Machine Learning

Lect 1 : Introduction

Dr. PRERANA MUKHERJEE

Training samples ↓ ↓ ↓ features

$x_1 = \{ d_1, d_2 \ldots d_n \} \in \mathbb{R}^n$

$x_2 = \{ d_2, d_4 \ldots d_n \}$

$x_3 = \{ d_3, d_4 \ldots d_n \}$

$\vdots$

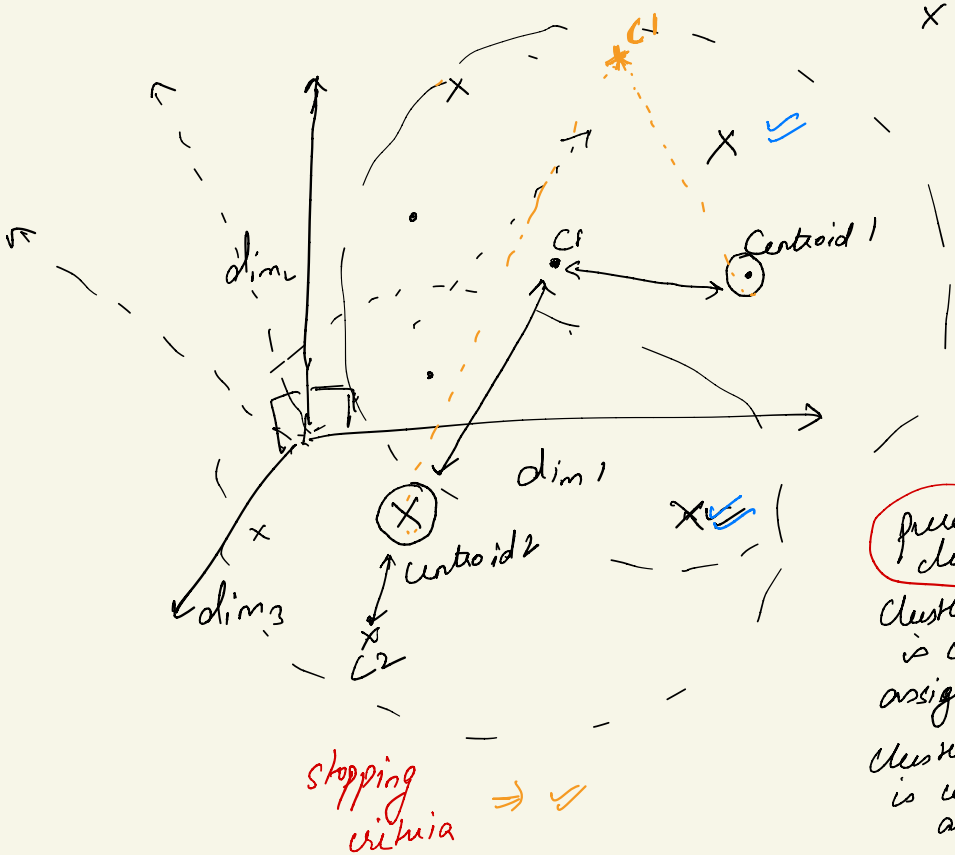
$x_n = \{ d_5, d_6 \ldots d_n \}$

wiki { Distance metrics

Unsupervised algorithms

k-means Input "k" = # clusters

• cluster 1
× cluster 2



stopping criteria ⇒

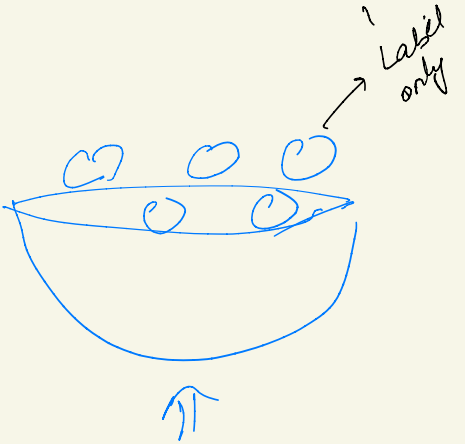
free clusters

cluster 1 is correctly assigned

cluster 2 is correctly assigned

Supervised Setting

Train a Classifier ⇒ SVM, kNN, Bayes, DNN (CNN, LSTM) . . .



- Apple (C1)
- Orange (C2)
- Pear (C3)
- Mango (C4)
- Banana (C5)

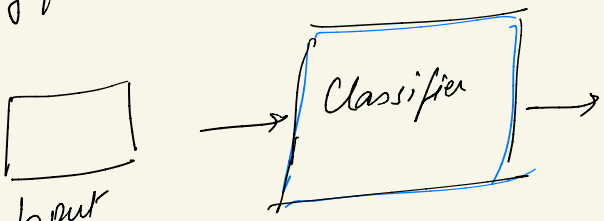
"5"

6th

Test data

Brain classifier ⇒ ANN

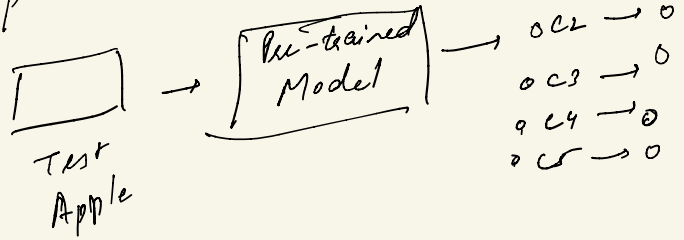
Training phase:



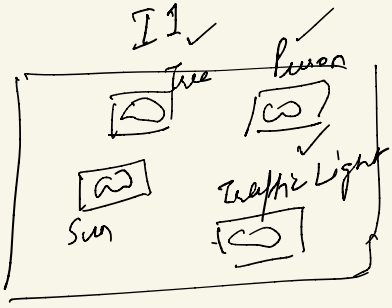
Input Visual feature

binary classifier 0/1

Test phase:



Test Apple



"multi-label classification"

Training phase ⇒ 4: person, tree, traffic light, vehicle

road scene

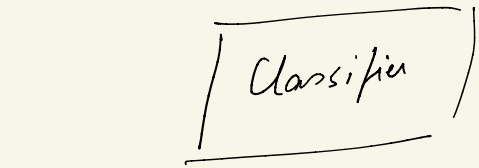
$\#n_1 \Rightarrow c_1$

$\#n_2 \Rightarrow c_2$

$\#n_3 \Rightarrow c_3$

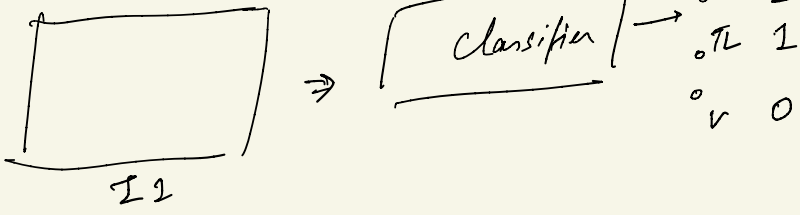
$\#n_4 \Rightarrow c_4$

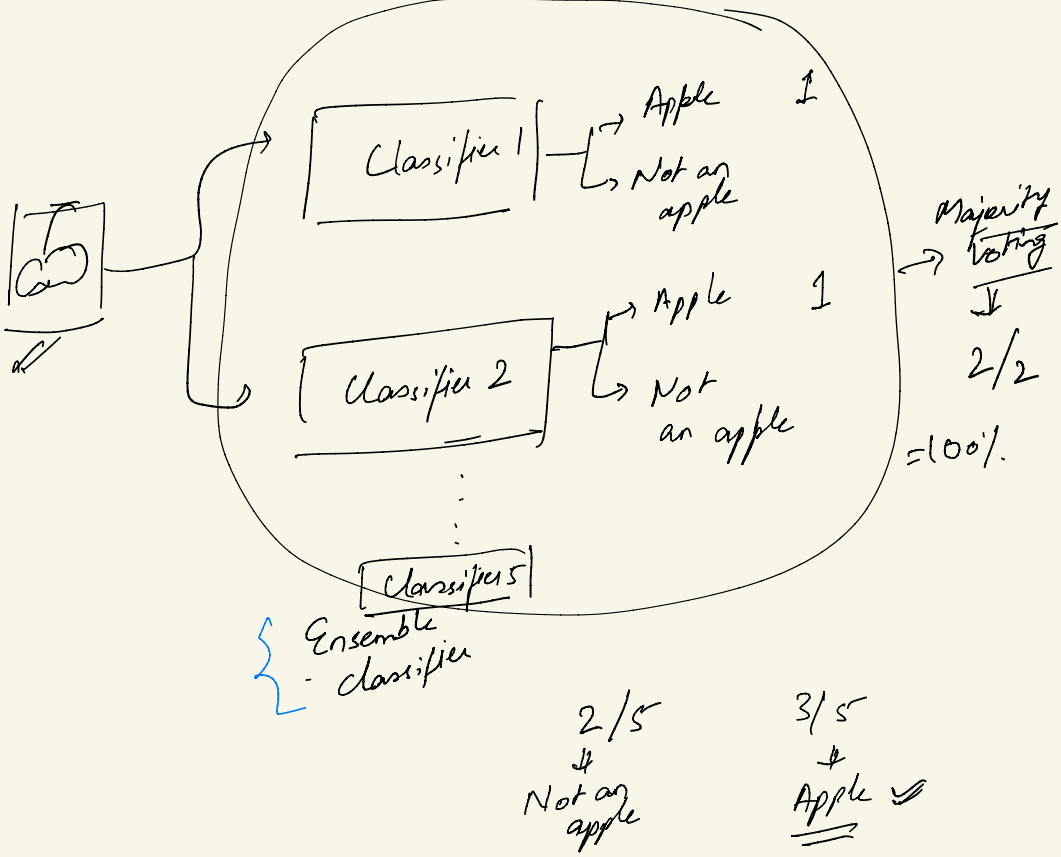
$\hat{C}$



Test phase ⇒

multi-label output



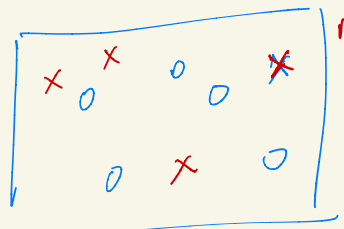


Semi supervised setting ⇒ EM, R.L. .. Reinforcement learning

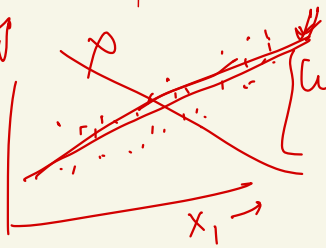
↓ expectation & maximization

→ Regression
- ↳ Linear
- ↳ Non-linear

→ Classification
- Unsupervised
- Supervised
- Semi-supervised



Curve fitting



Curve fitting
Line fitting

Purchase of Laptop

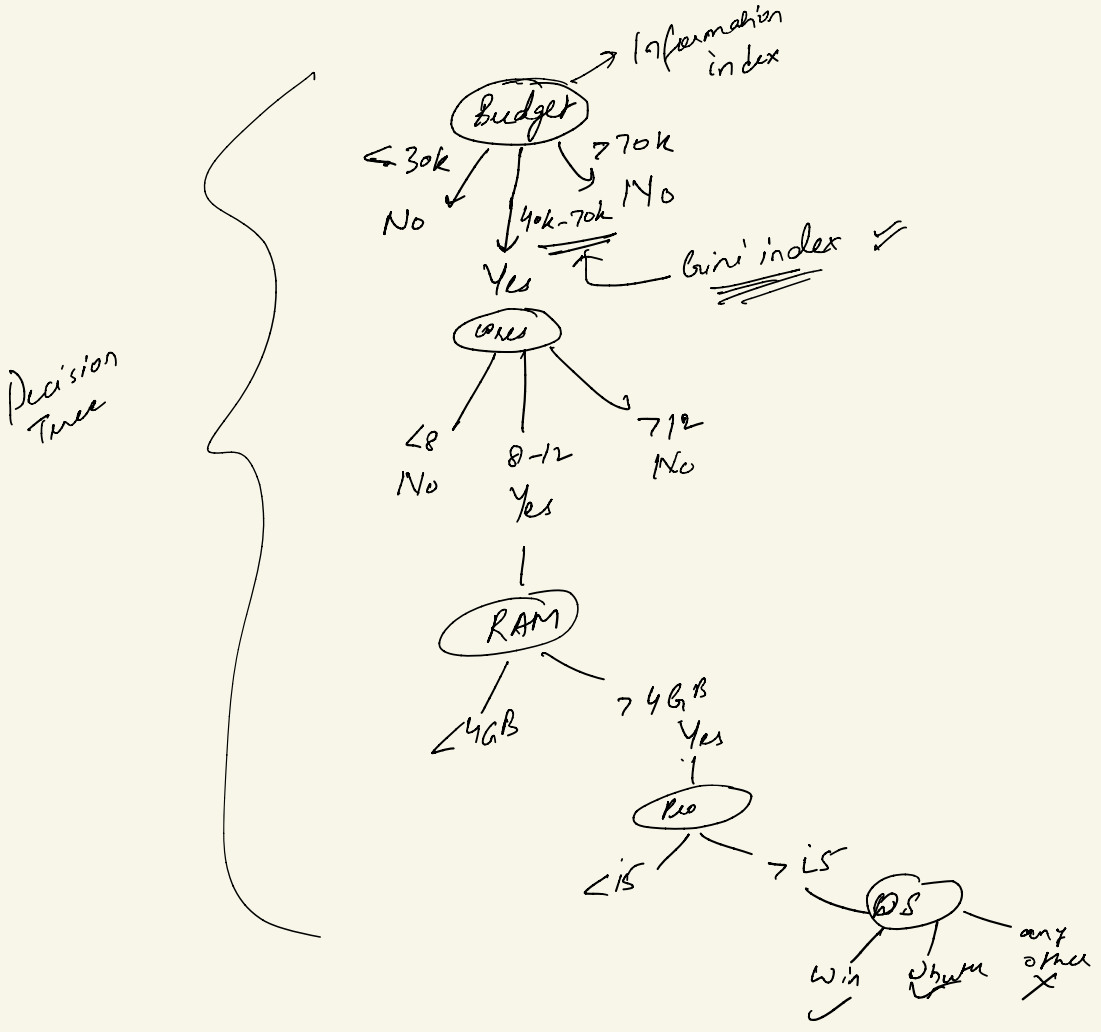
→ Budget (40k < INR < 70k)

→ # cores of Processor (CPU) (8 < # core < 12)

→ RAM (>4GB)

→ Processor generation (>i5)

→ Operating System (Windows 10, Ubuntu)



Machine Learning Lect-2

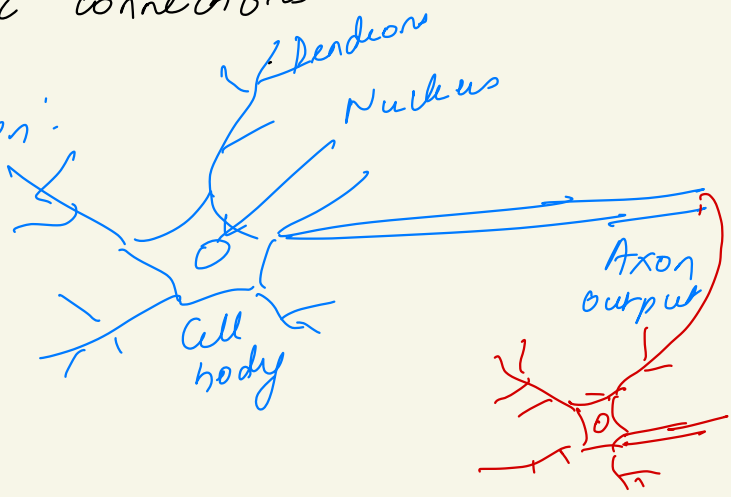
Introduction to Neural Networks

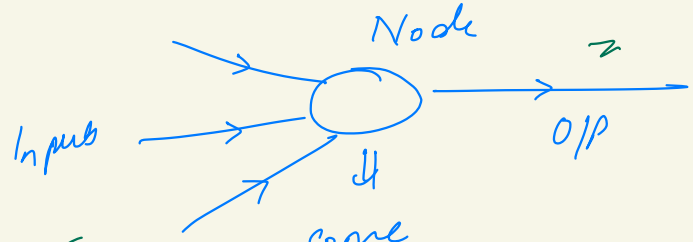
DR. PRERANA MUKHERJEE

Introduction

- Neural Networks
  - Inspired by how human brain works & maps the inputs to logical outputs.
  - Brain is the most complex part of human body & consists of 10 billion + neurons
  - Neurons are highly interconnected & act as individual units.
  - Characteristics of neurons:
    - Massive parallelism
    - Highly interconnected (solve the complex problems of your real world)
    - Model distributive associative memory through weights in the synaptic connections

Diagram of neuron:





Structure of neuron simulated by NN

some simple computations weighted sum of inputs & then applies some squashing/ activation function (could be sigmoid or other)

ANNs through synaptic connections O/p one neuron input using electric impulses through the synaptic layers.

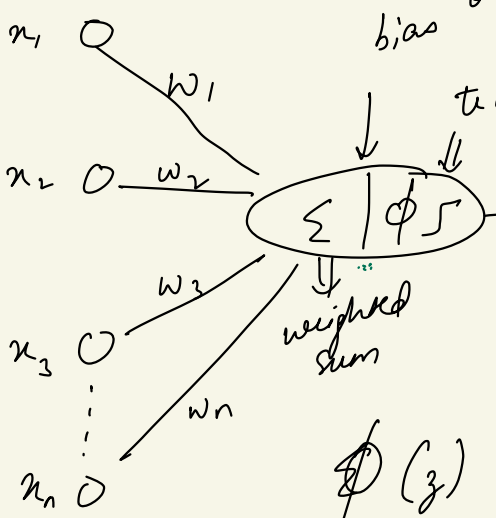
Incorporate the 2 fundamental components of the biological neural nets.

- Nodes - Neurons
- Weights - synapses

Basic unit in NN: Perceptron

Perceptron

Inputs : n



Training Data

x	y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_m	y_m

Training the n/w means learning the weights to have good fit for training data.

$\in \mathbb{R}^d$

$\phi(z) = z$ (identity)

$\phi(z) =$ (unit step)

$\phi(z) =$ (sigmoid)

$$y = \sum_{i=1}^{n} w_i x_i + b \quad \text{or} \quad \sum_{i=0}^{n} w_i x_i$$

bias

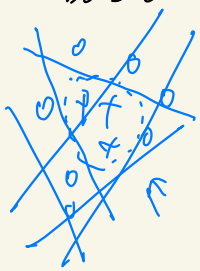
ϕ is identity function $\Rightarrow$ same passed as output

Or

$\phi_2(z) =$ (threshold function)

Perceptron training rules

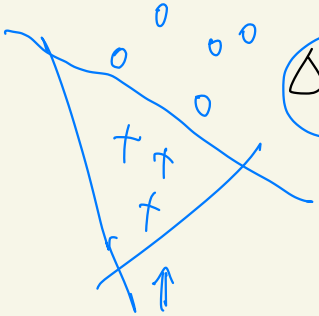
- we feed 1 eg at a time & update the weights based on output.



if output (predicted) == output (target) ⇒ no update in weights required.

$$w_i^{new} = w_i^{prev} + \Delta w_i$$

$$\Delta w_i = \eta \left(y - \hat{y}\right) x_i$$

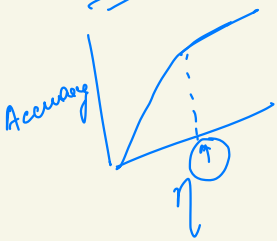


Learning rate — based on how fast you change weights

Target

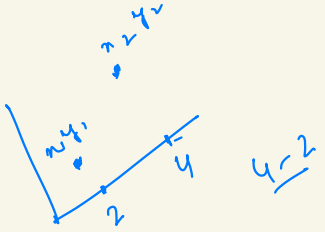
Predicted

Grid search



Stopping criteria ⇒ Perceptron learning convergence if D is linearly separable. (training data) & linear decision surface to separate +/-

What if data is not linearly separable ??



Solution : Gradient descent

can be used when wrt to particular parameter set / weight values you can define the error of the n/w. Perform GD on error func. to find the optimal parameter set for which the error fn. is minimized

$E := \frac{1}{2} \sum_{d \in D} (y_d - \hat{y}_d)^2$ ⇒ is like a surface & we want to find minima of this surface

½ so that differentiation is easy

Find partial derivative of error fn. with each & every weight and find the direction of the gradient & we go towards the -ve direction of gradient in order to go towards minima.

In certain cases, this error surface can be convex or quadratic & hence we'll be single minima ⇒ thus gradient descent guarantees the minima.

Multilayer network error surface will be ill-behaved (non-convex)
↑
there will be local minima
↓
get stuck

$\phi_1(z) =$ step function → is not differentiable ∴ so gradient descent can't be done

$\phi_1(z) = z$ is differentiable ⇒ gradient descent can be done

Stochastic gradient descent

1 eg. from training set at a time

$$E = \frac{1}{2}(y_d - \hat{y}_d)^2$$

Training rule of G.D

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = x_{ij}(y_j - \hat{y}_j)$$

Single layer n/w capture linear decisions / linear separable data.

Capture non-linear functions or complex ⇒ multi-layer n/ws

→ different n/ws → if linear units are connected with each other

combination is again linear fn.

$$\phi_3(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Assignment love it →

$$\phi_3'(z) = \phi_3(z) \cdot (1 - \phi_3(z)) \quad (*)$$

$$E = \frac{1}{2} \sum_{d \in D} \left(y_d - \underbrace{\sigma(w \cdot x_d)}_{\hat{y}_d} \right)^2$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_d \frac{\partial E}{\partial \hat{y}_d} \cdot \frac{\partial \hat{y}_d}{\partial w_i}$$

$$= \sum_d (y_d - \hat{y}_d) \cdot$$

$$\frac{\partial (y_d - \sigma(w x_d))}{\partial w_i}$$

$$\sigma'(w \cdot x_d) \cdot x_{id}$$

$$= \sum_d (y_d - \hat{y}_d) x_{id} \cdot$$

$$\sigma'(w \cdot x_d) \quad \text{logistic} \quad (1)$$

$\frac{\partial E}{\partial w_i} = \sum_d (y_d - \hat{y}_d) \cdot \sigma'(w \cdot x_d) \cdot x_{id}$

to represent complex functions we want non-linear units (which are differentiable) ⇓ eg sigmoid/ transfer function

Using (*) in (1)

$$\frac{\partial E}{\partial w_i} = \sum_d (y_d - \hat{y}_d)\, x_{id}\, \hat{y}_d (1 - \hat{y}_d)$$

$$\Delta w_i = -\eta \sum_d (y_d - \hat{y}_d)\, \hat{y}_d (1 - \hat{y}_d)\, x_{id}$$

↑ training rule of sigmoid unit

↑↑ single layered NN ⇒ for linearly separable functn.

Machine Learning Lect 3

Gradient Descent

Dr. Purnendu Mukherjee

9 Aug '20

Ques AND function implementation using Single Layer Perceptron

Truth table of AND

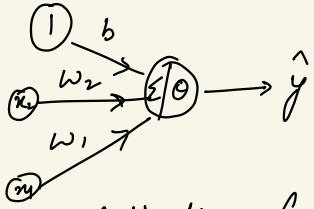
x_1	x_2	t
1	1	1
1	0	0
0	1	0
0	0	0

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

0 → -1 (Label)

1 → 1 (Label)

Perceptron N/w



$y_{in} = \sum_{i=0}^{n} w_i x_i$

Error $= t - \hat{y}$

Activation function $\theta = 0$

$$\theta(y_{in}) = \begin{cases} 1 & y_{in} > 0 \\ 0 & y_{in} = 0 \\ -1 & y_{in} < 0 \end{cases}$$

η = $\alpha = 1$ some value b/w 0 to 1

Epoch:

Initialize weights = 0

$w_1, w_2, b = 0$

i)

$y_{in} = \sum w_i x_i + b$

$= b + w_1 x_1 + w_2 x_2 = 0 + 0(1) + 0(1)$

$= 0$

$\hat{y} = \theta(y_{in}) = 0$

$\hat{y} \neq t \Rightarrow$ update weights

$w_i(new) = w_i(old) + \alpha t x_i$

$\Delta w = \eta (t - \hat{y}) x_i$

$w_{1new} = w_{1old} + \alpha t x_1$

$= 0 + 1 \cdot 1 \cdot 1 = 1$

$w_{2new} = w_{2old} + \alpha t x_2$

$= 0 + 1 \cdot 1 \cdot 1 = 1$

$b_{new} = b_{old} + \alpha t$

$= 0 + 1 \cdot 1 = 1$

ii) $y_{in} = b + w_1 x_1 + w_2 x_2$

$= 1 + (1)(1) + (1)(-1)$

$= 1 + 1 - 1 = 1$

$\hat{y} = \theta(y_{in}) = 1$

$\hat{y} \neq t \Rightarrow$ update weights

$w_{1new} = w_{1old} + \alpha (t - \hat{y}) x_1$

$= 1 + 1(-1 - 1) \cdot 1$

$= 1 - 2 = -1$

$w_{2new} = w_{2old} + \alpha (t - \hat{y}) x_2$

$= 1 + 1(-1 - 1) - 1$

$= 1 + 2 = 3$

$b_{new} = b_{old} + \alpha (t - \hat{y})$

$= 1 + 1(-2) = -1$

iii) $y_{in} = b + w_1 x_1 + w_2 x_2$

$= -1 + (-1)(-1) + (3)(1)$

$= -1 + 1 + 3 = 3$

$\hat{y} = \theta(y_{in}) = 1$

$\hat{y} \neq t \Rightarrow$ update weights

$$w_{1,new} = w_{1,old} + \alpha (t - \hat{y}) x_1$$

$$w_{1,new} = -1 + 1(-1 - (-1)) - 1$$

$$= -1 + 2 = 1$$

$$w_{2new} = w_{2old} + \alpha (t - \hat{y}) x_2$$

$$= 3 + 1(-1 - 1) 1$$

$$= 3 - 2 = 1$$

$$b_{new} = -1 + 1(-2) = -1 - 2$$

$$= -3$$

(iv) $y_{in} = -3 + (1)(-1) + 1(-1)$

$$= -3 - 1 - 1 = -5$$

$\hat{y} = -1$

$\hat{y} = t \Rightarrow$ no change in weight

Final weights are

$w_1 = 1 \quad w_2 = 1 \quad b = -3$

Decision boundary

$$b + w_1 x_1 + w_2 x_2 = 0$$

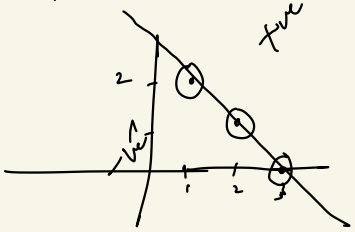
$$-3 + x_1 + x_2 = 0$$

$$x_1 + x_2 = 3$$

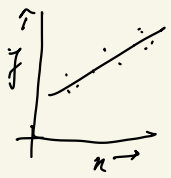
$x_1 = 0 \quad x_2 = 3$

$x_1 = 1 \quad x_2 = 2$

$x_1 = 2 \quad x_2 = 1$



Gradient Descent

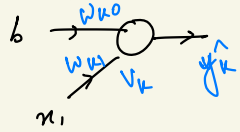


$y = f(x)$

fit a straight line given a set of observations

Linear Regression $y = f(x)$

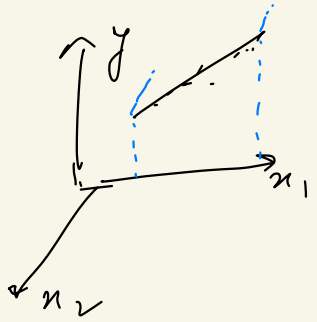
Single neuron which works on linear activation unit $v_k = \sum_{j=0} w_{kj} x_j = \hat{y_k}$

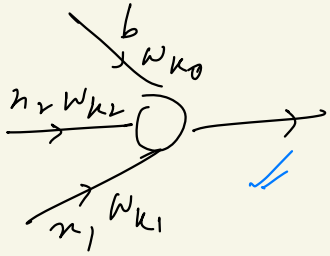


2 parameters: slope & intercept

$y = f(x_1, x_2)$

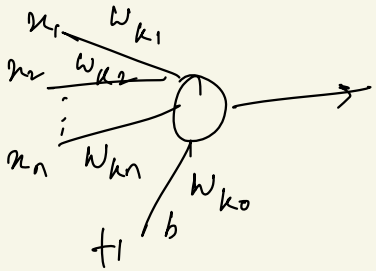
Line fitting → 3D line fitting





2 slope components along $x_1 y$ plane w_{k1}, $x_2 y$ plane w_{k2} + 1 intercept

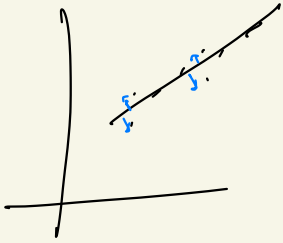
$y = f(x_1, x_2 \ldots x_n)$



$y = \sum w_{kj} x_j$

n-d straight line

So many gradients along individual planes + bias



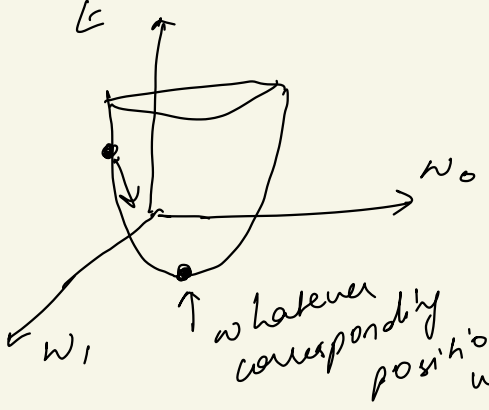
not passing exactly through points

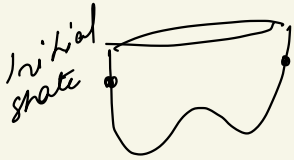
Error values some positive & some negative ⇒ combined error can be plot as a function of 2 variables w_1 & w_0

slope intercept

natural tendency is to reach the minima (minimum error)

fit a curve which searching all over the plane gives min error for best combination of parameters $(w_1, \& w_0)$





These 2 minimas could be of different magnitudes out of the several minimas the surface can exhibit.

Only 1 global minima rest are local minimas.

System adopts & learns to reach best fitment & in the process it reaches global minima or gets trapped at local minima.

Gradient point up but the point moves opposite to the direction of the gradient.

Let us have p : no. of observations

$$E = \sum_p E^p$$

$$E^p = \frac{1}{2} \sum_{o=0} \left(t_o^p - y_o^p \right)^2 \quad \text{— (1)}$$

no of different O/Ps

t_o^p ← Desired, y_o^p ← Actual

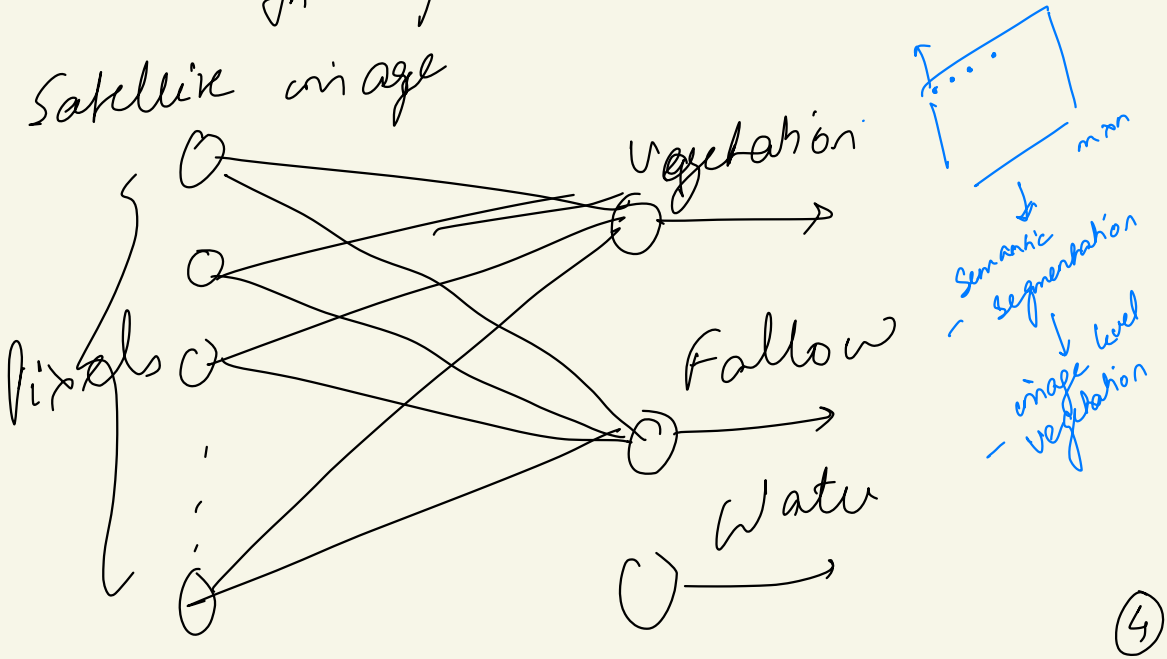
Industrial output

$$y_0 = f_1(x_1, x_2 \ldots x_n)$$

$$y_1 = f_2(x_1, x_2 \ldots x_n)$$

$$\vdots$$

$$y_n = f_n(x_1, x_2, \ldots x_n)$$



$$G = \frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_p E^p$$

$$= \sum_p \frac{\partial E^p}{\partial w_{ij}}$$

Chain rule

$$\frac{\partial E}{\partial w_{oi}} = \frac{\partial E}{\partial y_o} \cdot \frac{\partial y_o}{\partial w_{oi}}$$

From (1)

$$\frac{\partial E}{\partial y_o} = -(t_o - y_o) \quad \text{--- (2)}$$

$$y_o = \sum_j w_{oj} x_j \quad (5)$$

$w_1 x_1 + w_2 x_2 + \cdots w_{0i} x_i + \cdots w_n x_n$

$$\frac{\partial y_0}{\partial w_{0i}} = \frac{\partial}{\partial w_{0i}} \sum_j w_{0j} x_j$$

$$= x_i \quad -(3)$$

$$\frac{\partial E}{\partial w_{0i}} = -\frac{(t_0 - y_0)\, x_i}{w.r.t\ 1\ \text{particular weight}}$$

Error term

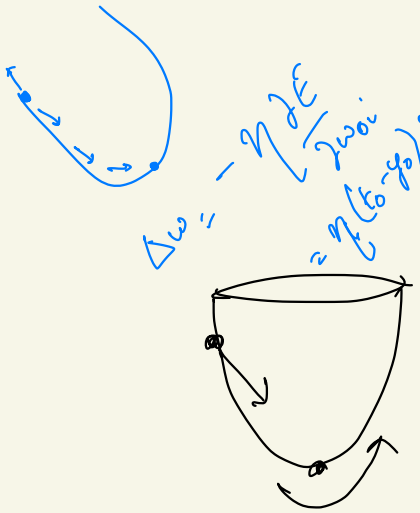
$(-\eta(-)) \Rightarrow +ve$

$$\Delta w_{0i} = \eta (t_0 - y_0)\, x_i$$

rate at which it is falling

$$w_{0i} = w_{di} + \Delta w_{0i}$$

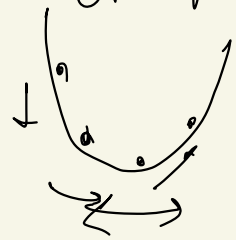
$$\Delta w = -\eta \frac{\partial E}{\partial w_{0i}} = \eta (t_0 - y_0).x_i$$



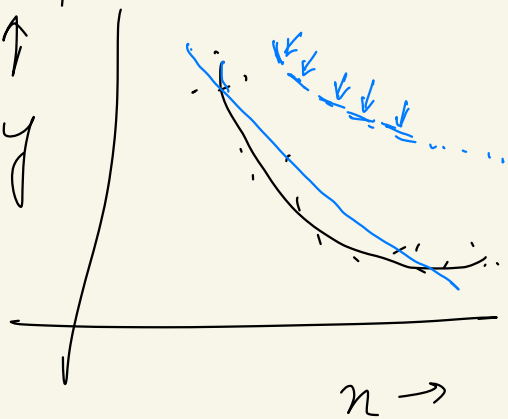
$\eta \Rightarrow$ high it is fast (might cross the minima)
else slower

Linear n-d straight line fitting

neural network → linear activation fn



oscillate to reach minima



non-linear curve fitting

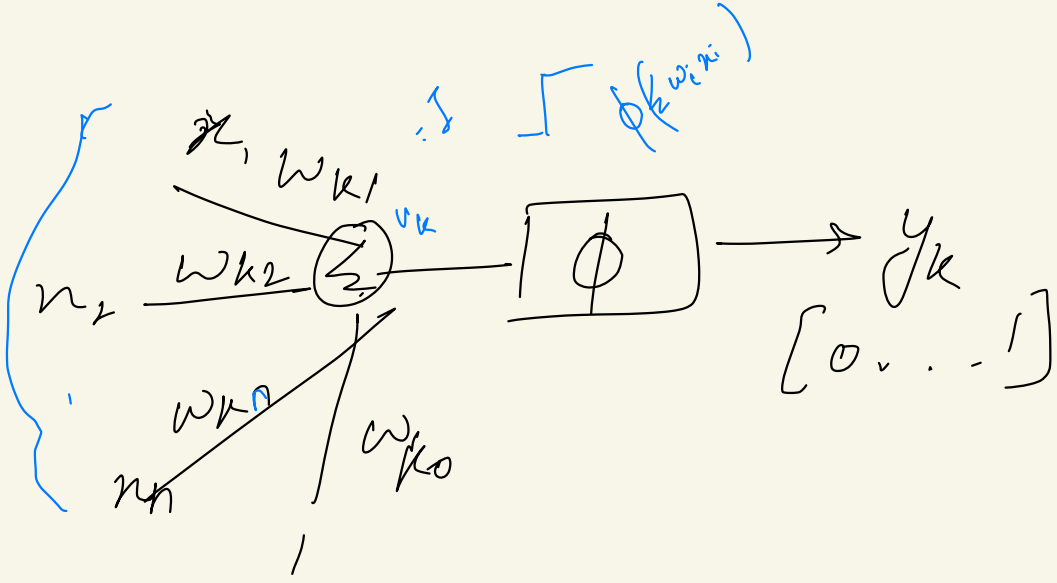
↓

break it into piecewise linear components

↓

complicated process

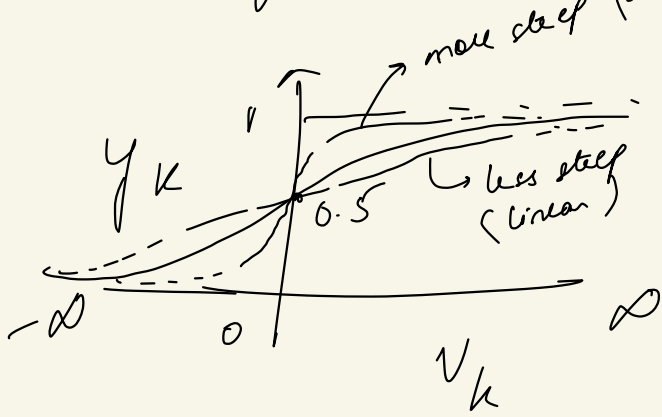
Soln: non-linear model



$$v_k = \sum_{j=0}^{n} x_j \cdot w_{kj}$$

$$y_k = \phi(v_k)$$

$y = f(x)$ iff $x \uparrow$ $y \uparrow$



i) monotonically increasing

ii) continuously differentiable

fn: not continuous ⇒ not differentiable

Mc Culloch & Pitts n.

Tunability : shape control

$$\phi(v_k) = \frac{1}{1 + e^{-a v_k}} \quad \text{(Sigmoid)}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Quotient rule:

$$\sigma'(z) = \frac{(1+e^{-z}) \cdot 0 - (1)(-e^{-z})}{(1+e^{-z})^2}$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} = \frac{1 - 1 + e^{-z}}{(1+e^{-z})^2}$$

$$= \frac{1+e^{-z}}{(1+e^{-z})^2} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} - \frac{1}{(1+e^{-z})^2}$$

$$= \frac{1}{(1+e^{-z})} \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= \sigma(z)\,(1 - \sigma(z))$$

Machine Learning Lect 5 (42)

Multi Layer Neural Network

Dr. Purnima Mukherjee

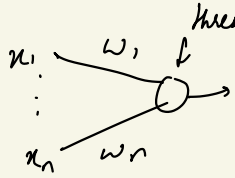
Multilayer Neural N/w

Single layer NN can represent the linear functions but in order to represent non-linear functions we require multilayer NN which can be achieved by stacking perceptrons into different architectures

Feed forward NN.

Limitations in Perceptron:

i) They exhibit the monotonic behaviour



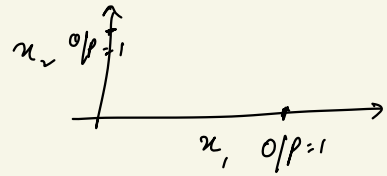
ii) if link has +ve weight activation can only increase as input values increases.

(It cannot represent functions where input interactions can cancel each other.

iii) Logic gates like AND can be represented as it is monotonic function o/p is 1 when $x_1 = 1, x_2 = 1$

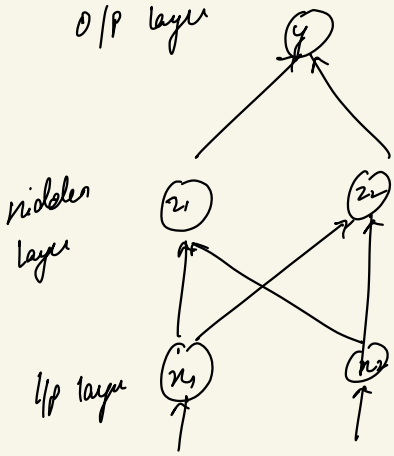
but XOR cannot be represented by SLP as it is not a monotonic property

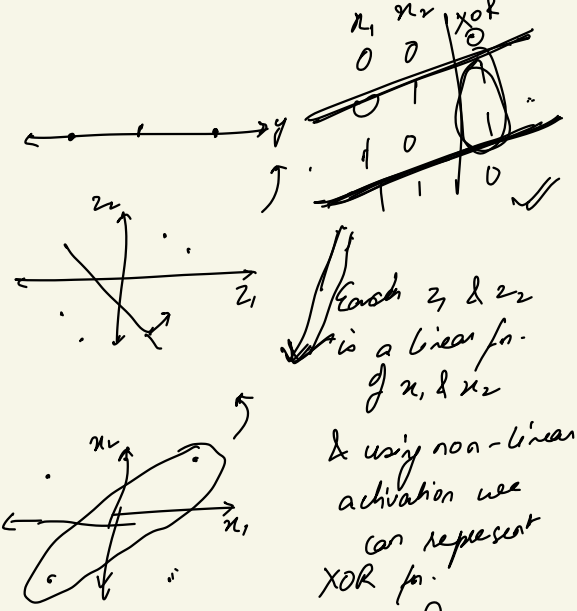
Can't represent functions where input interactions can cancel each other



Sol: Multilayer NN.

iv) can represent only linear separable functions





Each z_1 & z_2 is a linear fn. of x_1 & x_2

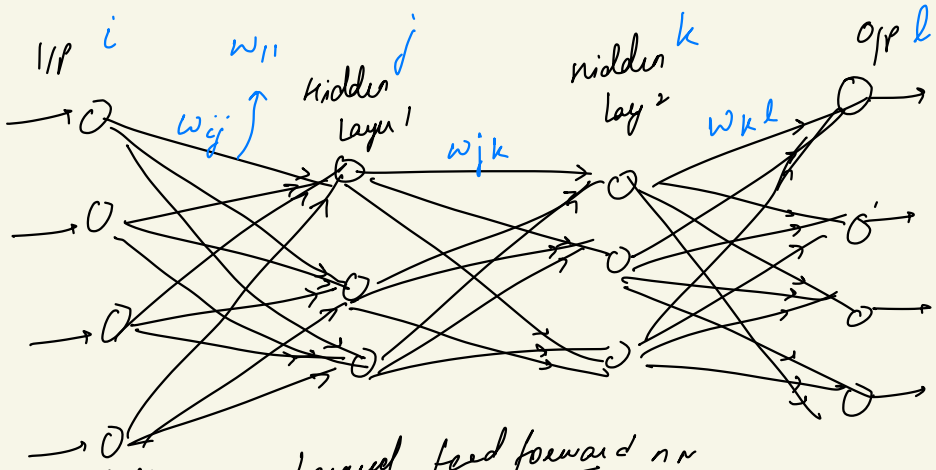
& using non-linear activation we can represent XOR fn.

Why hidden units : because in training we can't observe them & using them we can represent many non-linear functions

Power / Expressiveness of MLP NN

- Can represent interactions among inputs
- Two layer inputs can represent any Boolean function & continuous functions (within a tolerance) as long as the number of hidden units is sufficient & appropriate activation functions used
- Learning algorithm exists, but weaker guarantees than perceptron learning algorithms.

→ Three layer NN can represent all computable functions as they have good representational capability.



Fully connected

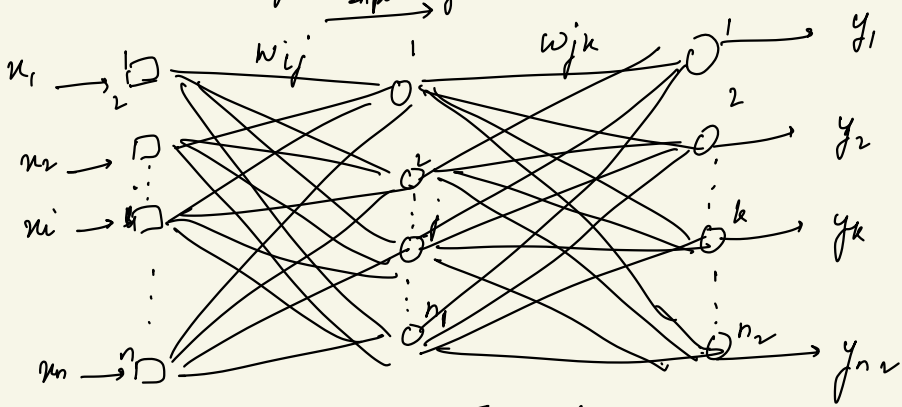
Layered feed forward nn

organized the neurons into layers

layer i → layer i+1

as there is no back links

Two layer back propagation NN.



$\Delta w =$

Perception ; if there is change is o/p (error) it is difference b/w ideal & obtained o/p

training rule

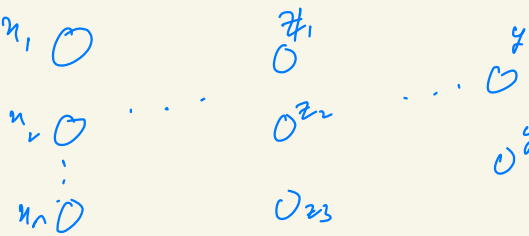
you change the weights

It can be achieved in SLP

But there is problem in MLP ??

We do not know the ideal output in hidden units to compute the change in weights (w_{ij})

Solution: Error observed at O/P layer is propagated backwards.



Error at y_1 is backpropagated to z_1, z_2 & z_3

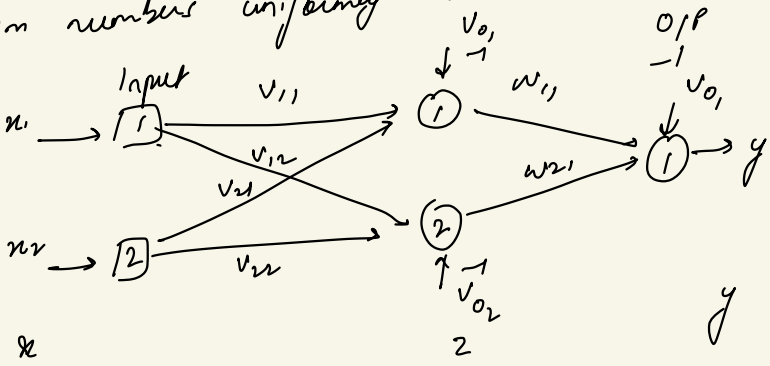
Error at y_2 is also backpropagated to z_1, z_2 & z_3

& thus we get the notional error & then we can update the weights

Backpropagation training algo.

Step 1 : Initialization

Set all weights & threshold levels of the network to random numbers uniformly distributed inside a small range.



Step 2 Forward Computing

- Apply an input vector $\bar{x}$ to input units
- Compute the activation / output at the hidden layer

$z_j = \phi\left(\sum_i v_{ij}\, x_i\right)$

- Compute the output vector $\bar{y}$ at output layer

$y_k = \phi\left(\sum_j w_{jk}\, z_j\right)$

$\bar{y}$ is the result of computation

Training examples

$\bar{x}_1$	$\bar{y}_1$	$\hat{y}_1$
$\bar{x}_2$	$\bar{y}_2$	$\hat{y}_2$
⋮	⋮	
$\bar{x}_m$	$\bar{y}_m$	$\hat{y}_m$

Error ⇒ based on this change weights

Step 3 Learning for BP Net

- Update of weights in W (between o/p & hidden layer)
  - delta rule
- Not applicable to update V (b/w input & hidden)
  - don't know the target values of hidden units $z_1, z_2 \ldots z_p$

Solution: Propagate error at o/p units to hidden units to drive the update of weights in V (again by delta rule)

(error back-propagation learning)

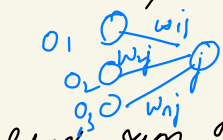
- Error backpropagation can be continued downward if net has more weight than one hidden layer
- How to compute errors on hidden units?

For 1 output neuron, error fn. is

$$E = \frac{1}{2}\sum (y - \hat{y})^2$$

For each unit j, the output o_j is defined as,

$$o_j = \phi(net_j) = \phi\left(\sum_{k=1}^{n} w_{kj}\, o_k\right)$$

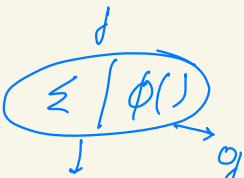


The input net$_j$ to a neuron is the weighted sum of outputs o_k of previous n neurons.

Find derivative of the error:

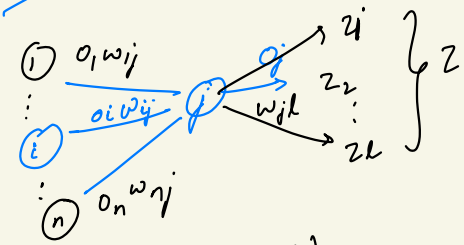
Backpropagation learning rule

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}} \quad -(*)$$



$w_1 o_1 + w_2 o_2 + \cdots$

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}\, o_k\right) = o_i \quad -(1)$$

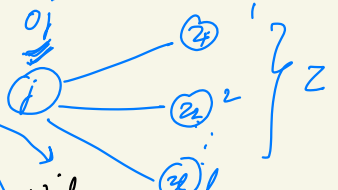


$\phi'(z) = \phi(z)(1 - \phi(z))$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j}\,\phi(net_j) = \phi(net_j)\left(1 - \phi(net_j)\right) \quad -(2)$$

Depends on output activation fn. (Let's assume sigmoid)

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(1, 2, \ldots l)}{\partial o_j}$$



$net_k = \sum w_{jk}\, o_j = w_1 o_1 + w_2 o_2 + \cdots$

$$= \sum_k \left(\frac{\partial E}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j}\right) = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial net_k} \cdot w_{jk} \quad -(3)$$

$$\frac{\partial E}{\partial w_{ij}} = \delta_j \cdot o_i$$

δ_j → Errors computed at j^{th} Layer which is backpropagated at j

$$\delta_j = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\begin{cases} (o_j - t_j)\, o_j\, (1 - o_j) & \Rightarrow \text{if it is an output} \\ \sum_l (\delta_l w_{jl})\, o_j\, (1 - o_j) & \Rightarrow \text{if there is some intermediate unit} \end{cases}$$

3 ✓✓
- ✓ Perceptron training rule
- ✓ Gradient training rule
- ✓✓ Backpropagation training rule

NN ✓✓

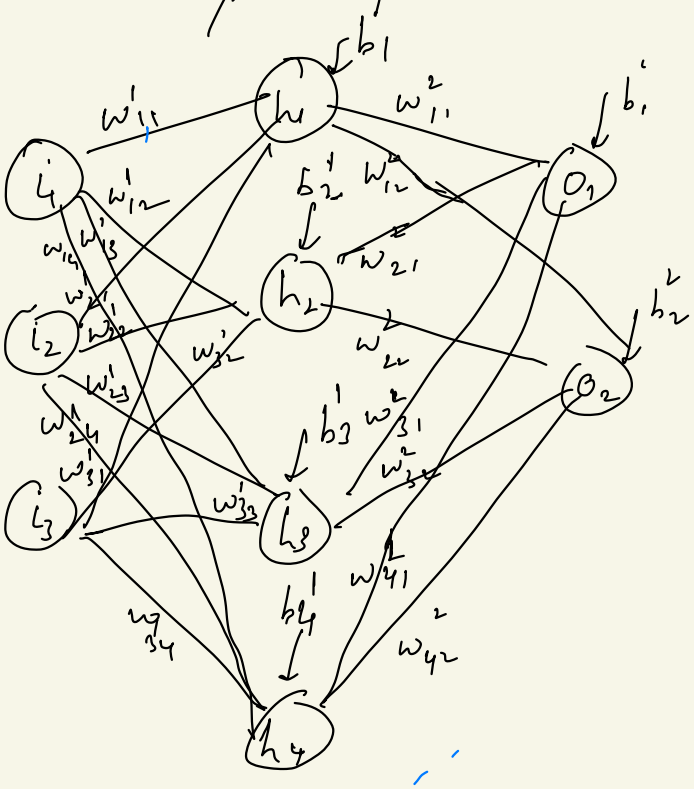
Machine Learning

Lect. 6

Dr. Gaurav Mukherjee

Multi Layer Perceptron

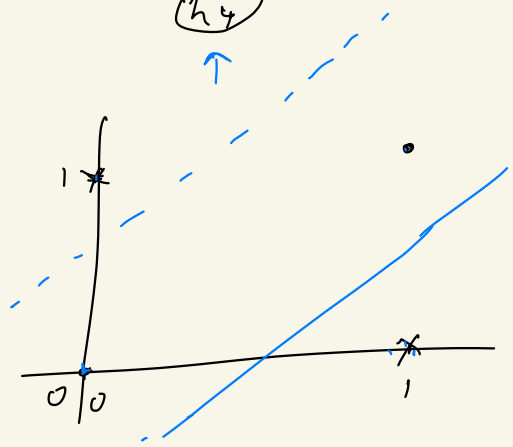
XOR problem



$$h_n = a\left(\sum_{k=1}^{3} i_k w_{kn} + b_n\right)$$

$a(x)$ = sigmoid $(x) = \frac{1}{1+e^{-x}}$

Universal approximation theorem

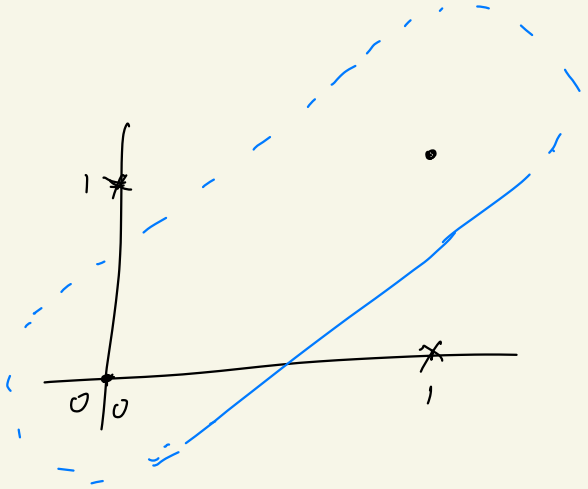


x_1	x_2	O/P y
0	0	0
0	1	1
1	0	1
1	1	0

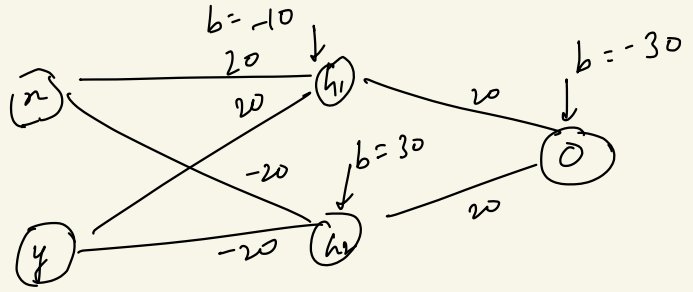
How to seperate this with a linear boundary

↓

Perceptron can't solve this but MLP can!



⇒ Non linear seperation

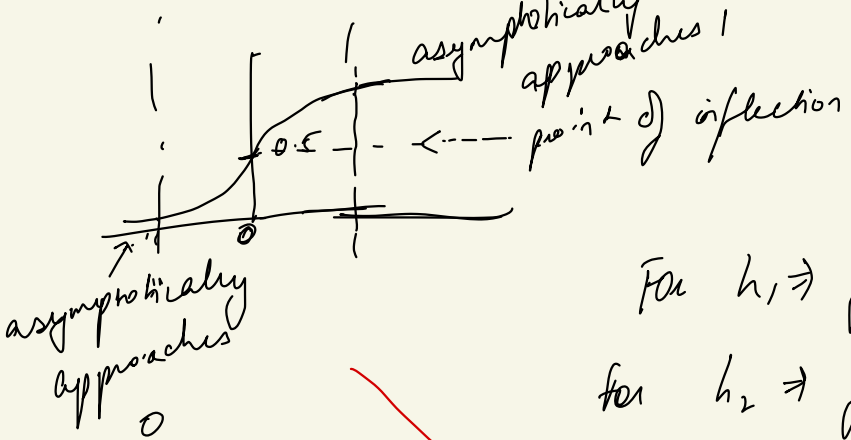


x_1	x_2	O/P $\hat{y}$
0	0	0.000045
0	1	0.99995
1	0	0.9995
1	1	0.000045

$h_1 = \text{sigmoid}(20x + 20y - 10) \approx 0$

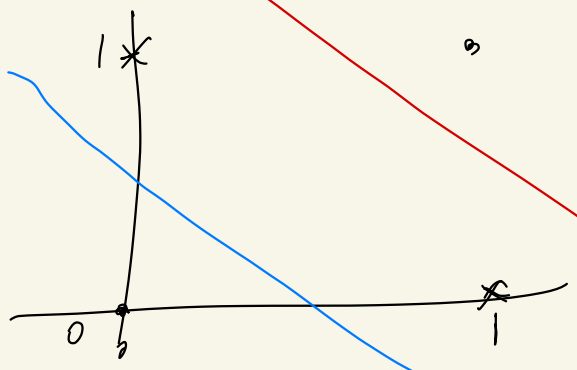
$h_2 = \text{sigmoid}(-20x - 20y + 30) \approx 0$

$0 = 20h_1 + 20h_2 - 30$



For $h_1 \Rightarrow y \simeq 0.5 - x$

for $h_2 \Rightarrow y \simeq 1.5 - x$



with only these given set of weights you can learn such complex XOR functions

But how to learn these weights?

↓

Sol: Backpropagation

$\sigma(20\times0+20\times0-10)\approx0$ $\sigma(-20\times0-20\times0+30)\approx1$ $\sigma(20\times0+20\times1-30)\approx0$

$\sigma(20\times1+20\times1-10)\approx1$ $\sigma(-20\times1-20\times1+30)\approx0$ $\sigma(20\times1+20\times0-30)\approx0$

$\sigma(20\times0+20\times1-10)\approx1$ $\sigma(-20\times0-20\times1+30)\approx1$ $\sigma(20\times1+20\times1-30)\approx1$

$\sigma(20\times1+20\times0-10)\approx1$ $\sigma(-20\times1-20\times0+30)\approx1$ $\sigma(20\times1+20\times1-30)\approx1$

Machine Learning

Lect 8

Dr. Pranav
Mukherjee

SVM : Support Vector Machines : hyperplane based classifiers

Vapnik Chervonenkis

['VC' dimension]

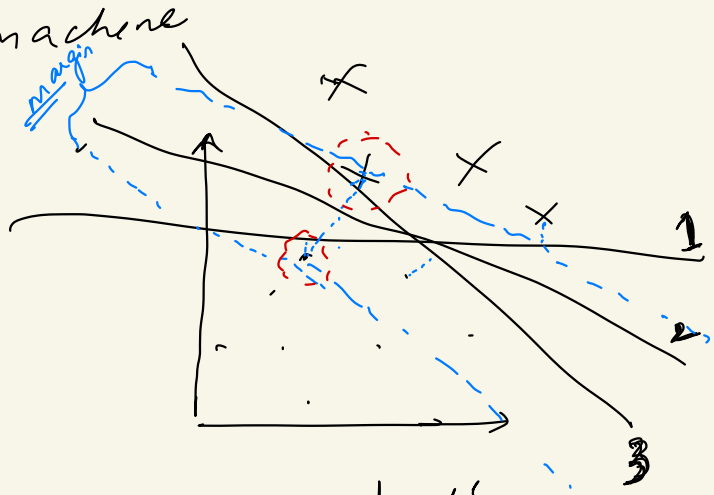
Twin SVM

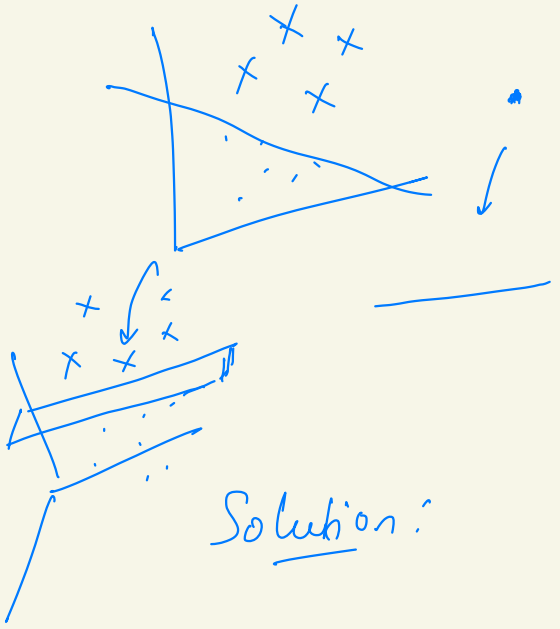
MCM: Minimal complexity Machine

complexity of machine

$\simeq O(n)$

$= O(n)$





1, 2, 3 valid hyperplanes

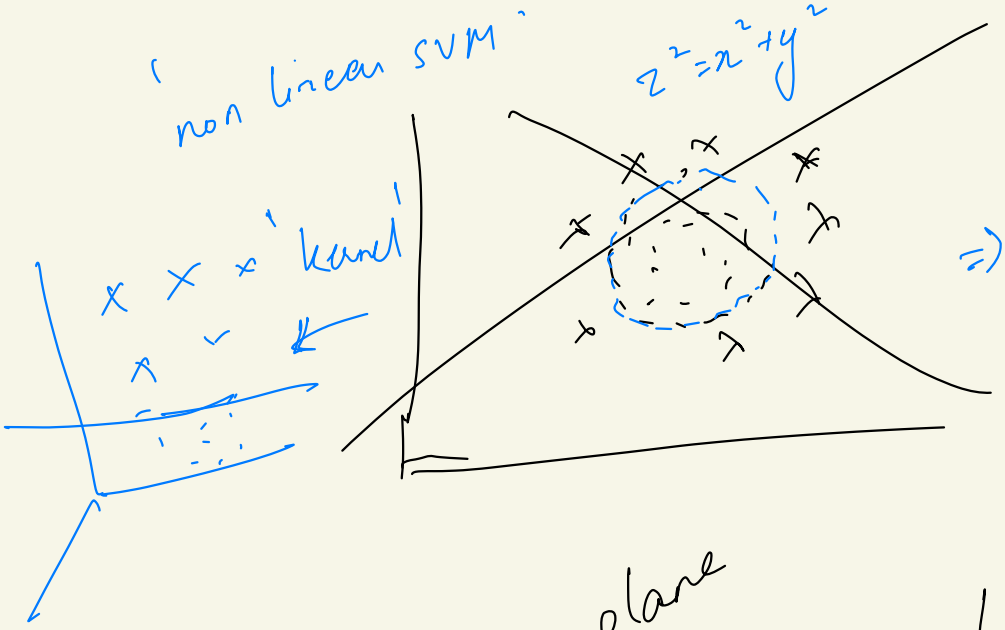
1, 2, 3 is a good hyperplane!

Solution: Find that hyperplane which has maximum margin!

Support vectors

non linear SVM

$z^2 = x^2 + y^2$



kernel

low dimensional space

is non linearly separable

mercer's condition

kernel trick

line → plane

manifold learning ←

high dimensional space

linearly separable

homogeneous coordinate system

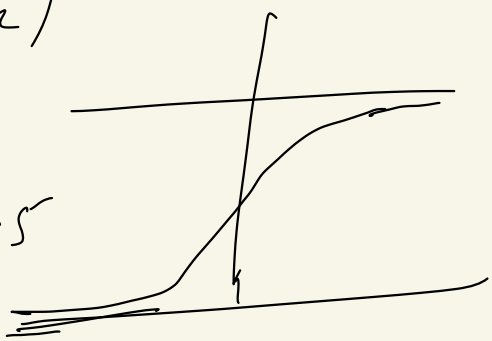
$x, y \longrightarrow w \quad \frac{x}{w}, \frac{y}{w}, w$

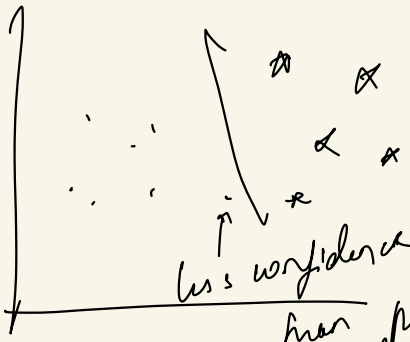
Logistic Regression:

$P(y=1|x) = h(x) = \sigma(\beta^T x)$

Predict 1 when $h(x) > 0.5$

0 when $h(x) < 0.5$





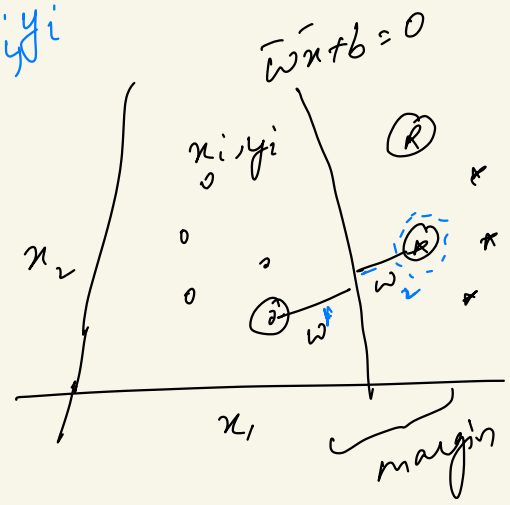
less confidence from points farther from decision boundary

Large no of features

use Bayesian Learning

we look at all possible classifiers weigh them by their

aposterior probability & combined to get the answer
It is not computationally efficient.



$w_1 = w_2 = w$

$w_1 x_1 + w_2 x_2 + b = 0$

Goal: margin width is highest & classifier has highest generalization ability. Points closest to margin: Support vectors

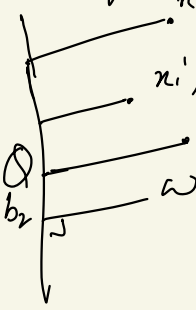
Distance of closest +ve = Distance of closest -ve

minimum of 2 support vectors

Functional margin: dist of x_i, y_i wrt w, b from decision boundary

$\gamma^i = y_i (w^T x_i + b)$ = Normal to the line

$2w$ = Margin



Larger functional margin more confidence to classify

$S = \{(x_1, y_1) \ldots (x_m, y_m)\}$

$\gamma = \min_{i=1}^{m} \gamma^i$

and also depends on coefficients of line (Scaling factors)

Line: $2x + 3y + 1 = 0$

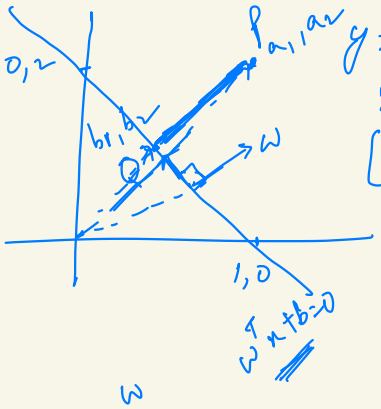
Geometric margin: to get rid of scaling factors

unit vector: $\frac{w}{\|w\|}$

$= \left(\frac{2}{\sqrt{13}}, \frac{3}{\sqrt{13}}\right)$

$P = Q + \gamma \frac{w}{\|w\|}$

$P = Q + \gamma \frac{w}{\|w\|}$



$y = -2x + 2$

$2x + y - 2 = 0$

$\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} - 2 = 0$

$w \qquad\qquad b$

$\frac{w}{\|w\|} \left[\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right]$

$(a_1, a_2) = (b_1, b_2) + \gamma \frac{w}{\|w\|}$

$w^T \left((a_1, a_2) - \gamma \frac{w}{\|w\|} \right) + b = 0$

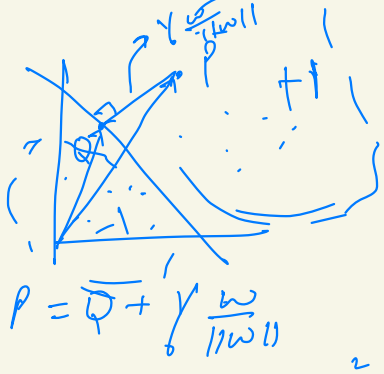
$\gamma = \frac{w^T (a_1, a_2) + b}{\|w\|}$

$\gamma = \min_{i=1}^{m} \gamma_i \rightarrow$ GM

$\gamma_i = y_i \left(w^T x + b \right)$

$= \frac{w^T}{\|w\|} (a_1, a_2) + \frac{b}{\|w\|}$

$\gamma = y \frac{w^T}{\|w\|} (a_1, a_2) + \frac{b}{\|w\|}$



$P = Q + \gamma \frac{w}{\|w\|}$

$w^T w = \|w\|^2$

$w^T P = w^T Q + \gamma \frac{w^T w}{\|w\|}$

$w^T P = w^T Q + \gamma \|w\|$

Let $\|w\| = 1$ (after normalization)

$x_i, y_i \rightarrow \gamma = y \left(w^T (a_1, a_2) + b \right)$ ✓✓

(GM) ↑ class label

Maximize Margin width ✓

→ Maximize $\frac{\gamma}{\|w\|}$ subject to

→ $y_i (w^T x_i + b) \geq \gamma$ for $i = 1, 2 \ldots m$

→ Scale so that $\gamma = 1$

→ Maximize $\frac{1}{\|w\|}$ is same as minimize $\|w\|$ $= w \cdot w$

→ minimize $w \cdot w$ subject to constraints

for all x_i, y_i, $i = 1, \ldots m$:

$w^T x_i + b \geq 1$ if $y_i = 1$

$w^T x_i + b \leq -1$ if $y_i = -1$

if (w, b) is decision boundary

$\frac{\gamma}{\|w\|}$ is the geometric margin

max

Large Margin Linear Classifier

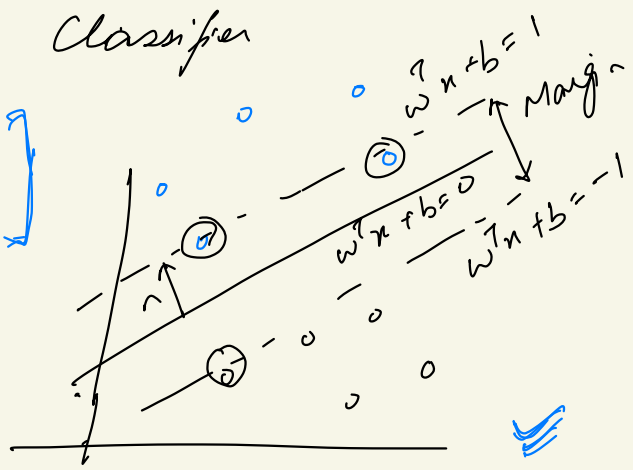
Formulation:

Primal form

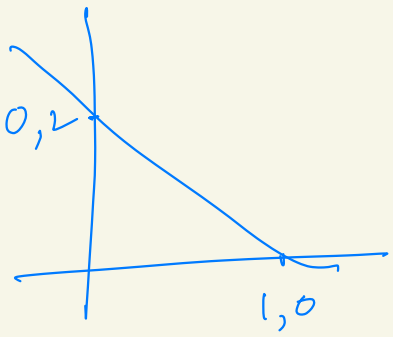
$$\text{minimize} \quad \frac{1}{2}\|w\|^2$$

such that

$$y_i(w^T x_i + b) \geq 1$$



- Optimization problem with convex quadratic objectives & m inequality linear constraints
- Can be solved using QP
- Lagrange duality to get the optimization problem's dual form.
- Allow us to use kernels to get optimal margin classifiers to work efficiently in very high dimensional spaces.
- Allow us to derive an efficient algorithm for solving the above optimization problem that will typically do much better than generic QP software.



$$x + \frac{y}{2} = 1$$

$$2x + y = 2$$

$$2x + y - 2 = 0$$

$$w = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

$$\frac{w}{\|w\|} = \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{bmatrix}$$

Machine Learning

Lect 9

Dr. Sreeparna Mukherjee

Solving the Optimization Problem

accuracy = misclassification

minimize $\frac{1}{2}\|w\|^2$

s.t. $y_i(w^T x_i + b) \geq 1$ (linear inequality constraints)

→ Optimization problem with convex quadratic objectives and linear constraints

→ Can be solved using QP.

→ Lagrange duality to get the optimization problem's dual form:

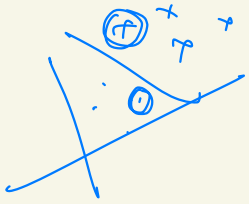
- Allow us to use kernels to get optimal margin classifiers to work efficiently in very high dimensional space
- Allow us to derive an efficient algorithm for solving the above optimization problem that will typically do much better than generic QP software.

Lagrangian Duality

The Primal Problem: $\min_w f(w)$ → Parameters: to find values of w to minimize $f(w)$

s.t. $g_i(w) \leq 0$, $i = 1, \ldots, k$ (k linear inequality constraints)

$h_i(w) = 0$, $i = 1, \ldots, l$ ('l' linear equality constraints)



The generalized Lagrangian:

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

the α_i's ($\alpha_i \geq 0$) and β's are called the Lagrange multipliers.

Lemma:

$$\max_{\alpha, \beta, \alpha_i \geq 0} \mathcal{L}(w, \alpha, \beta) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

A re-written primal:

$$\min_w \max_{\alpha,\beta,\alpha_i \geq 0} L(w,\alpha,\beta)$$

The Primal Problem $p^* = \min_w \max_{\alpha,\beta,\alpha_i \geq 0} L(w,\alpha,\beta)$

The Dual Problem: $d^* = \max_{\alpha,\beta,\alpha_i \geq 0} \min_w L(w,\alpha,\beta)$

Theorem (weak duality):

$$d^* = \max_{\alpha,\beta,\alpha_i \geq 0} \min_w L(w,\alpha,\beta) \leq \min_w \max_{\alpha,\beta,\alpha_i \geq 0} L(w,\alpha,\beta) = p^*$$

Theorem (strong duality):

iff. there exist a saddle point of $L(w,\alpha,\beta)$ we have $\boxed{d^* = p^*}$ → optimal value of both primal & dual forms

KKT conditions

If we exists some saddle point of L, then it satisfies the following Karush Kuhn Tucker (KKT) conditions:

$$\frac{\partial}{\partial w_i} L(w,\alpha,\beta) = 0, \quad i = 1,\dots,k$$

$$\frac{\partial}{\partial \beta_i} L(w,\alpha,\beta) = 0, \quad i = 1,\dots,l$$

$$\alpha_i g_i(w) = 0, \quad i = 1,\dots,m$$

$$g_i(w) \leq 0, \quad i = 1,\dots,m$$

$$\alpha_i \geq 0, \quad i = 1,\dots,m$$

Theorem: If w^*, α^*, β^* satisfy the KKT condition, then it is also a solution to the primal and the dual problems.

Support Vectors

In SVM, we just have g_i's (inequality constraints)
no h_i's (equality constraints)

- Only a few α_i's can be nonzero ⇒ i's are S.Vs.
- Call the training data points whose α_i's are nonzero the support vectors.

$$\alpha_i g_i(w) = 0 \quad , i = 1, \dots, n$$

if $\alpha_i > 0$ then $g(w) = 0$

Solving the Optimization problem

Quadratic programming with linear constraints

$$\text{minimize } \underbrace{\frac{1}{2}\|w\|^2}_{f(w)} \quad \text{s.t. } y_i(w^T x_i + b) \geq 1 \Rightarrow$$

Lagrangian Function

$$f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) +$$

$$\text{minimize } L_p(w, b, \alpha_i) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left(y_i(w^T x_i + b) - 1\right) \quad \rightarrow (1)$$

s.t. $\alpha_i > 0$ ⇒ S.V.

minimize w.r.t w & b for fixed α

$$\frac{\partial L_p}{\partial w} = 0 \Rightarrow \boxed{w = \sum_{i=1}^{n} \alpha_i y_i x_i} \rightarrow (2)$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \rightarrow (3)$$

Put (2) in (1)

$$L_p(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j) - \rightarrow (4)$$

$$b \sum_{i=1}^{m} \alpha_i y_i$$

Put (3) in (4)

$$L_p(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

The Dual Problem is:

Now we have the following dual opt problem:

$$\max_{\alpha} J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

$$s.t \quad \alpha_i \geq 0, \quad i = 1, \ldots, k$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

This is a quadratic programming problem:
→ A global maximum of α_i can always be found.

This dual opt. problem is much easier to solve than primal formulation bcz constraints are simpler

SVM

Once we have the Lagrange multipliers $\{\alpha_i\}$ we can reconstruct the parameter vector w as a weighted combination of the training examples:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i$$

$$w = \sum_{i \in SV} \alpha_i y_i x_i$$

$\alpha_i \geq 0$

α_i's non zero for $i \in SV$

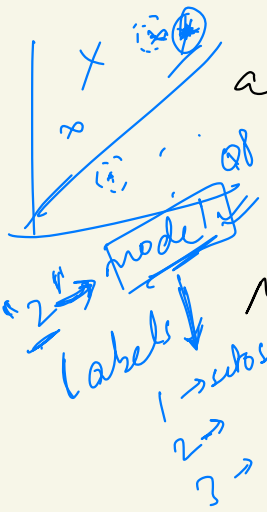
iris

For testing with new data 'z'

labels — Compute

$$w^T z + b = \sum_{i \in SV} \alpha_i y_i (x_i^T z) + b \quad \text{--- (1)}$$

and classify z as class 1 if sum is positive and class 2 otherwise



Note: w need not be formed explicitly we can just use (1) as we take dot product of support vectors with z

ovo classifier
ova classifier

Solving the Opt. Problem

→ The discriminant function is:

$$g(x) = w^T x + b = \sum_{i \in SV} \alpha_i x_i^T x + b$$

→ It relies on a dot product b/w test point x and support vectors x_i ⇒ we get a scalar

→ Solving the opt. problem involved is computing dot products $x_i^T x_j$ b/w all pairs of training points

→ The optimal w is a linear combination of a small no of data points.

Multi label classifier :

CL1 CL2 CL3

OVO

CL1 vs CL2

CL1 vs CL3

binary classifier

OVA

CL1 (positive) vs CL2 CL3 (negative)

KNN

Machine learning

Lect 10

Dr. Sreeraa
Mukherjee

Non linear SVM and kernel functions

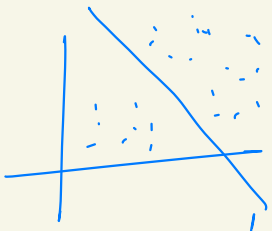
$\phi : x \rightarrow \phi(x)$

original feature space can be mapped into new feature space where it is linearly separable → higher dimensional

But what about computational cost !!

linear SVM : $\sum \alpha_i \alpha_j y_i y_j x_i \cdot x_j$ ↗ d-dimensional ↗ d-dimensional

we need to find pairwise dot products



mapping fn. $\phi : x \rightarrow \phi(x)$

if we have m training examples we need to do m^2 computations like this

Computation time $O(d^2)$

If Transform to high dimensional space $D \gg d$

$\sum \alpha_i \alpha_j y_i y_j \phi(x_i) \cdot \phi(x_j)$

$O(D^2)$

↯ more computational cost

$K(x_i, x_j)$

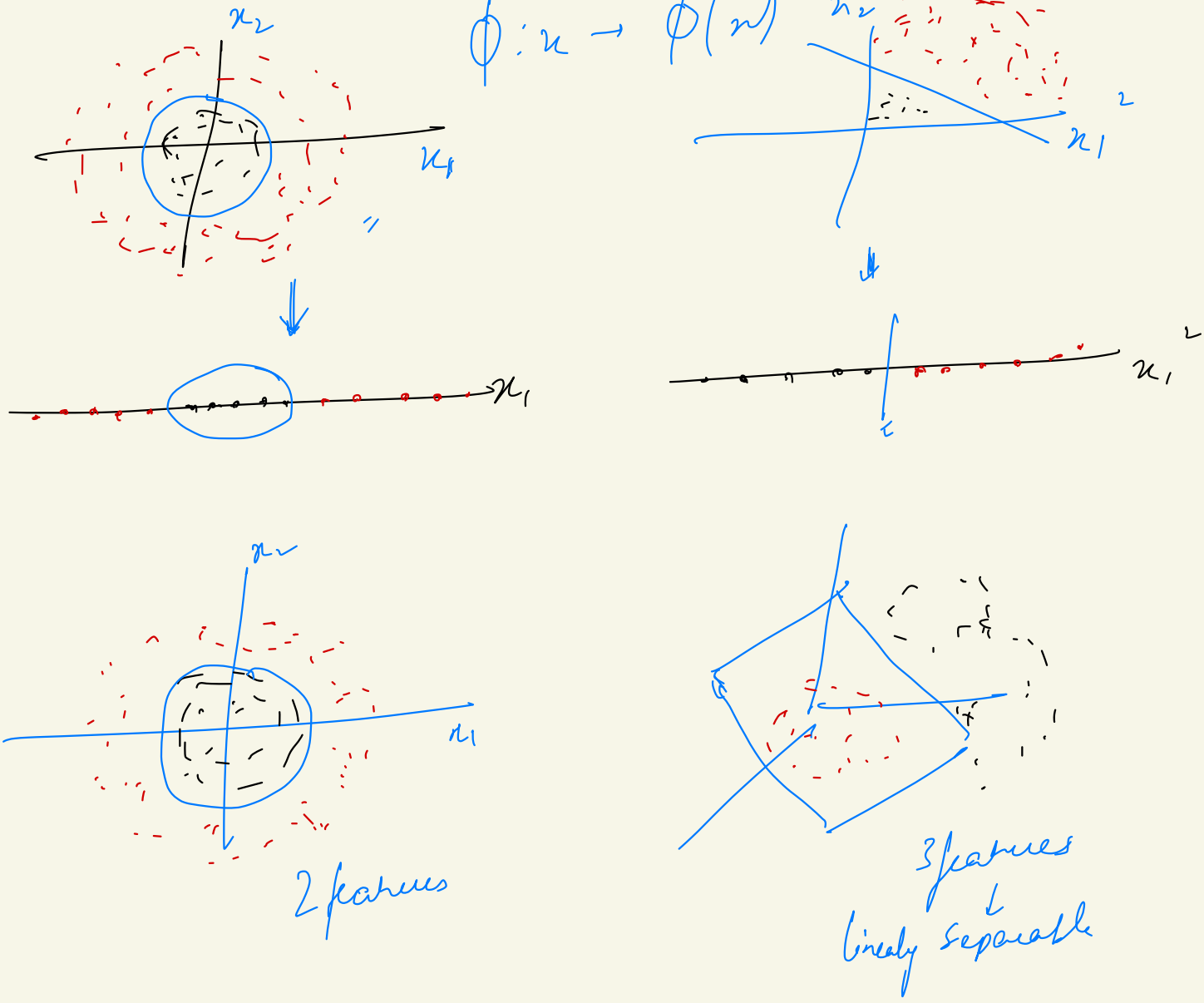
$\phi(x_i) = x_i \quad \phi(x_j) = x_j$

$e^{-\|x_i - x_j\|^2}$

$(1 + x_i x_j)^{power}$

Soln: Use kernel function to reduce computational costs

Visualize feature space



In certain feature transformation, where after transformation, SVM can be solved efficiently.

kernel function:

x_a	x_b	$x_a \cdot x_b$
$\phi(x_a)$	$\phi(x_b)$	$\phi(x_a) \cdot \phi(x_b)$

$$K(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$$

↳ may be easy to compute as it is a fn. of $x_a \& x_b$ without expanding $\phi(x_a)$ or $\phi(x_b)$

also called kernel trick

What is Kernel: i) Original input attributes is mapped to a new set of input features via feature mapping ϕ

ii) Since the algo can be written in terms of the scalar product, we replace $x_a \cdot x_b$ with $\phi(x_a) \cdot \phi(x_b)$

iii) for certain ϕ, there is a simple operation on 2 vectors in the low-dim space that can be used to compute the scalar product of their 2 images in the high dim space

$$k(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$$

let the kernel do the work rather than do the scalar product in high dim space

dot product = scalar or vector??

Nonlinear SVM: The kernel Trick

→ With this mapping, our discriminant function is now:

$$g(x) = w^T \phi(x) + b$$

$$= \sum_{i \in SV} \alpha_i \boxed{\phi(x_i)^T \phi(x)} + b$$

$w^T x + b$
$w^T \phi(x) + b$

Indual form

→ We only use the dot product of feature vectors in both the training & test

→ A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space:

$$k(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$$

This $k(x_a, x_b)$ may be very inexpensive to compute even if $\phi(x_a)$ may be extremely high dimensional

Kernel Example

$x_i = [x_{i1}\ x_{i2}]$ $x_j = [x_{j1}\ x_{j2}]$

2 dimensional vectors $\bar{x} = [x_1\ x_2]$

Let $k(x_i, x_j) = (1 + x_i \cdot x_j)^2$

$$k(x_i, x_j) = (1 + x_i \cdot x_j)^2$$

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= \left[1 \quad x_{i1}^2 \quad \sqrt{2}\, x_{i1} x_{i2} \quad x_{i2}^2 \quad \sqrt{2} x_{i1} \quad \sqrt{2} x_{i2}\right] \cdot \left[1 \quad x_{j1}^2 \quad \sqrt{2} x_{j1} x_{j2} \quad x_{j2}^2 \quad \sqrt{2} x_{j1} \quad \sqrt{2} x_{j2}\right]$$

$= \phi(x_i) \cdot \phi(x_j)$

where $\phi(x) = \left[1 \quad x_1^2 \quad \sqrt{2}x_1x_2 \quad x_2^2 \quad \sqrt{2}x_1 \quad \sqrt{2}x_2\right]$

$(1 + x_i \cdot x_j)^2$

$x_i = [x_{i1} \quad x_{i2}]$

$x_j = [x_{j1} \quad x_{j2}]$

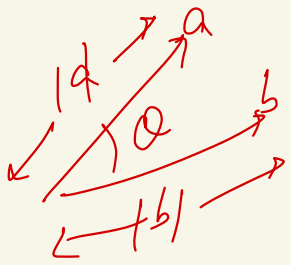
$\left[1 + x_{i1}x_{j1} + x_{i2}x_{j2}\right]^2$

$(a+b+c)^2 = a^2 + b^2 + c^2 + 2ab + 2bc + 2ca$

$= 1 + x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + 2x_{i1}x_{i2}x_{j1}x_{j2}$

Dot product

$a \cdot b = |a|\,|b| \cos\theta$

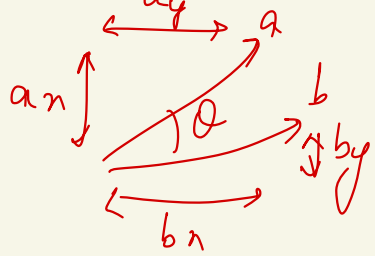


or

$a \cdot b = a_x \times b_x + a_y \times b_y$ (then y-components)

Scalar quantity

multiply the x-components of 2 vectors



$x_i = [x_{i1} \quad x_{i2}]$

$x_j = [x_{j1} \quad x_{j2}]$

$x_i \cdot x_j = x_{i1} \times x_{j1} + x_{i2} \times x_{j2}$

$(1 + x_i \cdot x_j)^2 = (1 + x_{i1}x_{j1} + x_{i2}x_{j2})^2 \Rightarrow (a+b+c)^2$

Commonly used kernel functions

→ Linear SVM corresponds to linear kernel:

$$k(x_i, x_j) = x_i \cdot x_j$$ where $\phi(x_i) = x_i$ & $\phi(x_j) = x_j$ (Identity mapping)

→ Polynomial of power p:

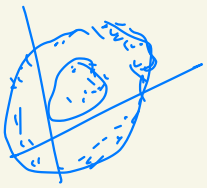
$$k(x_i, x_j) = (1 + x_i \cdot x_j)^p$$

→ Gaussian (radial basis function):

$$k(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

→ Sigmoid:

$$k(x_i, x_j) = \tanh(\beta_0 x_i \cdot x_j + \beta_1)$$



In general, functions that satisfy Mercer's condition can be kernel functions

Eg: Text classification we can use kernels based on similarity of words of text

Kernel functions

→ kernel function can be thought of as a similarity measure b/w the input objects

→ Not all similarity measure can be used as kernel function

→ Mercer's condition states that any positive semi-definite kernel $k(x, y)$ ie

$$\sum k(x_i, x_j)\, c_i c_j \geq 0 \quad \forall \text{ real no.}$$

Generally this similarity measure is symmetric

positive semi-definite matrix

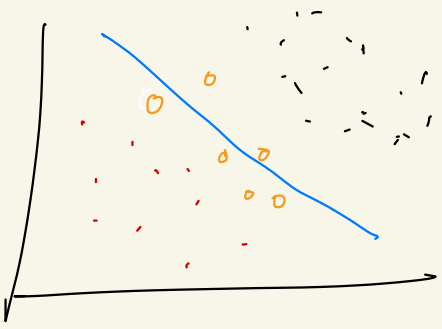
Any matrix is symmetric & all the eigen values are +ve

$$k(x_i, x_j) = k(x_j, x_i)$$

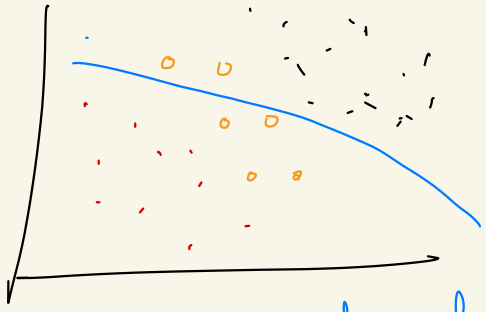
If you can write similarity b/w points as a matrix & this matrix is symmetric +ve semi definite then a kernel function can be either

→ Such functions can be expressed as a dot product in high dimensional space

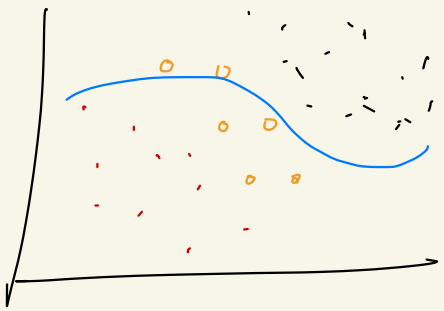
SVM examples



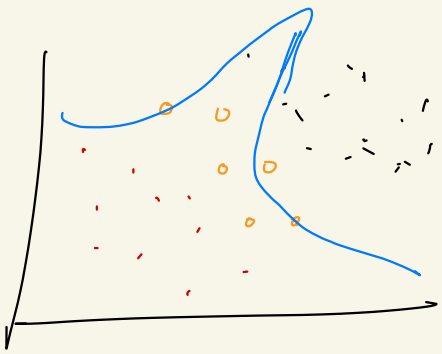
↓ Linear kernel with noise



↓ 2nd order polynomial



↓ 4th order polynomial



↓ 8th order polynomial

Non linear SVM optimization

Formulation (Lagrangian Dual Problem)

$\phi(x_i) \cdot \phi(x_j)$ ⇑ $x_i \cdot x_j$

maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \; k(x_i, x_j)$

s.t $0 \le \alpha_i \le C$ → hyperparameter

$C = 0.001$

KKT condition → $\sum_{i=1}^{n} \alpha_i y_i = 0$

The solution of the discriminant function is

$$g(x) = \sum_{i \in SV} \alpha_i \, k(x_i, x_j) + b$$

↓ $k \; \phi(x_i) \cdot x$

$\phi(x_i)^T \cdot \phi(x)$

test point x

Performance:

→ SVM work very well in practice

User must choose the kernel function & its parameters (C) appropriately

→ They can be expensive in time & space for big datasets

→ Computation of the maximum margin hyperplane depends on the square of number of training cases

$O(d^2)$ ↓ cost

→ We need to store all support vectors

→ The kernel trick can also be used to do PCA in a much higher-dimensional

space, thus giving a non linear version of PCA in the original space

Multiclass classification

SVMs can only handle 2-class outputs

Learn N-SVMs

- SVM 1 learns Class1 vs Rest
- SVM2 learns Class2 vs Rest

SVM N learns Class N vs Rest

Then to predict o/p for a new input, just predict with each SVM & find out which one put the prediction the furthest into the positive region

majority voting

Class1 Class2 Class3 ... Class N

OVO
one vs one
- SVM1 Class1 vs Rest
Class2 vs Rest
-
-

OVA
one vs all
Class1 ——— negative

sk-learn
svm.predict
svm.classify

python
matlab
⇒ Lib-svm library

Machine Learning

Lect 11

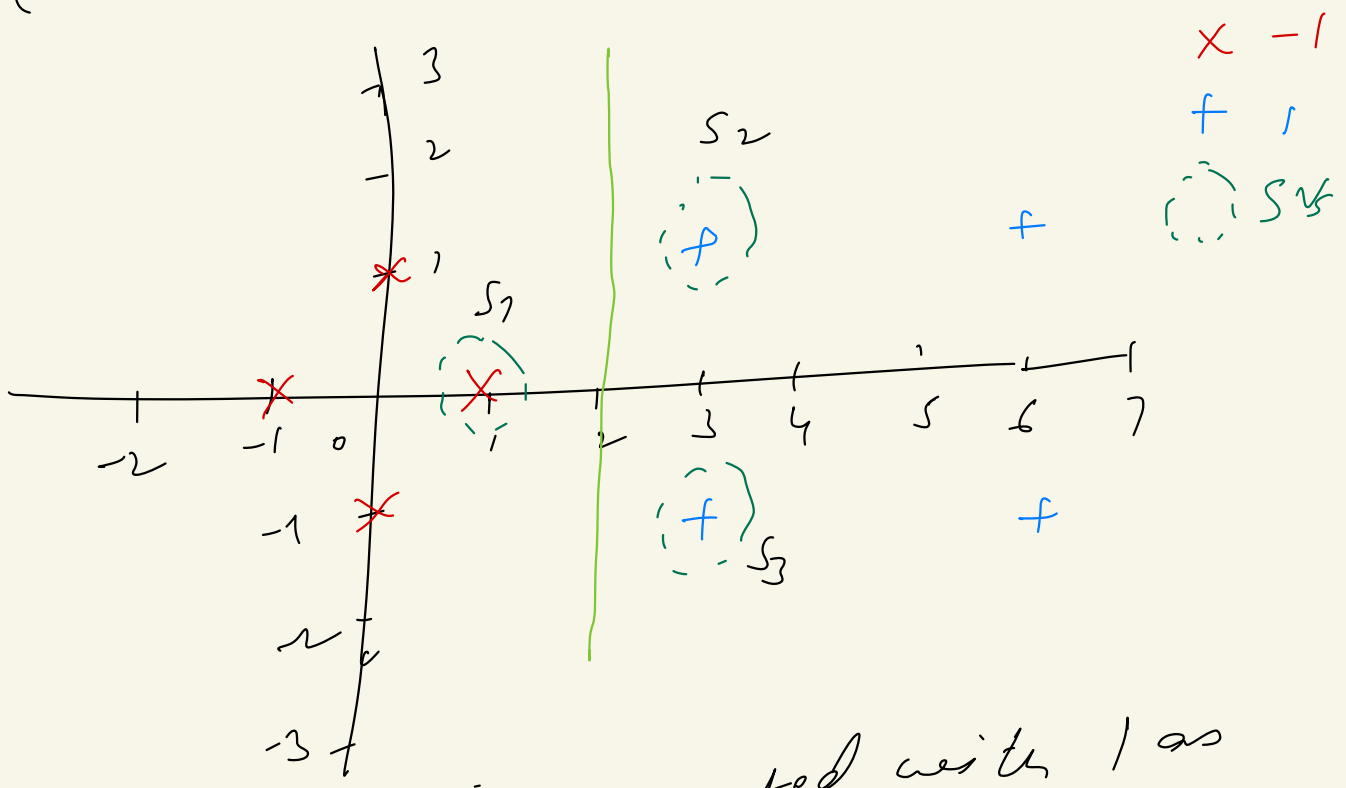
Dr. Swarna
Mukherjee

SVM Numerical

Suppose we are given the following data points

$\left\{ \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 3 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ , \end{pmatrix}, \begin{pmatrix} 6 \\ -1 \end{pmatrix} \right\}$ true labeled $+1$

$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix} \right\}$ very labeled -1



Each vector is augmented with 1 as bias input

$S_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \tilde{S}_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$

$S_2 = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \quad \tilde{S}_2 = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \quad S_3 = \begin{pmatrix} 3 \\ -1 \end{pmatrix} \tilde{S}_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$

$$\alpha_1 \tilde{s}_1 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_1 + \alpha_3 \tilde{s}_3 \tilde{s}_1 = -1$$

$$\alpha_1 \tilde{s}_2 \tilde{s}_1 + \alpha_2 \tilde{s}_2 \tilde{s}_2 + \alpha_3 \tilde{s}_2 \tilde{s}_3 = +1$$

$$\alpha_1 \tilde{s}_3 \tilde{s}_1 + \alpha_2 \tilde{s}_3 \tilde{s}_2 + \alpha_3 \tilde{s}_3 \tilde{s}_3 = +1$$

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} +$$

$$\alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} +$$

$$\alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} = +1$$

$$\alpha_3 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$$

$$+ \alpha_3 \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix} = +1$$

$$\alpha_1(1+0+1) + \alpha_2(3+0+1) + \alpha_3(3+0+1) = -1$$

$$\alpha_1(3+0+1) + \alpha_2(9+1+1) + \alpha_3(9-1+1) = 1$$

$$\alpha_1(3+0+1) + \alpha_2(9-1+1) + \alpha_3(9+1+1) = 1$$

$$2\alpha_1 + 4\alpha_2 + 4\alpha_3 = -1$$

$$9\alpha_1 + 11\alpha_2 + 9\alpha_3 = 1$$

$$4\alpha_1 + 9\alpha_2 + 11\alpha_3 = 1$$

$$\alpha_1 = -3.5 \quad \alpha_2 = 0.75 \quad \alpha_3 = 0.75$$

$$\tilde{w} = \sum_i \alpha_i \tilde{s}_i$$

$$= -3.5\begin{pmatrix}1\\0\\1\end{pmatrix} + 0.75\begin{pmatrix}3\\1\\1\end{pmatrix} + 0.75\begin{pmatrix}3\\-1\\1\end{pmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix} \rightarrow \tilde{w}$$
← bias

$$y = wx + b$$

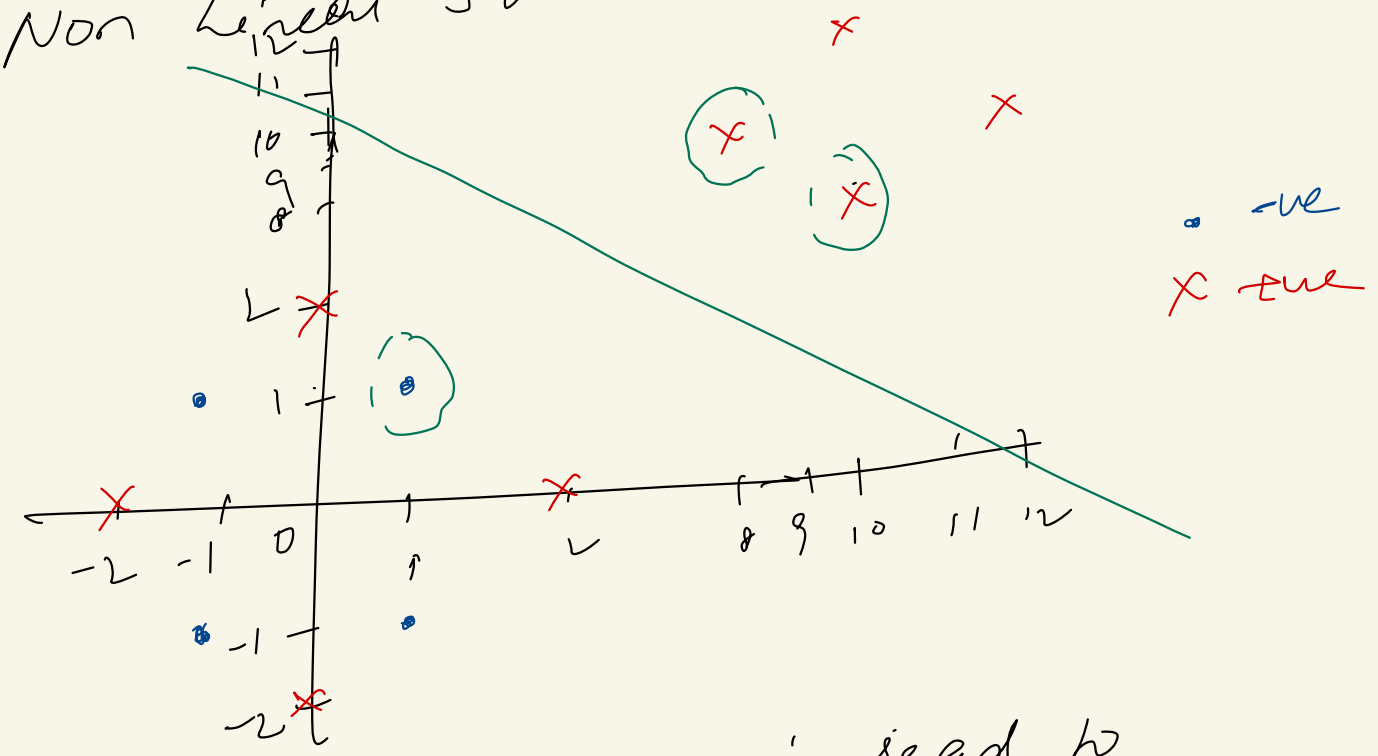
$$w = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad b = -2$$

Q/ -ve class points -1 $\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \end{pmatrix}$

+ve class points $+1$ $\begin{pmatrix} 4 \\ 0 \end{pmatrix}, \begin{pmatrix} 5 \\ 1 \end{pmatrix}, \begin{pmatrix} 5 \\ -1 \end{pmatrix}, \begin{pmatrix} 6 \\ 0 \end{pmatrix}$

find the generating function

Non Linear SVM



$\phi \Rightarrow$ Mapping function is reqd to transform the data to a new feature space where a seperating hyperplane can be found.

$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} 6 - x_1 + (x_1 - x_2)^2 \\ 6 - x_2 + (x_1 - x_2)^2 \end{cases} \quad \text{if } \sqrt{x_1^2 + x_2^2} > 2$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{otherwise}$$

→ Blue class vectors are: $\begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \end{pmatrix}$ no change since

$\sqrt{x_1^2 + x_2^2} < 2$ # vectors

→ Red class vectors $\begin{pmatrix}2\\0\end{pmatrix}, \begin{pmatrix}0\\2\end{pmatrix}, \begin{pmatrix}-2\\0\end{pmatrix}, \begin{pmatrix}0\\-2\end{pmatrix}$

$$\phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \phi\begin{pmatrix}2\\0\end{pmatrix} = \begin{pmatrix}6-2+(2-0)^2\\6-0+(2-0)^2\end{pmatrix}$$

$$= \begin{pmatrix}8\\10\end{pmatrix}$$

$$\phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \phi\begin{pmatrix}0\\2\end{pmatrix} = \begin{pmatrix}10\\8\end{pmatrix}$$

$$\phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \phi\begin{pmatrix}-2\\0\end{pmatrix} = \begin{pmatrix}12\\10\end{pmatrix}$$

$$\phi\begin{pmatrix}x_1\\x_2\end{pmatrix} = \phi\begin{pmatrix}0\\-2\end{pmatrix} = \begin{pmatrix}10\\12\end{pmatrix}$$

$$S_1 = \begin{pmatrix}8\\10\end{pmatrix} \quad S_2\begin{pmatrix}10\\8\end{pmatrix} \quad S_3\begin{pmatrix}1\\1\end{pmatrix}$$

$$\tilde{S}_1 = \begin{pmatrix}8\\10\\1\end{pmatrix} \quad \tilde{S}_2 = \begin{pmatrix}10\\8\\1\end{pmatrix} \quad \tilde{S}_3 = \begin{pmatrix}1\\1\\1\end{pmatrix}$$

$$\alpha_1 \tilde{S}_1 \tilde{S}_1 + \alpha_2 \tilde{S}_1 \tilde{S}_2 + \alpha_3 \tilde{S}_1 \tilde{S}_3 = +1$$

$$\alpha_1 \tilde{S}_2 \tilde{S}_1 + \alpha_2 \tilde{S}_2 \tilde{S}_2 + \alpha_3 \tilde{S}_2 \tilde{S}_3 = +1$$

$$\alpha_1 \tilde{S}_3 \tilde{S}_1 + \alpha_2 \tilde{S}_3 \tilde{S}_2 + \alpha_3 \tilde{S}_3 \tilde{S}_3 = -1$$

$$165\alpha_1 + 161\alpha_2 + 19\alpha_3 = +1$$

$$164\alpha_1 + 165\alpha_2 + 19\alpha_3 = +1$$

$$19\alpha_1 + 19\alpha_2 + 3\alpha_3 = -1$$

$$\alpha_1 = \alpha_2 = 0.8898 \quad \alpha_3 = -1.4218$$

$$\tilde{w} = \sum_i \alpha_i \tilde{s}_i$$

$$\tilde{w} = \alpha_1 \begin{pmatrix} 8 \\ 10 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 10 \\ 8 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.1243 \\ 0.1243 \\ -1.2501 \end{pmatrix}$$

$$y = wx + b$$

$$w = \begin{pmatrix} 0.1243/0.1243 \\ 0.1243/0.1243 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

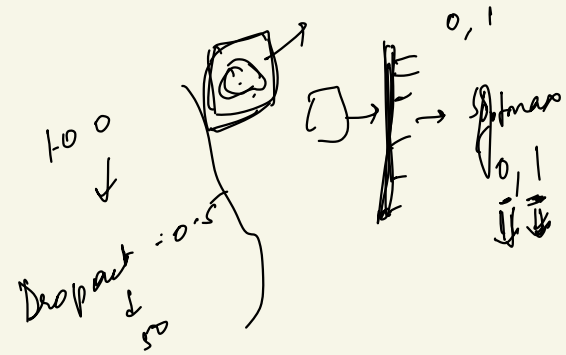
$$b = -1.2501/0.1243$$

$$= -10.057$$

Test point $x_1, x_2 = -1, 2$ belongs to Red point class true

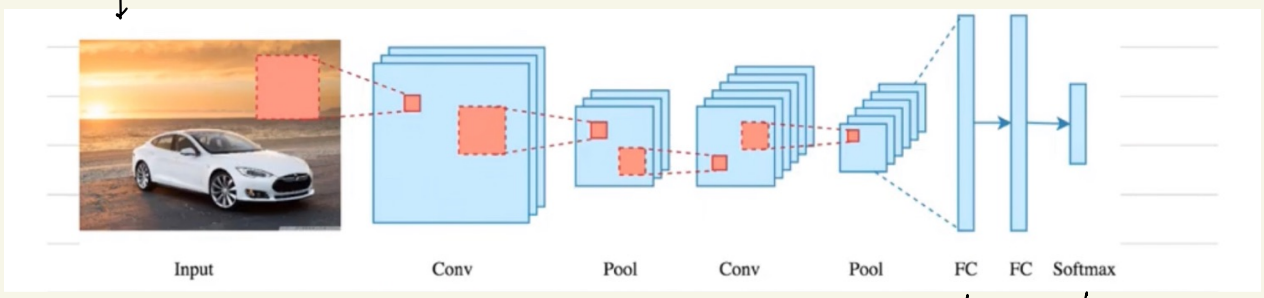
$$\phi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 16 \\ 13 \end{pmatrix}$$

$$w\,\phi(x) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}\begin{pmatrix} 16 \\ 13 \end{pmatrix} = 29 > 10$$



Introduction to Deep learning (CNNs)

Convolutional Neural networks



Correlation
Convolution
100°

1D input

$h(x)$ → f80°

CNNs over ANNs – i) No context learnt (Neighborhood of pixels) 3×3 4×4 2×2 array
ii) More parameters to learn in ANNs

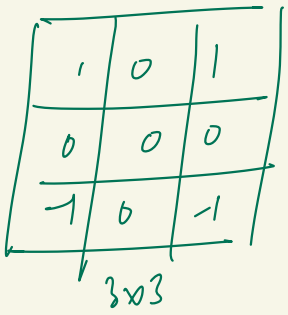
no flipping ↓ Flip kernel 48°

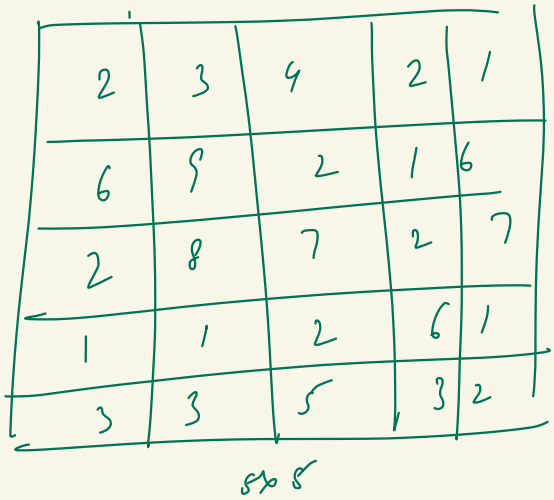
Correlation / Convolution

In case of CNNs → it is actually correlation only

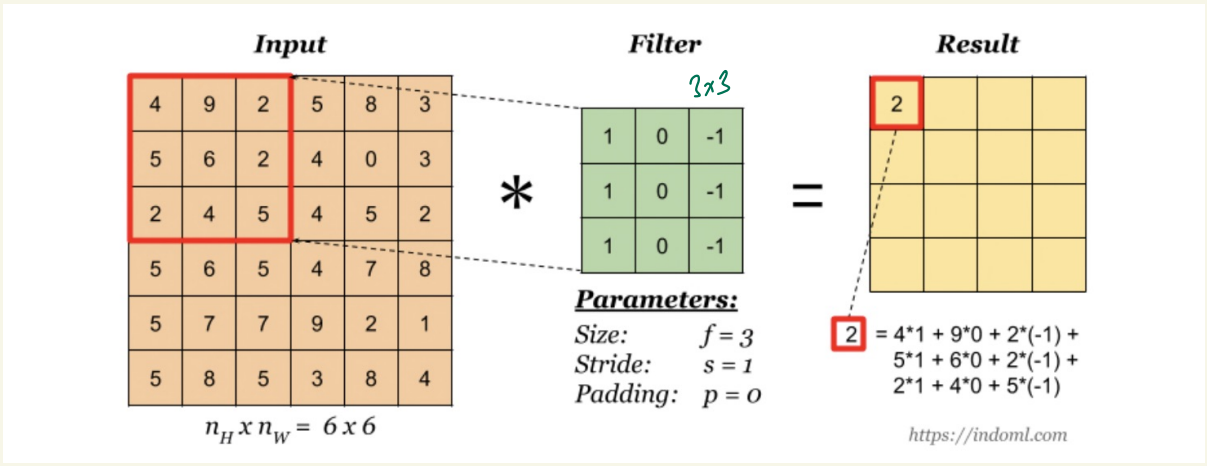
Conv (filter size, stride, padding) → pooling

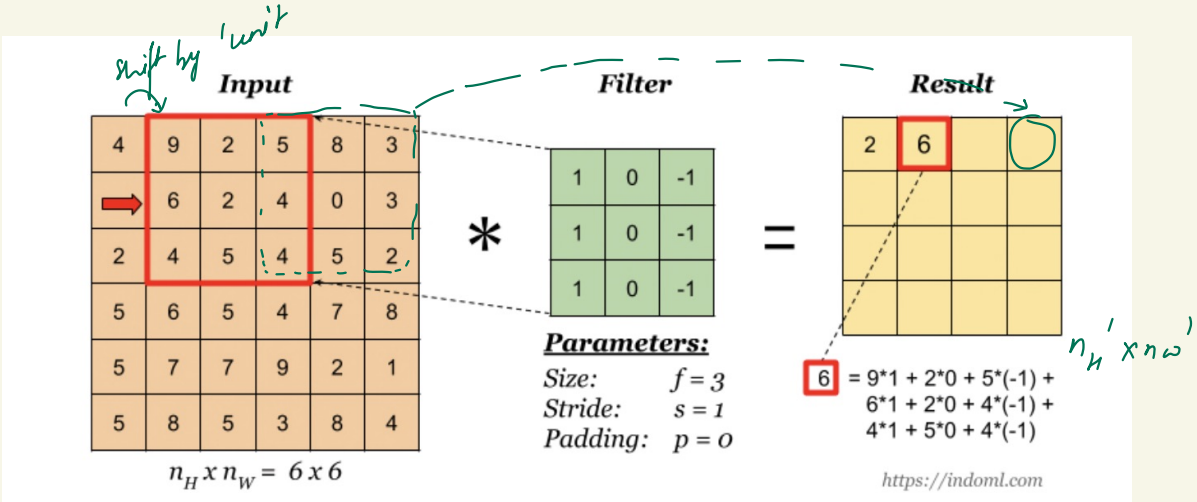
① Filter size





② Convolution





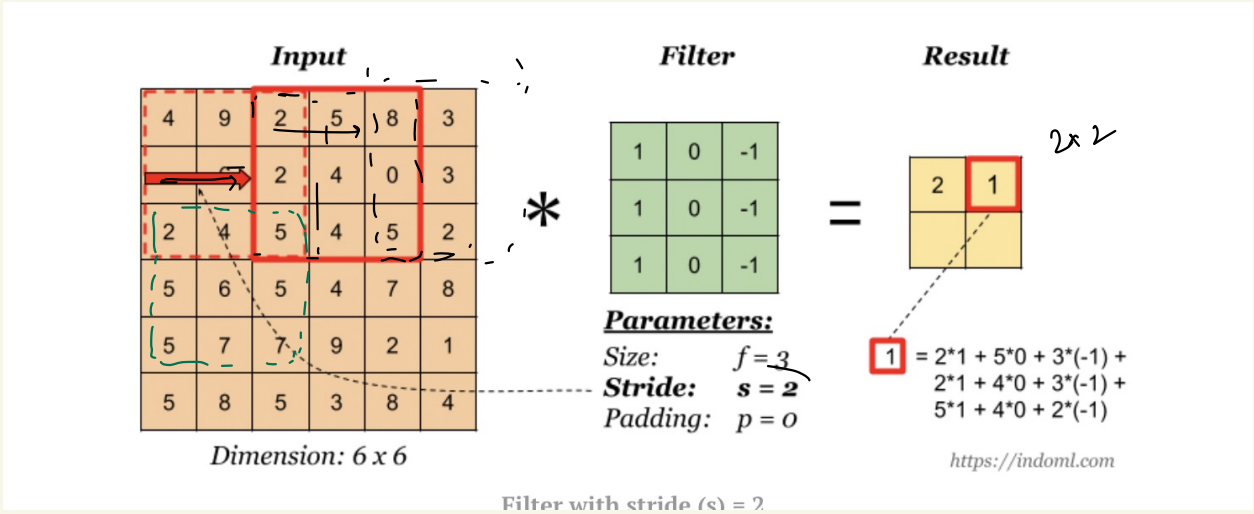
$$n_H' = n_H - f + 1 = 6 - 3 + 1 = 4$$

Stride

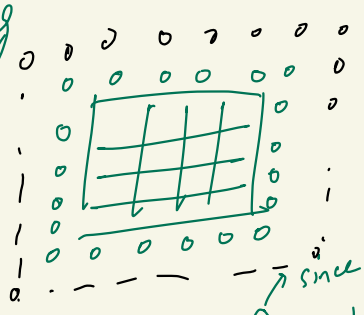
Let $s=1$, $p=0$

$$n_H' = n_H - f + 1$$

$$n_H' = \left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor$$



Padding



$4\times4 \rightarrow 6\times6 \rightarrow 8\times8$

$p=1$ $\quad p=2$

since $p=1$ increase image size by 2 units

$$n_H' = n - f + 2p + 1$$

Convolution reduces size

Is padding 'valid' || 'same' $\rightarrow p = \frac{f-1}{2}$

'valid' → reduces size of image

'same' → keeping original size of image same

$= \frac{3-1}{2} = 1$

$6\times6 \rightarrow 6\times6$

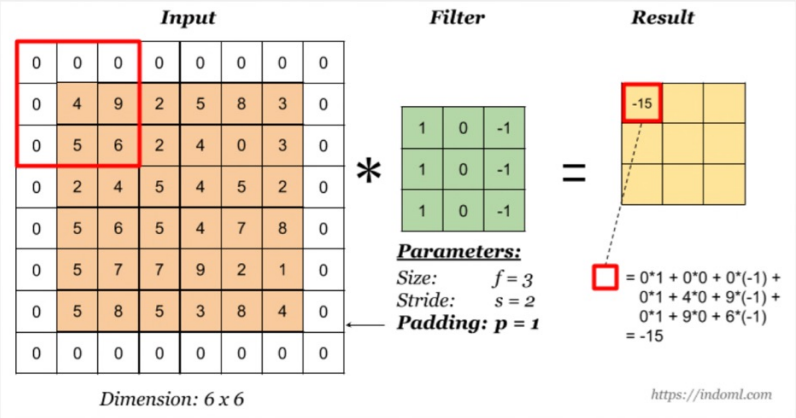
$$n_H' = \left\lfloor \frac{n_H + 2p - f}{s} + 1 \right\rfloor = n_H$$

$$\Rightarrow p = \frac{f-1}{2}$$

Padding

Padding has the following benefits:

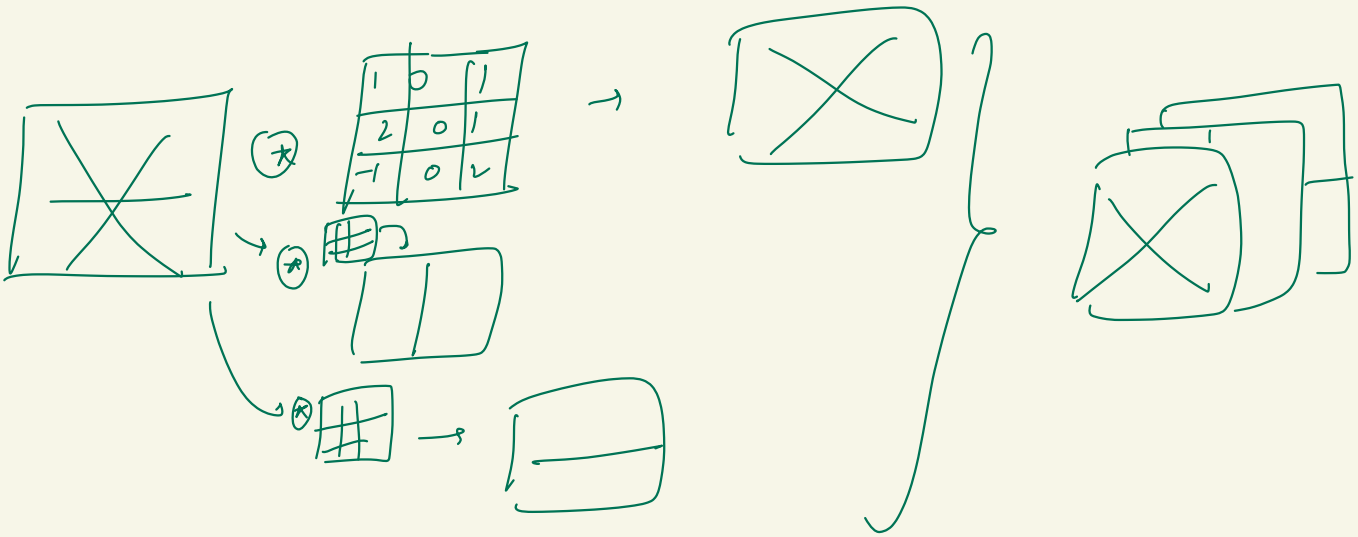
1. It allows us to use a CONV layer without necessarily shrinking the height and width of the volumes. This is important for building deeper networks, since otherwise the height/width would shrink as we go to deeper layers.
2. It helps us keep more of the information at the border of an image. Without padding, very few values at the next layer would be affected by pixels as the edges of an image.

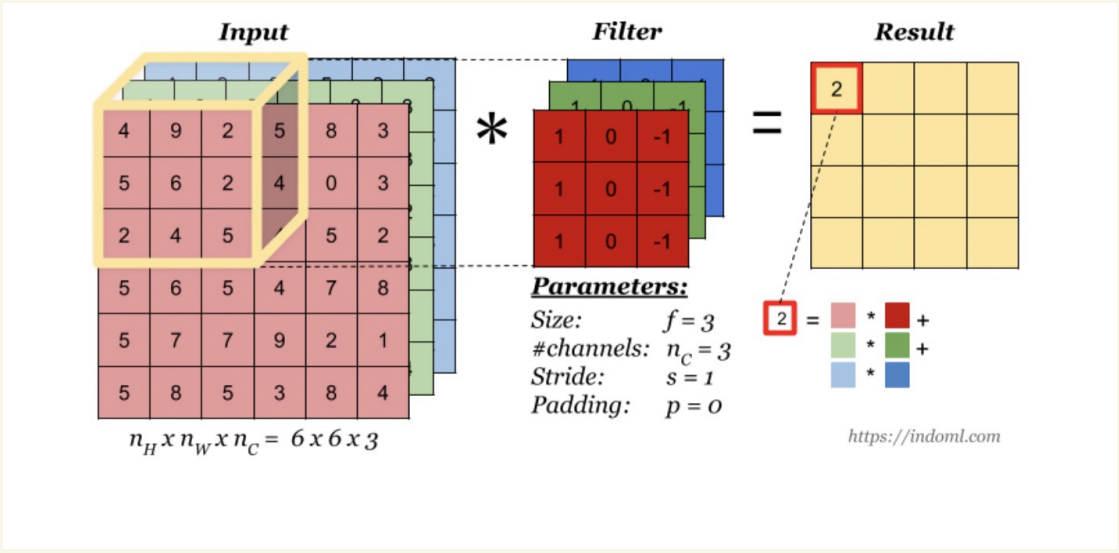


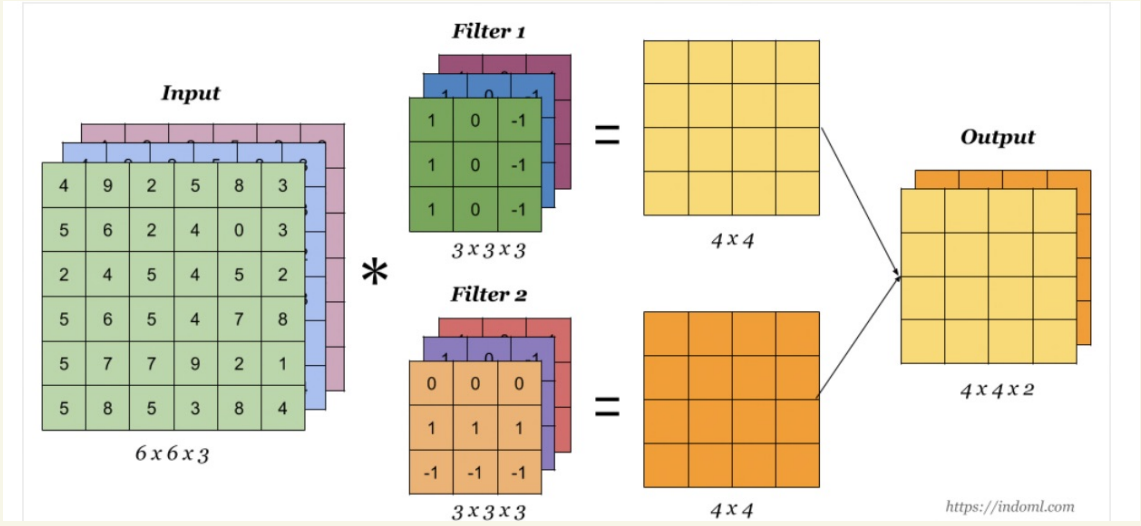
Notice the the dimension of the result has changed due to padding. See the following section on how to calculate output dimension.

Some padding terminologies:

- "**valid**" padding: no padding
- "**same**" padding: padding so that the output dimension is the same as the input

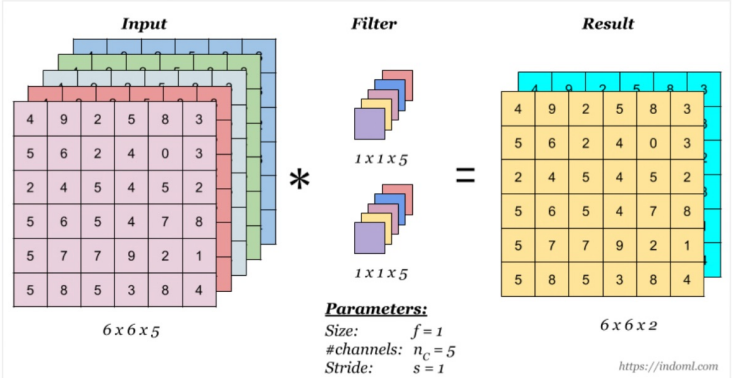


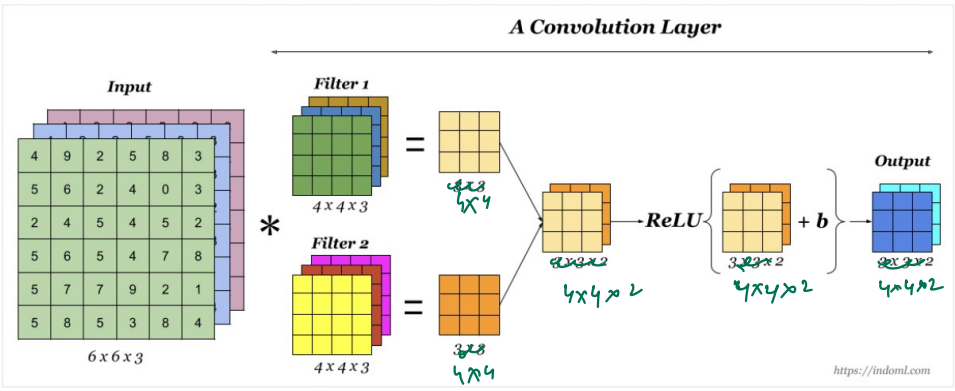




1 x 1 Convolution

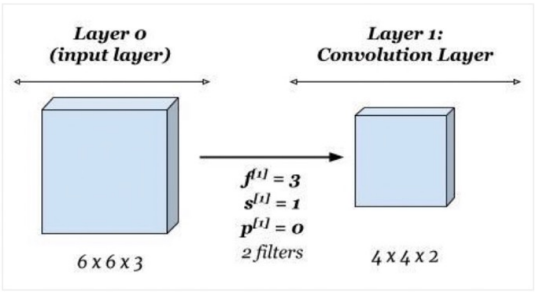
This is convolution with 1 x 1 filter. The effect is to flatten or "merge" channels together, which can save computations later in the network:





Shorthand Representation

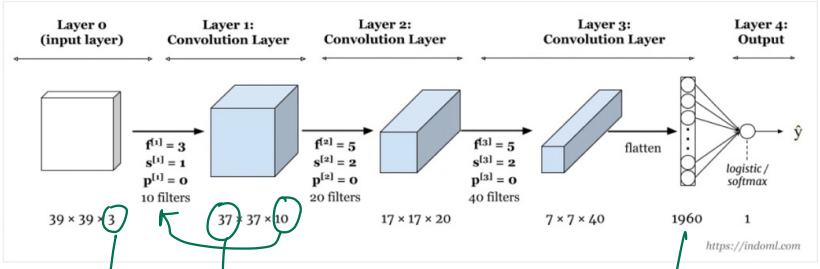
This simpler representation will be used from now on to represent one convolutional layer:



~

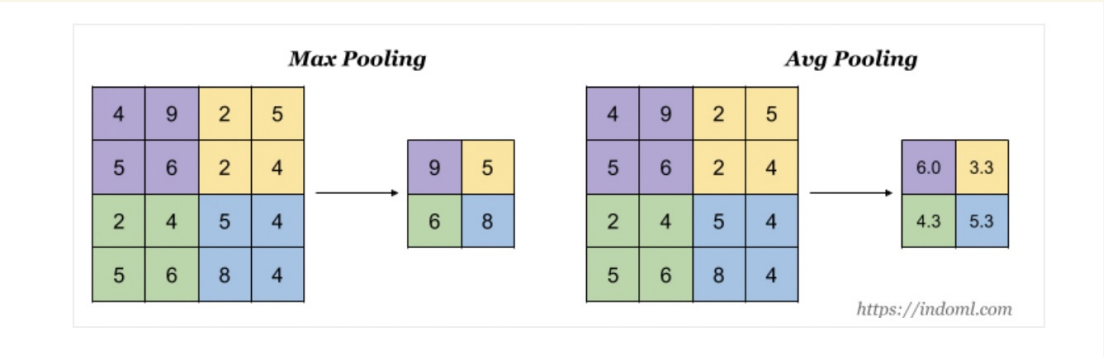
Sample Complete Network

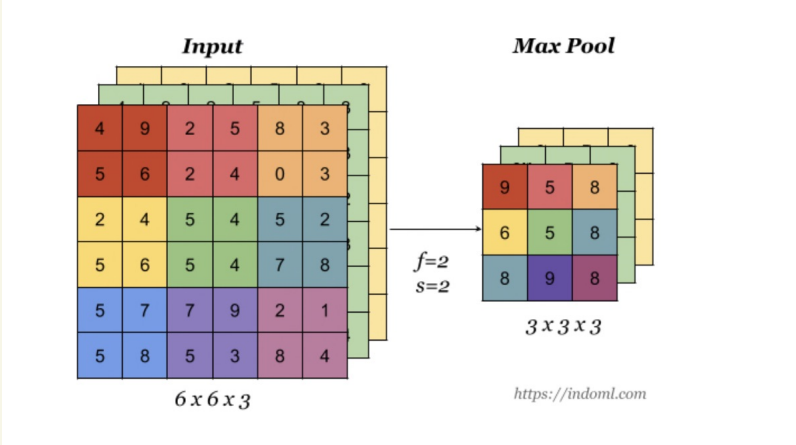
This is a sample network with three convolution layers. At the end of the network, the output of the convolution layer is flattened and is connected to a logistic regression or a softmax output layer.



Pooling → Avg
→ Max (more used)

Hy peparameters: size (f), stride(s), type (max/avg)

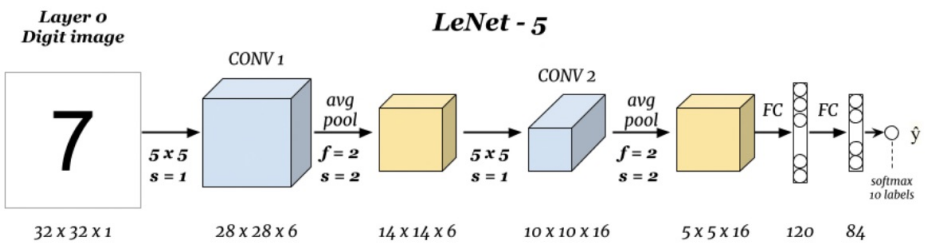




Well Known Architectures

Classic Network: LeNet – 5

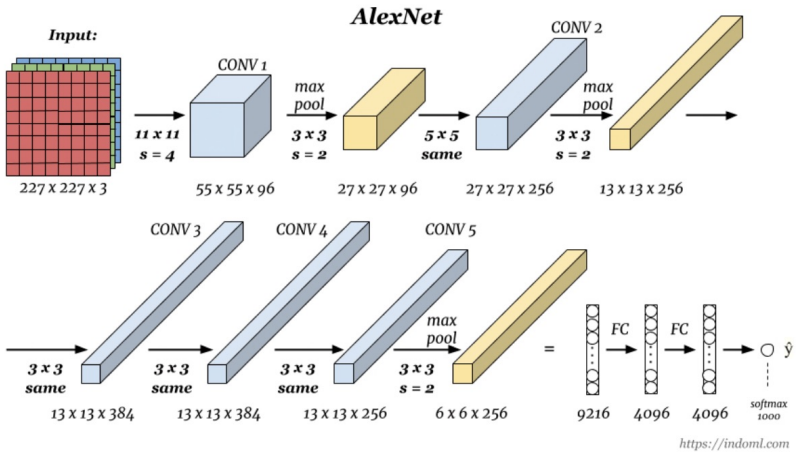
One example of classic networks is LeNet-5, from *Gradient-Based Learning Applied to Document Recognition* paper by Y. Lecun, L. Bottou, Y. Bengio and P. Haffner (1998):



- Number of parameters: ~ 60 thousands.

Classic Network: AlexNet

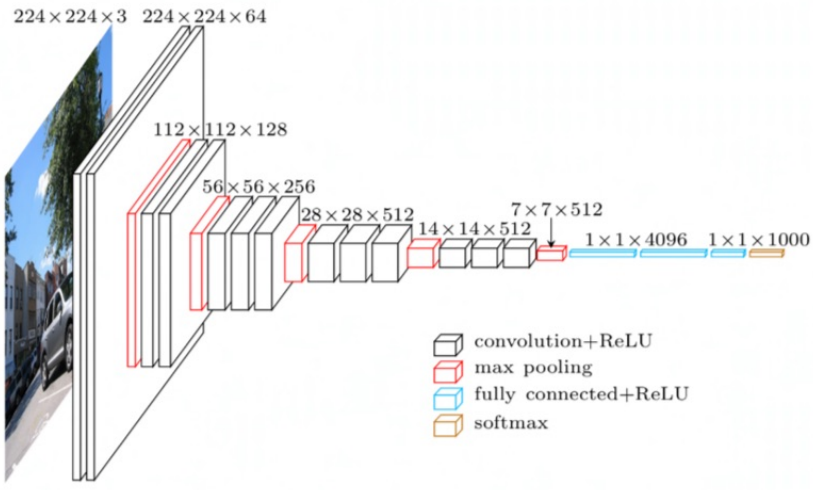
AlexNet is another classic CNN architecture from *ImageNet Classification with Deep Convolutional Neural Networks* paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever (2012).



- Number of parameters: ~ 60 millions

Classic Network: VGG-16

VGG-16 from *Very Deep Convolutional Networks for Large-Scale Image Recognition* paper by Karen Simonyan and Andrew Zisserman (2014). The number 16 refers to the fact that the network has 16 trainable layers (i.e. layers that have weights).

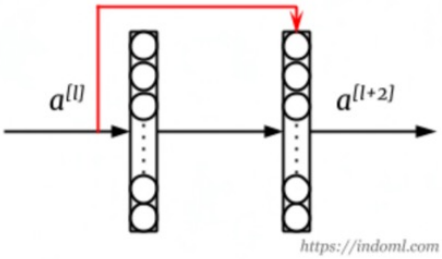


(image from **blog.heuritech.com**)

- Number of parameters: ~ 138 millions.
- The strength is in the simplicity: the dimension is halved and the depth is increased on every step (or stack of layers)

ResNet

The problem with deeper neural networks are they are harder to train and once the number of layers reach certain number, the training error starts to raise again. Deep networks are also harder to train due to exploding and vanishing gradients problem. ResNet (Residual Network), proposed by He at all in *Deep Residual Learning for Image Recognition paper* (2015), solves these problems by implementing skip connection where output from one layer is fed to layer deeper in the network:



In the image above, the skip connection is depicted by the red line. The activation $a^{[l+2]}$ is then calculated as:

$$z^{[l+2]} = W^{[l+2]}\, a^{[l+1]} + b^{[l+2]}$$

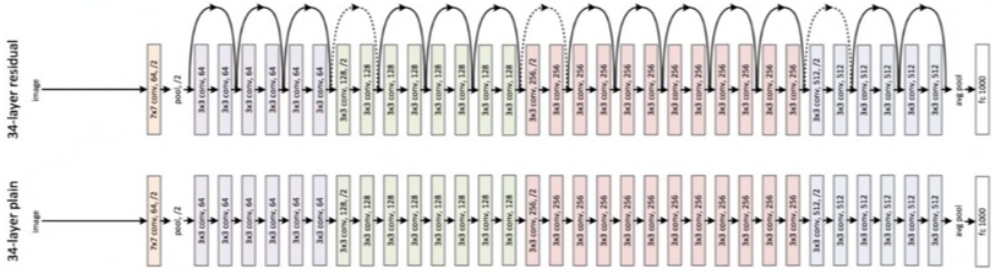
$$a^{[l+2]} = g^{[l+2]}(z^{[l+2]} + a^{[l]})$$

The advantages of ResNets are:

- performance doesn't degrade with very deep network
- cheaper to compute
- ability to train very very deep network

ResNet works because:

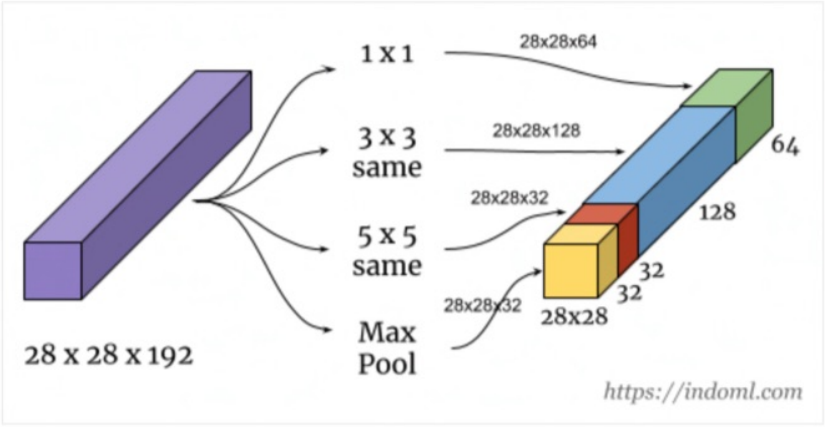
- identify function is easy for residual block to learn
- using a skip-connection helps the gradient to back-propagate and thus helps you to train deeper networks



Comparison of 34 layers ResNet with plain network (image from **euler.stat.yale.edu**)

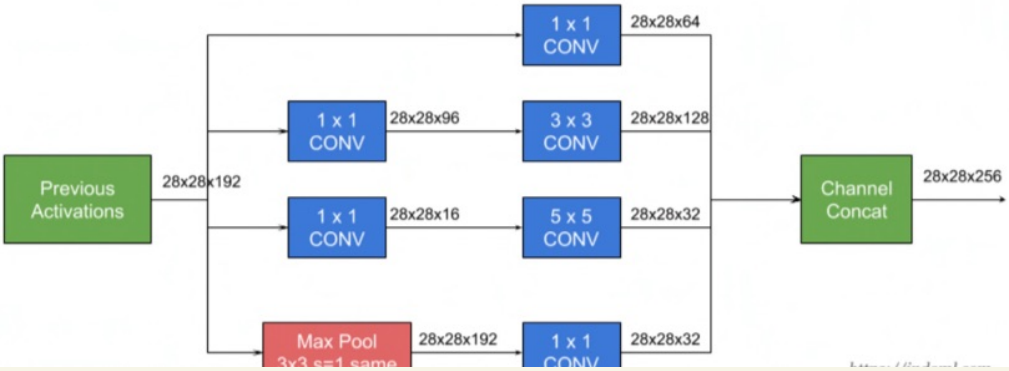
Inception

The motivation of the inception network is, rather than requiring us to pick the filter size manually, let the network decide what is best to put in a layer. We give it choices and hopefully it will pick up what is best to use in that layer:

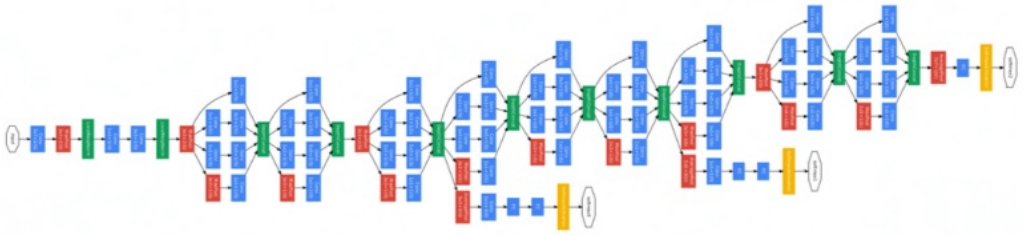


The problem with the above network is computation cost (e.g. for the 5 x 5 filter only, the computation cost is (28 x 28 x 32) x (5 x 5 x 192) = ~ 120 millions).

Using 1 x 1 convolution will reduce the computation to about 1/10 of that. With this idea, an inception module will look like this:



Below is an inception network called **GoogLeNet**, described in *Going Deeper with Convolutions paper* by Szegedy et all (2014), which has 9 inception modules:



GoogLeNet architecture (image **source**)

Data Augmentation

- Mirroring
- Random cropping
- Less common (perhaps due to their complexity):
  - Rotation
  - Shearing
  - Local warping
- Color shifting