network:

h_{11} h_{12}

$w_{1,1}$ $w_{1,2}$

x_1

1D input → 2 neurons to construct tower

2D → 4 neurons

$O(2^n)$

1 → 1 hr

2 → 1 hr 15 m

3 → 1 hr 30 m

4 → 3 hr

5 → 2 hr 15 m

only study

6 → 8 hr

7 → 2 hr

8 → 2 hr

2.1 + 4.2 + 1.3 + 8.4

2 + 8 + 18 + 32

Date: Page:

↗ stochastic gradient descent

SGD example :

Points : (x, y) : (1,2) , (2,4) , (3,6) , (4,8)

$y = wx$ (we need to find w that minimizes the cost function MSE)

$w_0 = 4, \quad \epsilon = 0.1$

$$\nabla_\theta J(\theta) = \frac{1}{4}\left[\sum_{i=1}^{4} \nabla_\theta \underbrace{L(x^{(i)}, y^{(i)}, \theta)}\right]$$

$$L = (y_i - wx_i)^2$$

$$\frac{\partial L}{\partial w} = 2(y_i - wx_i)$$

Gradient descent rule :

→ move in the direction opposite to gradient.

Let $\theta\,(w, b)$

$$\theta_{t+1} = \begin{bmatrix} w_{t+1} \\ b_{t+1} \end{bmatrix} = \begin{bmatrix} w_t \\ b_t \end{bmatrix} - \eta \begin{bmatrix} \nabla w_t \\ \nabla b_t \end{bmatrix}$$

where $\nabla w_t = \left.\frac{\partial L\,(w,b)}{\partial w}\right|_{w=w_t,\, b=b_t}$

$\nabla b = \left.\frac{\partial L}{\partial b}\right|_{w=w_t,\, b=b_t}$

SGD :

→ Randomly split training data into mini-batches.

→ Compute gradient descent for all mini-batch & average them.

Gradient descent :

for each training data, we calculate its impact on

$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ b \end{bmatrix}$ & then average it for all training data

	1	2	3	4	Average over all training data
w_0	-	-	-	-	- 0.28
w_1	-	-	-	-	+ 0.12
w_2	-	-	-	-	- 0.20
⋮	⋮	⋮	⋮	⋮	⋮
b	-	-	-	-	+ 0.04

$$-\eta\, \nabla C(w_1, w_2, \ldots, b_1, \ldots b_k)$$

$w_{new} = w - \eta \nabla C$

eg:

$w_1' = w_1 + 0.12$

$\therefore L(\theta) = -\sum_{c=1}^{K} y_c \log \hat{y}_c$ ← predicted

$y_c = 1$ if $c = l$
$= 0$ otherwise
(l is true class label)

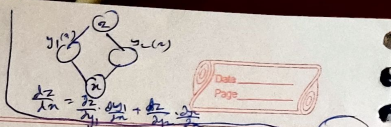
$\Rightarrow L(\theta) = -\log \hat{y}_l$ → predicted probability of true class

minimize $L(\theta)$ ~~maximize~~

or maximize $-L(\theta) = \underbrace{\log \hat{y}_l}_{\text{log-likelihood of data}}$

($\hat{y}_l$ is a f^n of θ $= [w_1, w_2, \ldots b_1, \ldots]$)

(Maximizing log-likelihood is equivalent to minimizing ~~MSE~~ MSE)



* Gradient wrt hidden layer:

(iii) $$\frac{\partial L(\theta)}{\partial h_{ij}} = \sum_{m=1}^{k} \frac{\partial L}{\partial a_{i+1,m}} \cdot \frac{\partial a_{i+1,m}}{\partial h_{ij}}$$

$$= \sum_{m=1}^{k} \frac{\partial L}{a_{i+1,m}} \left(w_{i+1,m,j} \right)$$

$$= \left(w_{i+1,\ldots,j} \right)^T \nabla_{a_{i+1}} L(\theta)$$

$$\nabla_{h_i} L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial h_{i1}} \\ \vdots \\ \frac{\partial L}{\partial h_{ik}} \end{bmatrix} = \left(W_{i+1} \right)^T \left(\nabla_{a_{i+1}} L(\theta) \right)$$

iv) $$\frac{\partial L(\theta)}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} \cdot \frac{\partial h_{ij}}{\partial a_{ij}} = \frac{\partial L(\theta)}{\partial h_{ij}} \cdot g'(a_{ij})$$

$(i < L)$

* Gradient wrt weight:

$$a_k = b_k + W_k h_{k-1}$$

v) $$\frac{\partial a_{ki}}{\partial W_{kij}} = h_{k-1,j}$$

$$\nabla_{W_k} L(\theta) = \begin{bmatrix} \frac{\partial L}{\partial W_{k0,0}} & \cdots & \frac{\partial L}{\partial W_{k0,n-1}} \\ \vdots & & \vdots \\ \frac{\partial L}{\partial W_{k,n-1,0}} & \cdots & \frac{\partial L}{\partial W_{k,n-1,n-1}} \end{bmatrix}$$

$$\frac{\partial L(\theta)}{\partial W_{kij}} = \frac{\partial L(\theta)}{\partial a_{ki}} \cdot \frac{\partial a_{ki}}{\partial W_{ki,ij}} \quad \left(\frac{\partial a_{ki}}{\partial W_{ki,ij}} = h_{k-1,j} \right)$$

$$= \nabla_{a_k} L(\theta) \cdot \left(h_{k-1} \right)^T$$

vi) $$a_{ki} = b_{ki} + \sum_j W_{kij} h_{i-1,j}$$

$$\frac{\partial L}{\partial b_{ki}} = \frac{\partial L(\theta)}{\partial a_{ki}} \cdot \frac{\partial a_{ki}}{\partial b_{ki}} \quad \left(\frac{\partial a_{ki}}{\partial b_{ki}} = 1 \right) \Rightarrow \frac{\partial L}{\partial b_{ki}} = \frac{\partial L(\theta)}{\partial a_{ki}} \quad \left(\nabla_{b_k} L(\theta) = \nabla_{a_k} L(\theta) \right)$$

Stochastic gradient descent:

→ Normal gradient descent:

Iterate over entire training data to find dw and db

& then update $w = w - \alpha \times dw$, $b = b - \alpha \times db$

→ Stochastic gradient descent:

Update w & b after each training data (mini-batch gradient descent)

eg: if batch size = 1

update w and b after every training data encountered.

if batch size = k

Update dw & db after k points encountered

(typical: 16, 32, 64)

$$dw = \sum_{i=1}^{n} (f(x) - y) f(x) \cdot (1 - f(x)) \cdot x$$

$f(x) = \sigma(x)$

(x, y) → from training data

iterating through entire training data (n items)

& then updating w in case of normal gradient descent

→ In 1 epoch (one pass over entire data) : N data points

B → mini-batch size

Gradient descent → 1

Stochastic " " → N

Mini batch " " → N/B

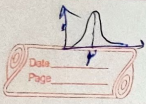
No. of steps in 1 epoch

("1 step" = 1 update of params)

Normal distribution:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

prob. density; f^2; standard deviation; mean



Bias & Variance:

* **Bias**: let $f(x)$ → true model
$\hat{f}(x)$ → estimate of the model

→ $\text{Bias}(\hat{f}(x)) = \underbrace{E[\hat{f}(x)]}_{\text{exp. value of model}} - f(x)$

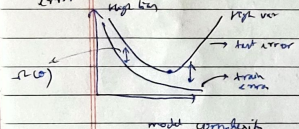
→ Simple model has high bias (underfitting)
Complex model → low bias (overfitting)

* **Variance**:

$$\text{Variance}(\hat{f}(x)) = E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right]$$

→ Simple model → low variance
complex " → high variance

Intuition: how much the different $\hat{f}(x)$ differ from each other



MSE:

$$E\left[(y - \hat{f}(x))^2\right] = \text{Bias}^2 + \text{Variance} + \sigma^2$$

(σ^2: irreducible error)

→ While training instead of minimizing training error $L_{train}(\theta)$, minimize:

$$\min_{w.r.t\ \theta} L_{train}(\theta) + \underbrace{\Omega(\theta)}_{\text{high for complex model}} = L(\theta)$$

L2 regularization: $\tilde{L}(w) = L(w) + \frac{\alpha}{2}\|w\|^2$

For SGD: $\nabla\tilde{L}(w) = \nabla L(w) + \alpha w$

i.e. $w_{t+1} = w_t - \eta \nabla L(w_t) - \eta\alpha w_t$

more weight will be penalized more

Solving, we get that to minimize $\frac{1}{m} x^T x = c$

$$\min_{P_{k+1}, \dots P_n} \sum_{j=k+1}^{n} p_j^T m C p_j \quad \because p_j^T p_j = 1$$

$$\forall j = k+1, k+2, \dots, n$$

(minimize $P^T \Sigma P^T$ → pick minimum eigen value)

Soln: pick $(n-k)$ smallest eigen ~~values~~ vectors & and discard them or pick the K largest eigen vectors

eg Let $v_1 = [1, 1]$, $v_2 = [-1, 1]$ are two basis vectors

Normalize $\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \Rightarrow P = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$

For the point $x = [3.3, 3]$

$$\hat{x} = xP = [3.3 \quad 3] \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$= \left[\frac{6.3}{\sqrt{2}} \quad \frac{-0.3}{\sqrt{2}} \right] = [\alpha_1 \; \alpha_2]$$

(∵ x can be reconstructed by $\alpha_1 v_1 + \alpha_2 v_2 = [3.3, 3]$)

Now with fewer dimensions ($k = 1 < n$)

(using only v_1 dimension)

$$\hat{x} = xP = [3.3 \quad 3] \underbrace{\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}}_{v_1} = \begin{bmatrix} 6.3 \\ \sqrt{2} \end{bmatrix}$$

$$x = \alpha_1 v_1 = \frac{6.3}{\sqrt{2}} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = [3.15 \quad 3.15]$$

(reconstruction with minimum error)

→ The i^{th} dimension of transformed pts $\hat{X}$:

$$\hat{X}_i = X P_i$$

Variance along this dimension $= \frac{\hat{X}_i^T \hat{X}_i}{m} = \frac{1}{m} P_i^T X^T X P_i = \frac{1}{m} P_i^T \lambda P_i$

$$= \frac{\lambda_i}{m}$$

(⇒ High eigen value ⇒ high variance)

→ $P = \frac{1}{m} X^T X$

→ $X^T X$ is symmetric ∵ $(X^T X)^T = X^T X$

→ P : orthogonal matrix

Orthonormal matrix
$A^{-1} = A^T$

Date
Page

We want $P^T \Sigma P = D$

but Σ is a square symmetric matrix
⇒ eigen vectors of Σ is orthogonal

Which orthogonal matrix P diagonalizes Σ ?

$D =$

Sol^n → Matrix P whose columns are eigen vectors of $\Sigma = X^T X$

$D = V^{-1} \Sigma V = P^T \Sigma P$

V → matrix whose columns are eigen vectors of $(\Sigma = X^T X)$

→ eigen vectors of $X^T X$ are orthogonal since $(X^T X)$ is symmetric

Procedure :

i) Find $\Sigma = X^T X$ (before it make mean of X : Onecan)

ii) Find eigen vectors of Σ : $P_1, P_2, \ldots P_n$ & form P (normalize these vectors)

iii) $\hat{X} = XP$

iv) To represent in K dimensions, select eigen vectors corresponding to top k eigen values & compute $\hat{X}' = XP'$ $\quad X \to m \times n$, $P' \to n \times K \Rightarrow \hat{X} = m \times K$

v) To reconstruct X: $x_i = \alpha_{i1} v_1 + \alpha_{i2} v_2 + \ldots + \alpha_{iK} v_K$

$$x_i = \sum_{j=1}^{n} \alpha_{ij} P_j$$

Taking the top K dimensions Cyclic nature of using less reduced dimension

$$\hat{x}_i = \sum_{j=1}^{K} \alpha_{ij} P_j$$

Select P_i's to minimize $e = \sum_{i=1}^{m} (x_i - \hat{x}_i)^T (x_i - \hat{x}_i)$

$p_1, p_2, \ldots, p_n$ are LI orthonormal vectors

$P = [p_1\ p_2 \cdots p_n]$ is the new basis

$x_1, x_2, \ldots x_m \in \mathbb{R}^n$ are data points

Representn of x_i in new basis P:

$x_i = \alpha_{i1} p_1 + \alpha_{i2} p_2 \cdots + \alpha_{in} p_n$
$(n \times 1)$

∵ basis is orthonormal, we can find the α_{ij}'s using

$\alpha_{ij} = x_i^T p_j = [\leftarrow x_i \rightarrow]\begin{bmatrix} \uparrow \\ p_j \\ \downarrow \end{bmatrix}$

Stacking

$\hat{X} = XP$

$\hat{x}_i = x_i^T [p_1 \cdots p_n]$ $\quad [\alpha_{i1} \cdots \alpha_{in}]$

$\hat{X} \rightarrow m \times n$
→ also reduced
→ matrix of transformed points

$\hat{x}_i^T = x_i^T P \Rightarrow \hat{X} = XP = \begin{bmatrix} \leftarrow x_1 \rightarrow \\ \leftarrow x_2 \rightarrow \\ \leftarrow x_m \rightarrow \end{bmatrix}\begin{bmatrix} \uparrow & \uparrow & \uparrow \\ p_1 & p_2 \cdots & p_n \\ \downarrow & \downarrow & \downarrow \end{bmatrix}$
$m \times n \qquad n \times n$

$\hat{x}_m^T = x_m^T P$

Now, $\hat{X} = XP$

Covariance of $\hat{X}$ matrix $= \frac{1}{m} \hat{X}^T \hat{X} = \frac{1}{m} (XP)^T XP = P^T \left(\frac{1}{m} X^T X\right) P$

$= P^T \Sigma P$

$\Sigma \rightarrow$ covariance matrix of X

We want, $\left(\frac{1}{m} \hat{X}^T \hat{X}\right)_{ij} = 0 \quad i \neq j$ (Covariance = 0) (non-redundant data)

$\neq 0 \quad , i = j$ (variance $\neq 0$) (high variance)

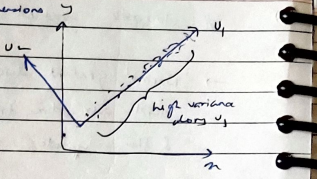
Hence $\frac{1}{m} \hat{X}^T \hat{X} = P^T \Sigma P = D$ — diagonal matrix

Standardize the data means making 0 mean & unit variance

PCA : represent data with fewer dimensions

m points, n features

	x	y	z
i)	–	–	–
ii)	–	–	–
m)	–	–	–



correlation b/w y & z

$$\rho_{yz} = \frac{\text{covariance}(y,z)}{\sigma_y \sigma_z} = \frac{\sum_{i=1}^{n} (y_i - \bar{y_i})(z_i - \bar{z})}{\sqrt{\sum_{i=1}^{n} (y_i - \bar{y})^2} \sqrt{\sum_{i=1}^{n} (z_i - \bar{z})^2}}$$

σ = standard deviation = $\sqrt{\text{variance}}$

Variance = $\sigma^2 = \frac{\sum (x_i - \bar{x})^2}{m}$

$$X_{m \times n} = \begin{bmatrix} \leftarrow x_1 \rightarrow \\ \leftarrow x_2 \rightarrow \\ \vdots \\ \leftarrow x_m \rightarrow \end{bmatrix} \qquad P = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ p_1 & p_2 & \cdots & p_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}_{n \times n}$$

Covar(y,z) = $\frac{1}{m} \sum (y_i - \bar{y})(z_i - \bar{z})$

* Covariance matrix : C_{ij} stores the covariance b/w column i & j

→ if the columns of X are 0 mean

then $C_{ij} = \frac{1}{m} \sum_{k=1}^{m} (x_{ki} - \mu_i)(x_{kj} - \mu_j)$

$= \frac{1}{m} \sum_{k=1}^{m} x_{ki} x_{kj}$ $\quad (\because \mu_i = \mu_j = 0)$

$X^T X$

$= \frac{1}{m} X_i^T X_j = \frac{1}{m} (X^T X)_{ij}$

C_{ij} = dot product of i^{th} col & j^{th} col of X

$\therefore$ $C = \Sigma = \boxed{\frac{1}{m} X^T X}$ ← covariance matrix

Momentum is calculated by taking the weighted avg of previous momentum V_{t-1} & gradient at the lookahead position $(\theta_t + \alpha V_{t-1})$

Momentum term adds the influence of previous gradients → more robust against oscillations & slow convergence compared to classic momentum term

Batch Normalization:

$\frac{\partial L}{\partial W}$ → calculate the loss at the lookahead position:

$$L = \frac{1}{N} \sum_{i=1}^{N} J(f(x_i, \theta), y_i)$$

$$L = \frac{1}{m} \sum J(f(x_i, \tilde{\theta}), y_i)$$

where $\tilde{\theta} = \theta_t + \alpha v^{t-1}$

$$\theta^{t+1} \leftarrow \theta^t - \eta \nabla_\theta J$$

(Problem: Hyperparam. α, ϵ are constant based on observation)

Consider $B = \{x_1, x_2, \ldots, x_m\}$

$$\hookrightarrow L = \frac{1}{M} \sum_{i=1}^{m} J(\ )$$

* Internal covariate shift:

when the distribution of the input features vary across the mini-batch.

would help if pre-activation at each layer is normalized

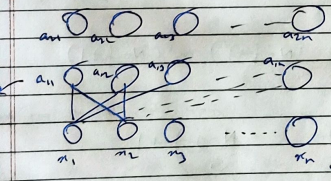
incured by standardizing the pre-activation at each unit.

eg: feature extraction of cats:

in one mini-batch - white cats

in another ⇒ black cats

Convergence will take time



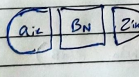
pre-activation

expectation taken over entire set of mini-batches

$$\hat{a}_{ik} = a_{ik} - E[a_{ik}]$$

$$\overline{SE[a_{ik}]}$$

(scaling and transformation)



$= Relu(BN_{a_{ik}})$

$$y_{ik}^{(k)} = \gamma^{(k)} \hat{a}_{ik} + \beta^{(k)}$$

(rescaling and retransformation in final output layer)

Homework: With batch normalization, how gradient descent & back propagation will work.

all not only) matrix

Optimization :

i) Gradient descent : $\theta^{t+1} = \theta_t - \varepsilon \nabla_\theta J$

$L = \frac{1}{N} \sum_{i=1}^{N} J(f(x^i, \theta), y^i)$

Problem → High computation time to all N sample training data cases

ii) SGD : batch size = 1 → choose a random point → let me

Now $L = J(f(x^r, \theta), y^r)$

Problem → slow convergence

iii) Mini batch → $L = \frac{1}{m} \sum_{i=1}^{m} J(f(x^i, \theta), y^i)$

* Bottleneck : estimating ε ($\varepsilon \uparrow$ overshooting or slow convergence) ($\varepsilon \downarrow$)

Solution : decay learning rate :

$\varepsilon_k = (1-\alpha)\varepsilon_0 + \alpha \varepsilon_\tau$ where $\alpha = \frac{k}{\tau}$

$\varepsilon_t \ll \varepsilon_0$ (typically $\varepsilon_\tau < 0.01 \varepsilon_0$)

k → no. of iterations, τ → total no. of iterations

→ $d\phi = \left(\frac{\partial \phi}{\partial x}\hat{i} + \frac{\partial \phi}{\partial y}\hat{j} + \frac{\partial \phi}{\partial z}\hat{k}\right) \cdot (dx\,\hat{i} + dy\,\hat{j} + dz\,\hat{k})$

$= \vec{\nabla}\phi \cdot \vec{n}$

→ $\vec{\nabla}\phi$ points to a direction ⊥ to the level surface

$d\phi = \vec{\nabla}\phi \cdot \vec{dr}$

$= |\vec{\nabla}\phi|\,|\vec{dr}| \cos\theta$

$|\vec{\nabla}\phi|$ → direction of steepest ascent

$\nabla\phi = |\vec{\nabla}\phi|\,\hat{N}$ (unit vector normal to surface)

* Handling oscillation : momentum-based technique : (Momentum and momentum)

↳ $v(t)$

$V(t) = \alpha V^{t-1} - \varepsilon g_t$

$\theta^{t+1} = \theta^t + V^t$

θ^0 → randomly initialized parameter

$V^0 \longrightarrow -\varepsilon g_t$

α → momentum parameter ($\alpha \in [0,1)$)

ε → learning rate

Terminal velocity : $\frac{\varepsilon \|g\|}{(1-\alpha)}$ is $(-g)$ direction

Nesterov : We have θ^t, we used $\theta^{t+1} = \theta^t + V_t$

Step V_t : $V^t = \alpha V^{t-1} - \varepsilon \nabla_\theta L(\theta_t + \alpha V_{t-1})$

Doing previous steps but while calculating gradients we add dV_i to them calculated.

$V_1 = -\varepsilon g_1$

$V_2 = -\varepsilon g_2 - \varepsilon g_1 \alpha$

$V_3 = -\varepsilon g_3 - \varepsilon g_2 \alpha - \varepsilon g_1 \alpha^2$

$\varepsilon - \varepsilon g_1 (1 + \alpha + \alpha^2)$

$$E(\xi_i^2) = V$$

Date
Page

→ If perfectly correlated, $C = V$

$$\Rightarrow mse = \frac{V}{K} - \frac{(K-1)V}{K} = V$$ (bagging doesn't h

Dropout:

→ dropping neurons of neural network (remove node & its incoming / outgoing connect

→ For each round, retain nodes with a fixed prob. (eg 0.5 for hidden nodes) and ($p = 0.8$) for visible (input) nodes.

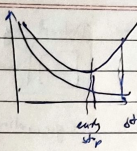
* In each round, take one training instance (mini-batch)
→ apply dropout
→ forward propogate, compute loss & back propogate
(update weights which were active in that round)

* At test time,
use full neural network & scale the output of each node by p (fraction of times it was 'on' during training)

Regularization

* Early stopping.

$$w_{t+1} = w_t - \eta \nabla w_t$$
$$= w_0 - \eta \sum_{i=1}^{t} \nabla w_i$$



let τ be the max value of ∇w_i

$$\Rightarrow w_{t+1} \leq w_0 - \eta t \tau$$

→ For early stopping, allow only 't' no. of updates

→ Early stopping is also related to L2 regularization.

Ensemble methods:

→ Combine output of different models to reduce test error

→ eg: Logistic reg., SVM, Naive bayes } take majority

→ Training can be done with diff. hyperparams, features, or diff. samples of training data

* Bagging:

→ ensemble method using diff. instances of same classifier

→ Construct multiple training sets by sampling with replacement ($T_1, T_2, \ldots T_k$)

→ Train i^{th} classifier with training set T_i

→ bother if the errors across classifiers are independent

Ex: K models of linear regression

Error for each model → ε_i

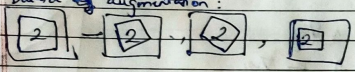
ε → zero mean multivariate normal distributed

$$E(\varepsilon_i^2) = V \qquad E(\varepsilon_i \varepsilon_j) = C$$

$$mse = E\left(\left(\frac{1}{K}\sum \varepsilon_i\right)^2\right) = \frac{V}{K} + \frac{(K-1)C}{K}$$

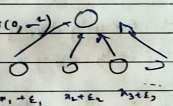
If no correlation, $mse = \frac{V}{K}$ (error reduced by factor of K)

→ Dataset eg augmentation :



(generating more training examples from existing one so that training model is more generalized.

* Adding noise to the inputs :



$\tilde{x}_i = x_i + \varepsilon_i$

$\hat{y} = \sum_{i=1}^{n} w_i x_i$ → predicted without noise

$\tilde{y} = \sum_{i=1}^{n} w_i \tilde{x}_i$ → with noise

$E(\varepsilon_i) = 0$

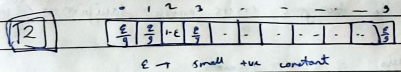
$E(\varepsilon_i \varepsilon_j) = \sigma^2$

$E(XY) = E(X) \cdot E(Y)$ if X & Y are independent

$$E[(\tilde{y} - y)^2] = \underbrace{E[(\hat{y} - y)^2]}_{\text{Mean squared error without noise. (training error)}} + \underbrace{\sigma^2 \sum_{i=1}^{n} w_i^2}_{\text{same as L2 norm penalty}}$$

∴ Adding noise to input is equivalent to L2 regularization for gaussian normal distribution

* Adding noise to outputs :



ε → small +ve constant

minimize : $\sum_{i=0}^{9} p_i \log q_i$ (Loss function)

true distribution + noise : $p = \{\frac{\varepsilon}{9}, \frac{\varepsilon}{9}, 1-\varepsilon, \frac{\varepsilon}{9}, \ldots\}$

estimated distribution q_θ

Now minimize.

Momentum based GD: overcomes obstacles, accelerates convergence, provides stability by maintaining a consistent direction & accumulating influence of past gradient.

→ Adagrad → adaptive hyperparameters: idea: decrease learning rate for params that have large gradients & increase it for params with low gradients.

RMS prop: same formula

$\theta^{t+1} = \theta^t - \frac{\eta g_t}{\sqrt{\varepsilon I + r_t}}$

$r_t = \beta r_{t-1} + (1-\beta) g_{t-1}^2$

$s_t = \sum_{i=1}^{t} g_i \odot g_i$

where $r_t = \sum_{i=1}^{t} g_i \odot g_i$

Problem: learning rate decreases continuously → can cause slow training

Convolutional Neural Network:

→ image classification

Problems with DFNN:

→ too many pixels (eg $10^3 \times 10^3$, each pixel → RGB (3 channels)) (too many inputs)

→ localized features (eg face) (spatial / temporal dependencies)

→ transformation shouldn't affect result (equivariance)

Image properties:

i) locality

ii) stationarity

iii) hierarchy (image is made up of multiple smaller sub-images)

Tasks in CNN:

i) Convolution layer:

→ Impose a kernel on the image & do element wise multiplication & then add

Image → N × N

Kernel → f × f

Resultant matrix → $(N-f+1) \times (N-f+1)$ $(N-f+1)(N-f+1)$

(when stride = 1)

If stride = S (move S-steps in one move)

then resultant matrix = $\left(\frac{N-f}{S}\right) + 1$

Image: $32 \times 32 \times 3$ (3 layers)

image size → 3 channels (RGB)

Kernel → eg one particular feature

dot product of image portion & kernel → how similarity b/w them

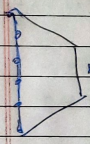
→ Uses multiple kernels for diff features.
→ predefined kernels
→ learned kernels

Advantages:

(i) → significantly reduces no. of ~~features~~ parameters

CNN → compromising the receptive filed of the a unit (neuron)

receptive field → No. of units in lower layer that affects the output of higher layer



Normal → N×K parameters

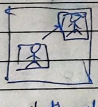
↓

f×f×K (using CNN)

→ To prevent loss, increase depth of neural network

(ii) → regions similar in values will output similar values (locality)

* Invariance:



$I \rightarrow C$

$\in R^{N \times N}$ $\in R^{(N-f+1)\times(N-f+1)}$

→ translational invariance → movement of objects within the image shouldn't change the convolutional output (C) much

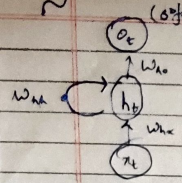
$f(g(I)) \approx f(I)$ f → convolution, g → translation/rotation/scale

* Equivariance: Apply f^n, then transform ≡ transform & then apply f^n

$f(g(I)) \approx g(f(I))$

Advantage: order of layering will not (Kernel 1 → Kernel 2 → Kernel 3) matter.

Even with varying lengths, the no of parameters to learn remains same.

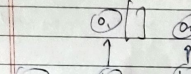


Forward prop:
start with random values of all parameters and h_0. Compute all the values

Backpropagation:
→ No. of steps needs to be decided

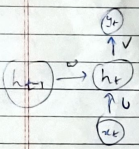
output is a vector

Loss function: $\frac{1}{(E_t)} \frac{1}{2} \sum_{k=1}^{out} (\hat{y}_k - y_k)^2$



Overall loss → $\frac{1}{2} \sum_{t=0}^{t} \sum_{k=0}^{out} (\hat{y}_{tk} - y_{tk})^2$

(2 steps)



ϕ → activation fn

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W^T \text{diag}(\phi'(h_{i-1}))$$

$$h_i = U x_t + W \phi(h_{i-1})$$

$$\frac{\partial h_i}{\partial h_{i-1}} = W^T \frac{\partial \phi(h_{i-1})}{\partial h_{i-1}}$$

$L = ab$

$$\frac{\partial L}{\partial x} = a^T \frac{\partial b}{\partial x} + b^T \frac{\partial a}{\partial x}$$

h_i → vector

Residual Network (resnet)
↳ can learn identity layer
↳ F(x) + x

Date
Page

— But as we keep on ↑ depth in ResNet,
after certain point — error increases due to overfitting

Recurrent Neural Network (RNN):

Problem with CNN

Words → Word2Vec → Vector (fixed length)

(similar words have similar vector)

I
want
to
IIT
in
2017

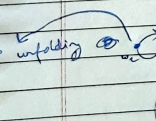
— 6×6 image
and then apply CNN

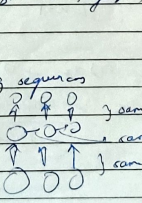
Problems:
— variable length std (image)
(height)
→ a word may be dependent on a word at a long distance. This may not be captured.
→ In 2013, I want to IIT Patna will generate different result.

Application of RNN :
i) input weather, output
ii) prose generation
iii) Sentiment Analysis
iv) text-to-speech
v) machine translation (language 1 ⇔ language 2)

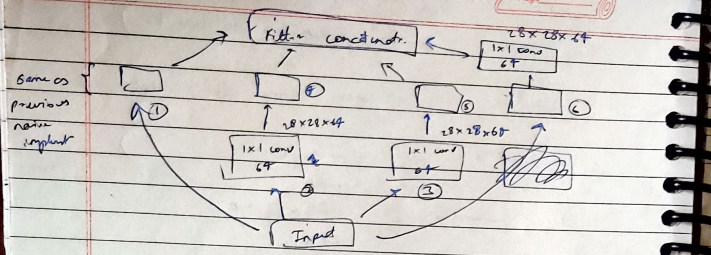
RNN :→
we share parameters across long sequences

• unfolding





→ Perceive the image at multiple scales parallelly.



~~Outputs: 28×28 (128+128+64+64 depth)~~

=

Computations: (reduced)

i) $28 \times 28 \times 256 \times 1 \times 1 \times 128$

ii) $\times 1 \times 1 \times 64$

iii) $\times 1 \times 1 \times 64$

iv) $28 \times 28 \times 64 \times 3 \times 3 \times 192$

v) $\times 5 \times 5 \times 96$

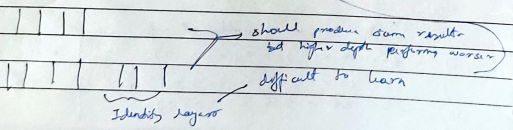
vi) $28 \times 28 \times 256 \times 1 \times 1 \times 64$

FLOPS = 358 Million

Why increasing depths may give more terror.
(both training & testing error)

↳ Not due to overfitting
as ⊗ higher training error would also be observed

→ Problem may be due to initialization. (of weights)



Deeper model → more non-linearity
→ increase receptive field
smaller filters → lesser parameters

Date
Page

$(n \times n)$ → n^2 parameters to learn

Vgg Net : (19 layers) (more depth, smaller filters)

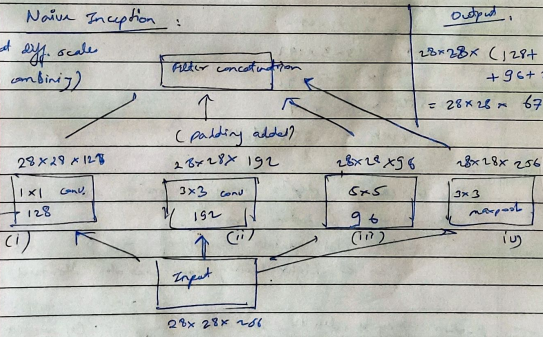
3 × 3 filters → Cno. of filters
→ 3 channels

No. of parameters = $3 \times 3 \times 3 \times C$
$= 27C$

→ Complexity of CNN's :
measured by the # of floating point operations (FLOPs)

Google Net :

→ Naive Inception :
(filtering at diff scales and then combining)



Output :
28×28×(128 + 192 + 96 + 256)
= 28×28×672

Consider the no. of multiplications :

i) $28 \times 28 \times 256 \times 1 \times 1 \times 128$
ii) $28 \times 28 \times 256 \times 3 \times 3 \times 192$
iii) $28 \times 28 \times 256 \times 5 \times 5 \times 96$
iv) ....

Total Flops = 854 Million

Universal Approximator:

1 hidden layer can represent any arbitrary f^n
but units may be exponential.

sol^n ← Increase depth

AlexNet:

Image → 227 × 227 × 3

Filter → (11 × 11 × 3)

↳ 96 filters

CONV1: stride = 4

$$\frac{227 - 11}{4} + 1 = 55$$

(55 × 55 × 96)

MaxPool: 3×3 filter of stride 2

$$= \left(\frac{55 - 3}{2}\right) + 1$$

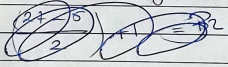
output: (27 × 27 × 96)

Same padding

Conv2: 5 × 5 filters → 256 filters

stride = 1

pad = 2



Output →

$$\frac{N + 2p - f}{s} + 1$$

$$= \frac{27 + 4 - 5}{1} + 1$$

$$= 27$$

Output → 27 × 27 × 256

Maxpool2: 3×3 filter, stride = 2

Output → $\left(\frac{27 - 3}{2} + 1\right) = 13 \times 13 \times 256$

$$\phi_{h_{i-1}} = \begin{bmatrix} \phi_{h_{i-1}}^{(1)} \\ \phi_{h_{i-1}}^{(2)} \\ \phi_{h_{i-1}}^{(3)} \\ \vdots \end{bmatrix} \qquad h_i = \begin{bmatrix} h_i^{(1)} \\ h_i^{(2)} \\ h_i^{(3)} \\ \vdots \\ h_i^{(n)} \end{bmatrix}$$

$$\frac{\partial \phi(h_{i-1})}{\partial h_i} = \begin{bmatrix} \frac{\partial \phi_{h_{i-1}}^{(1)}}{\partial h_{i-1}^{(1)}} & \frac{\partial \phi_{h_{i-1}}^{(1)}}{\partial h_{i-1}^{(2)}} & \cdots \\ \frac{\partial \phi_{h_{i-1}}^{(2)}}{\partial h_{i-1}^{(1)}} & \frac{\partial \phi_{h_{i-1}}^{(2)}}{\partial h_{i-1}^{(2)}} & \cdots \\ \frac{\partial \phi_{h_{i-1}}^{(3)}}{\partial h_{i-1}^{(1)}} & & \end{bmatrix}$$

$$= \begin{bmatrix} \phi'^{(1)}_{h_{i-1}} & 0 & 0 & \cdots \\ 0 & \phi'^{(2)}_{h_{i-1}} & & \\ 0 & & \ddots & \\ 0 & & & \\ \vdots & & & \end{bmatrix}$$

* Vanishing gradients : due to longer RNN chain

$$\prod \frac{\partial h_i}{\partial h_{i-1}} \quad (\text{all} < 1) \approx 0$$

(RNN stops learning)
(gradient = 0)

Sol^n:

i) divide chain into batches

(so a back propagation over a smaller no. of steps)

(gradient is calculated over a small batch only but propagation starts from beginning to end)

Paper: Attention is all you need

Attention → Transformers (parallelization) | RNN → GRU (memory)

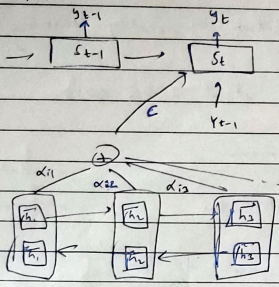
BERT: Bidirectional Encoder representation using Transformers → pre-trained models

Paper: Pre-training of Deep bidirectional Transformers for language Understanding.
— Jacob Devlin et. al

Auto encoders :

CNN, RNN, GRU, LSTM,
(+ bidirectional)
Attention & Transformers, Autoencoder

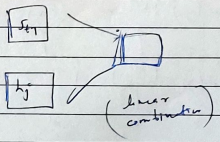
Attention based :



$$c^{(t)} = \sum_{j=1}^{T_x} \alpha_{tj} h_j$$

$$\alpha_{tj} = \frac{\exp(\tilde{\alpha}_{tj})}{\sum_{j=1}^{T_x} \exp(\tilde{\alpha}_{tj})}$$

$$\tilde{\alpha}_{tj} = f(s_{t-1}, h_j)$$



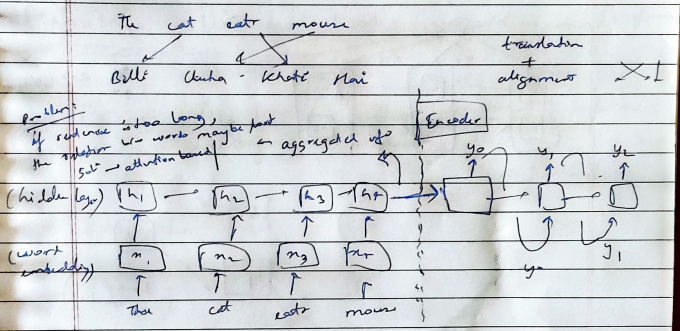
Attention and Transformers

Date 10/09/23
Page

→ Paper:

Neural machine translation by jointly learning to align and translate

— Bahdanau et al ICLR 2015



RNN based encoder:

Encoder: $x = (x_1, x_2, \ldots, x_{T_x})$

$h_t = f(h_{t-1}, x_t)$

Context vector: $\leftarrow c = g(h_1, h_2, \ldots h_{T_x})$

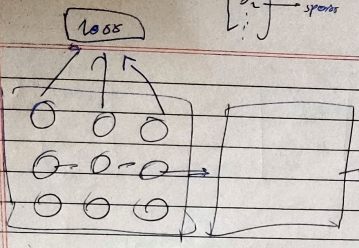
Decoder: $p(y) = \prod_{t=1}^{T_y} p(y_t | \{y_1, y_2 \ldots y_{t-1}\}, c)$

$p(y_t | \{y_1, y_2 \ldots y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$

eg: wikipedia article — classify the document

politics
sports

Date
Page



opposite of vanishing gradient
↓
exploding gradient (easier to handle)

$$h_t = \underset{n\times 1}{U x_t} + \underset{n\times 1}{W \phi(h_{t-1})}$$

New input $\longrightarrow$ $I = \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}_{2n\times 1}$ $\quad W_a = [\,W \;\; U\,]$

$$h_t = \phi\left(W_a \begin{bmatrix} h_{t-1}, x_t \end{bmatrix} + b_a\right)$$

Applicat^n of RNN :

i) Language generation & Sequence Model.

Suppose $y_1, y_2 \cdots, y_n$ be the words in a language

Goal of a language model :

→ Find the probability of a given sequence of words

$P(y_1, y_2, \cdots, y_5)$

→ Helpful in tasks like speech recognition